ADSP-SC58x/ADSP-2158x SHARC+ Processor Hardware Reference

Revision 1.0, September 2017

Part Number
82-100129-01

,

ANALOG
DEVICES

Notices

Copyright Information

© 2017 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, CrossCore, EngineerZone, EZ-Board, EZ-KIT Lite, EZ-Extender, SHARC, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

Blackfin+, SHARC+, and EZ-KIT Mini are registered trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Contents

Introduction

Power and Clock Management (DPM/RCU/CGU/CDU) 1–2

System Interrupts and Triggers (SEC/TRU) 1–3

System Memory (L2CTL/DMC/SMC/OTPC/SMPU) 1–4

Direct Memory Access (DMA/MDMA/EMDMA/CRC) 1–7

Peripherals 1–9

- General-Purpose I/O (GPIO) Peripherals 1–9
- DAI/SRU Peripherals 1–14
- Dedicated Pin Peripherals 1–15

System Accelerators (FFT/FIR/IIR/HAE/SINC) 1–17

Security and Protection (SPU/PKTE/PKIC/PKA/TRNG) 1–19

Safety (WDOG/TMU/VMU) 1–20

Analog Subsystem (HADC) 1–20

System Debug (SCB/SWU/DBG/TAPC/CSPFT/STM) 1–20

ARM Cortex-A5 Sub-System

Cortex A5 Features 2–1

Functional Description 2–2

- A5 Block Diagram 2–2
- Control Co-Processor (CP15) 2–2
- L1 Cache 2–3
- Prefetch Unit (PFU) 2–3
- Memory-Management Unit (MMU) 2–3
- L2 Cache 2–4
- Floating-Point Unit (FPU) 2–5
- NeON 2–6
- Generic Interrupt Controller (GIC) 2–7

A5 Configurations 2–7

Clock Generation Unit (CGU)

CGU Features 3–1
CGU Functional Description 3–1
ADSP-SC58x CGU Register List 3–2
ADSP-SC58x CGU Interrupt List 3–2
ADSP-SC58x CGU Trigger List 3–3
CGU Definitions 3–3
CGU PLL Block Diagram 3–4
CGU Operating Modes 3–6
CGU Power-up Sequence 3–6
CGU Event Control 3–6
Oscillator Watchdog 3–7
CGU Programming Model 3–8
Configuring CGU Modes 3–8
Changing Clock Frequencies 3–9
Changing the PLL Clock Frequency 3–9
Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency 3–10
Changing the OCLK Frequency 3–11
Aligning All Clocks 3–11
Shutting Off CCLKn From Another Master 3–12
Valid Clock Multiplier Settings 3–12
PLL Bypass and PLL Disable 3–13
ADSP-SC5xx Specific Information 3–13
ADSP-SC58x CGU Register Descriptions 3–13
Core Clock Buffer Disable Register 3–15
Core Clock Buffer Status Register 3–16
CLKOUT Select Register 3–17
Control Register 3–19
Clocks Divisor Register 3–21

Oscillator Watchdog Register 3–24
PLL Control Register 3–26
Revision ID Register 3–28
System Clock Buffer Disable Register 3–29
System Clock Buffer Status Register 3–31
Status Register 3–33
Time Stamp Counter 32 LSB Register 3–36
Time Stamp Counter 32 MSB Register 3–37
Time Stamp Control Register 3–38
Time Stamp Counter Initial 32 LSB Value Register 3–39
Time Stamp Counter Initial MSB Value Register 3–40

Clock Distribution Unit (CDU)

CDU Features 4–1
CDU Functional Description 4–1
CDU Block Diagram 4–1
CDU Definitions 4–2
CDU Clock Configuration Options 4–3
CDU Programming Model 4–6
Changing the PLL and Clock Frequency 4–6
Changing the Clock Frequency 4–7
ADSP-SC58x CDU Register Descriptions 4–8
CDU Configuration 4–9
CLKIN Select 4–10
CDU Revision ID 4–11
CDU Status 4–12

Dynamic Power Management (DPM)

DPM Features 5–1
DPM Functional Description 5–1
ADSP-SC58x DPM Register List 5–1

DPM Definitions 5–1
DPM Operating Modes 5–2
Reset State 5–2
Full-on Mode 5–2
DPM Event Control 5–3
DPM Programming Model 5–3
ADSP-SC58x DPM Register Descriptions 5–5
Control Register 5–6
Peripherals Disable Register 0 5–7
Peripherals Disable Register 1 5–9
Revision ID 5–11
Status Register 5–12

Reset Control Unit (RCU)

RCU Features 6–1
RCU Functional Description 6–2
ADSP-SC58x RCU Register List 6–2
ADSP-SC58x RCU Trigger List 6–3
RCU Definitions 6–3
RCU Architectural Concepts 6–4
RCU Status and Error Signals 6–4
Resetting the ARM Core through Another Core or System Master 6–5
Resetting a SHARC+ Core Through Another Core 6–5
ADSP-SC58x RCU Register Descriptions 6–7
Boot Code Register 6–8
Core Reset Outputs Control Register 6–11
Core Reset Outputs Status Register 6–12
Control Register 6–13
Message Register 6–15
Message Clear Bits Register 6–19

Message Set Bits Register 6–20
System Interface Disable Register 6–21
System Interface Status Register 6–22
Status Register 6–23
Software Vector Register 0 6–25
Software Vector Register 1 6–26
Software Vector Register 2 6–27
SVECT Lock Register 6–28

System Event Controller (SEC) and Generic Interrupt Controller (GIC)

SEC Features 7–1
SEC Functional Description 7–2
ADSP-SC58x SEC Register List 7–2
ADSP-SC58x SEC Interrupt List 7–3
Combined SEC and GIC Interrupt List 7–3
SEC Definitions 7–12
SEC Block Diagram 7–13
SEC Fault Interface (SFI) 7–13
SEC Core Interface (SCI) 7–14
SEC Source Interface (SSI) 7–15
SEC Architectural Concepts 7–15
System Interrupt Acknowledge 7–15
System Interrupt Groups 7–16
System Interrupt Flow 7–16
System Interrupt Priorities 7–17
SEC Error 7–17
SEC Programming Model 7–17
Programming Concepts 7–18
Programming Examples 7–18
Fault Management Interface Programming Model 7–18
Configuring a System Source to Interrupt a Core 7–19

Core/SEC Handshaking Requirements to Ensure Proper Interrupt Handling 7–20

Configuring a System Source as a Fault 7–20

Configuring the WDOG Expiry Event to Issue a System Reset 7–21

SEC Programming Restrictions 7–21

ADSP-SC58x SEC Register Descriptions 7–22

SCI Active Register n 7–24

SCI Control Register n 7–25

SCI Group Mask Register n 7–27

SCI Priority Level Register n 7–29

SCI Priority Mask Register n 7–31

Core Pending Register n 7–32

SCI Source ID Register n 7–33

SCI Status Register n 7–34

Global End Register 7–36

Fault COP Period Register 7–37

Fault COP Period Current Register 7–38

Fault Control Register 7–39

Fault Delay Register 7–42

Fault Delay Current Register 7–43

Fault End Register 7–44

Fault Source ID Register 7–45

Fault System Reset Delay Register 7–46

Fault System Reset Delay Current Register 7–47

Fault Status Register 7–48

Global Control Register 7–50

Global Status Register 7–51

Global Raise Register 7–53

Source Control Register n 7–54

Source Status Register n 7–57

GIC Overview 7–58

GIC Functional Description 7–59

GIC Block Diagram 7–60

ADSP-SC58x GICDST Register List 7–60

ADSP-SC58x GICCPU Register List 7–60

ADSP-SC58x GICDST Register Descriptions 7–61

GIC Port 0 Enable 7–62

Software Generated Interrupt Priority Register 7–63

Shared Peripheral Interrupt Priority Register 7–64

Software Generated Interrupt Active Register 7–65

Software Generated Interrupt Control Register 7–66

Software Generated Interrupt Clear-Pending Register 7–68

Software Generated Interrupt Pending Set Register 7–69

Software Generated Interrupt Security Register 7–70

Shared Peripheral Interrupt Register 7–71

Shared Peripheral Interrupt Active Register 7–72

Shared Peripheral Interrupt Configuration Register 7–73

Shared Peripheral Interrupt Enable Clear Register 7–74

Shared Peripheral Interrupt Enable Set Register 7–75

Shared Peripheral Interrupt Pending Clear Register 7–76

Shared Peripheral Interrupt Pending Set Register 7–77

Shared Peripheral Interrupt Security Register 7–78

Shared Peripheral Interrupt Processor Targets Register 7–79

ADSP-SC58x GICCPU Register Descriptions 7–79

Aliased Binary Point Register (ICCABPR) 7–80

Binary Point Register (ICCBPR) 7–81

CPU Interface Control Register (ICCICR) 7–82

End of Interrupt Register (ICCEOIR) 7–83

Highest Pending Interrupt Register (ICCHPIR) 7–84

Interrupt Acknowledge Register (ICCIAR) 7–85

Priority Mask Register (ICCIPMR) 7–86

Running Priority Register (ICCRPR) 7–87

ADSP-SC58x GICDST Register Descriptions 7–87

GIC Port 0 Enable 7–89
Software Generated Interrupt Priority Register 7–90
Shared Peripheral Interrupt Priority Register 7–91
Software Generated Interrupt Active Register 7–92
Software Generated Interrupt Control Register 7–93
Software Generated Interrupt Clear-Pending Register 7–95
Software Generated Interrupt Pending Set Register 7–96
Software Generated Interrupt Security Register 7–97
Shared Peripheral Interrupt Register 7–98
Shared Peripheral Interrupt Active Register 7–99
Shared Peripheral Interrupt Configuration Register 7–100
Shared Peripheral Interrupt Enable Clear Register 7–101
Shared Peripheral Interrupt Enable Set Register 7–102
Shared Peripheral Interrupt Pending Clear Register 7–103
Shared Peripheral Interrupt Pending Set Register 7–104
Shared Peripheral Interrupt Security Register 7–105
Shared Peripheral Interrupt Processor Targets Register 7–106

Trigger Routing Unit (TRU)

TRU Features 8–1
TRU Functional Description 8–1
  ADSP-SC58x TRU Register List 8–2
  ADSP-SC58x TRU Interrupt List 8–2
  ADSP-SC58x Trigger List 8–3
  TRU Definitions 8–12
  TRU Block Diagram 8–12
TRU Architectural Concepts 8–13
TRU Programming Model 8–13
  Programming Concepts 8–13
  Programming Examples 8–14

Configuring a Simple Trigger Sequence........8–14
Toggle a GPIO on Timer Expiry Event........8–14
TRU Event Control........8–14
TRU Status and Error Signals........8–14
ADSP-SC58x TRU Register Descriptions........8–14
Error Address Register........8–16
Global Control Register........8–17
Master Trigger Register........8–19
Slave Select Register........8–20
Status Information Register........8–21

L2 System Memory

L2 System Memory Features........9–1
L2 System Memory Functional Description........9–1
ADSP-SC58x L2CTL Interrupt List........9–2
ADSP-SC58x L2CTL Register List........9–2
L2 System Memory Block Diagram........9–2
L2 System Memory Architectural Concepts........9–3
Access Characteristics........9–3
Read/Write Latency and Throughput........9–3
L2 Memory Controller Block Diagram (Instance)........9–4
Arbitration and Priority........9–4
Data Integrity........9–6
ECC Algorithm........9–6
ECC Hardware Control........9–8
ECC Error Management........9–9
Memory Refresh........9–9
L2 System Memory Event Control........9–10
ECC Error Interrupt........9–10
ADSP-SC58x L2CTL Register Descriptions........9–10
Control Register........9–12

Error Type 0 Address Register 9–16
Error Type 1 Address Register 9–17
ECC Error Address 0 Register 9–18
ECC Error Address 1 Register 9–19
ECC Error Address 2 Register 9–20
ECC Error Address 3 Register 9–21
ECC Error Address 4 Register 9–22
ECC Error Address 5 Register 9–23
ECC Error Address 6 Register 9–24
ECC Error Address 7 Register 9–25
Error Type 0 Register 9–26
Error Type 1 Register 9–28
Refresh Address Register 9–29
Read Priority Count Register 9–30
Status Register 9–31
Write Priority Count Register 9–34

Dynamic Memory Controller (DMC)

DMC Features 10–1
  Feature Exclusions 10–3
DMC Functional Description 10–3
  ADSP-SC58x DMC Register List 10–3
  ADSP-SC58x DMC Register List 10–4
  Protocol Controller 10–5
  Efficiency Controller 10–5
    Page-Based Scheduling 10–5
    Same Master Transaction Scheduling 10–5
    DMC Read Data Buffer 10–6
    Closed Page Per Bank 10–6
    SCB ID-Based Priority 10–6
    Delaying up to Eight Auto-Refresh Commands 10–7

    - Page and Bank Interleaving 10–7
  - System Crossbar Slave Interface 10–7
    - Read/Write Command and Data Buffers 10–8
  - Peripheral Bus Slave Interface 10–8
  - Architectural Concepts 10–8
    - Controller On Die Termination (ODT) 10–9
    - Mode Register Set and Extended Mode Register Set Command 10–9
    - DDR3 Reset Functionality 10–9
    - DDR3 SDRAM Organization 10–9
    - DDR2 SDRAM Organization 10–10
    - LPDDR SDRAM Organization 10–11
    - DMC Clocking 10–12
    - DMC DMA 10–12
- DMC Operating Modes 10–12
  - Self-Refresh Mode 10–13
- DMC Event Control 10–14
- DMC Programming Model 10–14
  - PHY DLL Calibration 10–17
  - DDR2 OCD Calibration 10–17
  - DDR3 ZQ Calibration Short CMD 10–17
  - DDR3 ZQ Calibration Long CMD 10–17
  - On Die Termination (DDR2/DDR3) 10–18
  - Output Driver Impedance 10–18
  - Initializing the DMC (ADSP-SC58x) 10–19
- ADSP-SC58x DMC Register Descriptions 10–22
  - Configuration Register 10–24
  - Controller to PHY Interface Register 10–26
  - Control Register 10–27
  - DLL Control Register 10–31
  - Data Calibration Address Register 10–32

Data Calibration Data 0 Register 10–33
Data Calibration Data 1 Register 10–34
Efficiency Control Register 10–35
Shadow EMR1 DDR2 Register 10–39
Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR) 10–41
Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3) 10–43
Shadow MR1 Register (DDR3) 10–46
Shadow MR2 Register (DDR3) 10–49
Mask (Mode Register Shadow) Register 10–51
Priority ID Register 1 10–53
Priority ID Register 2 10–54
Priority ID Mask Register 1 10–55
Priority ID Mask Register 2 10–56
DMC Read Data Buffer ID Register 1 10–57
DMC Read Data Buffer ID Register 2 10–58
DMC Read Data Buffer Mask Register 1 10–59
DMC Read Data Buffer Mask Register 2 10–60
Status Register 10–61
Timing 0 Register 10–64
Timing 1 Register 10–66
Timing 2 Register 10–67
ADSP-SC58x DMC Register Descriptions 10–68
Calibration PAD Control 0 Register 10–69
Calibration PAD Control 2 Register 10–70
PHY Control 0 Register 10–71
PHY Control 1 Register 10–72
PHY Control 2 Register 10–73
PHY Control 3 Register 10–74
PHY Control 4 Register 10–75

Static Memory Controller (SMC)

SMC Features 11–1

SMC Definitions 11–1

SMC Functional Description 11–3

ADSP-SC58x SMC Register List 11–4

SMC Architectural Concepts 11–4

Avoiding Bus Contention 11–5

ARDY Input Control 11–6

SMC Operating Modes 11–6

Asynchronous Flash Mode 11–7

Asynchronous Page Mode 11–7

SMC Event Control 11–7

SMC Programmable Timing Characteristics 11–7

Asynchronous SRAM Reads and Writes 11–7

Asynchronous SRAM Reads with IDLE Transition Cycles Inserted 11–9

High-Speed Asynchronous SRAM Read Burst 11–9

High-Speed Asynchronous SRAM Writes 11–10

Asynchronous SRAM Reads with ARDY 11–11

Asynchronous Flash Reads 11–12

Asynchronous Flash Writes 11–14

Asynchronous Flash Page Mode Reads 11–15

Asynchronous FIFO Reads and Writes 11–15

SMC Programming Model 11–17

ADSP-SC58x SMC Register Descriptions 11–17

Bank 0 Control Register 11–19

Bank 0 Extended Timing Register 11–22

Bank 0 Timing Register 11–24

Bank 1 Control Register 11–26

Bank 1 Extended Timing Register 11–29

Bank 1 Timing Register 11–31

Bank 2 Control Register 11–33
Bank 2 Extended Timing Register 11–36
Bank 2 Timing Register 11–38
Bank 3 Control Register 11–40
Bank 3 Extended Timing Register 11–43
Bank 3 Timing Register 11–45

One-Time Programmable Memory Controller (OTPC)

OTPC Features 12–1
Error Correction 12–1
OTP Layout 12–1
OTPC Event Control 12–2
OTPC Interrupt Signals 12–3
OTPC Status and Error Signals 12–3
OTP API Overview 12–3
OTP Programming 12–3
OTP Program 12–3
OTP Reading 12–4
OTP Get Field 12–4
OTP Counters 12–6
Lock API 12–6
ADSP-SC58x OTPC Interrupt List 12–7
ADSP-SC58x OTPC Register Descriptions 12–7
OTP Security State Register 12–8
OTP Status Register 12–9

System Memory Protection Unit (SMPU)

SMPU Features 13–1
SMPU Functional Description 13–2
ADSP-SC58x SMPU Register List 13–3
ADSP-SC58x SMPU Interrupts 13–4

Memory Writes 13–4

Memory Reads 13–4

ID Comparison 13–5

Memory Region 13–8

SMPU Definitions 13–9

SMPU Block Diagram 13–10

SMPU Architectural Concepts 13–10

SMPU Operating Modes 13–11

SMPU Interrupt Signals 13–11

SMPU Status and Error Signals 13–12

SMPU Programming Example 13–12

ADSP-SC58x SMPU Register Descriptions 13–13

Bus Error Address Register 13–15

Bus Error Details Register 13–16

SMPU Control Register 13–17

Exclusive Access IDn Address 13–19

Exclusive Access Status 13–20

Interrupt Address Register 13–21

Interrupt Details Register 13–22

Region n Address Register 13–23

Region n Control Register 13–24

SMPU Revision ID Register 13–27

Region n ID A Register 13–28

Region n ID B Register 13–29

Region n ID Mask A Register 13–30

Region n ID Mask B Register 13–31

SMPU Control Secure Accesses Register 13–32

Region n Control Secure Accesses Register 13–34

SMPU Status Register 13–35

General-Purpose Ports (PORT)

PORT Features 14–2

PORT Functional Description 14–2

ADSP-SC58x PORT Register List 14–3

ADSP-SC58x PINT Register List 14–3

ADSP-SC58x PINT Interrupt List 14–4

ADSP-SC58x PINT Trigger List 14–4

ADSP-SC58x PADS Register List 14–5

PORT Architectural Concepts 14–5

Internal Interfaces 14–5

External Interfaces 14–5

GPIO Functionality 14–5

Input Mode 14–5

Output Mode 14–6

Open-Drain Mode 14–6

Port Multiplexing Control 14–6

PORT Event Control 14–7

PORT Interrupt Signals 14–7

PORT Programming Model 14–9

ADSP-SC58x PORT Register Descriptions 14–12

Port x GPIO Data Register 14–14

Port x GPIO Data Clear Register 14–18

Port x GPIO Data Set Register 14–22

Port x GPIO Output Toggle Register 14–25

Port x GPIO Direction Register 14–28

Port x GPIO Direction Clear Register 14–32

Port x GPIO Direction Set Register 14–36

Port x Function Enable Register 14–39

Port x Function Enable Clear Register 14–42

Port x Function Enable Set Register 14–45

Port x GPIO Input Enable Register 14–48
Port x GPIO Input Enable Clear Register 14–51
Port x GPIO Input Enable Set Register 14–54
Port x GPIO Lock Register 14–57
Port x Multiplexer Control Register 14–59
Port x GPIO Polarity Invert Register 14–61
Port x GPIO Polarity Invert Clear Register 14–65
Port x GPIO Polarity Invert Set Register 14–68
ADSP-SC58x PINT Register Descriptions 14–70
PINT Assign Register 14–72
PINT Edge Clear Register 14–73
PINT Edge Set Register 14–76
PINT Invert Clear Register 14–79
PINT Invert Set Register 14–82
PINT Latch Register 14–85
PINT Mask Clear Register 14–89
PINT Mask Set Register 14–92
PINT Pin State Register 14–95
PINT Request Register 14–99
ADSP-SC58x PADS Register Descriptions 14–103
DAI0 Port Input Enable Control Register 14–104
DAI1 Port Input Enable Control Register 14–105
Peripheral PAD Configuration0 Register 14–106

Link Port (LP)

LP Features 15–1
LP Functional Description 15–1
ADSP-SC58x LP Register List 15–1
ADSP-SC58x LP Interrupt List 15–2
ADSP-SC58x LP Trigger List 15–2

ADSP-SC58x LP DMA Channel List 15–3

Block Diagram..... 15–3

External Connections 15–3

Internal Blocks 15–4

Architectural Concepts 15–4

Link Port Protocol..... 15–4

FIFO Buffers 15–7

Handshake for Link Port Enable Process 15–9

Clocking 15–10

Multi-Processor Connectivity 15–10

LP Operating Modes 15–12

LP Data Transfer Modes..... 15–12

Core Data Transfers 15–12

DMA Data Transfers..... 15–12

LP Event Control..... 15–13

Interrupt Signals 15–13

Enabling Link Port Interrupts 15–13

Status and Error Signals..... 15–14

LP Programming Model 15–14

Setting Up a DMA Transmit Operation 15–14

Setting Up a DMA Receive Operation..... 15–15

Setting Up a Core Transmit Operation..... 15–16

Setting Up a Core Receive Operation 15–16

ADSP-SC58x LP Register Descriptions 15–17

Control Register 15–18

Clock Divider Value Register 15–20

Receive Buffer Register 15–21

Status Register 15–22

Transmit Buffer Register 15–24

Shadow Input Transmit Buffer Register 15–25

Shadow Output Transmit Buffer Register 15–26

Serial Peripheral Interface (SPI)

SPI Features 16–1
SPI Functional Description 16–2
ADSP-SC58x SPI Register List 16–2
ADSP-SC58x SPI Interrupt List 16–3
ADSP-SC58x SPI Trigger List 16–4
ADSP-SC58x SPI DMA Channel List 16–5
SPI Block Diagram 16–5
Transfer Protocol 16–6
Clock Considerations 16–7
Controlling Delay Between Frames 16–8
Flow Control 16–9
Slave Select Operation 16–10
Beginning and Ending a Non-DMA SPI Transfer 16–11
Transmit Operation in Non-DMA Mode 16–12
Receive Operation in Non-DMA Mode 16–12
Dual I/O Mode 16–12
Quad I/O Mode (SPI2 only) 16–13
Fast Mode 16–14
Memory-Mapped Mode (SPI2 only) 16–15
Memory-Mapped Description of Operation 16–16
Memory-Mapped Architectural Concepts 16–17
Memory-Mapped Read Accesses 16–19
Memory-Mapped High-Performance Features 16–23
Merged Read Accesses 16–23
Wrap Around Accesses 16–23
Execute-In-Place (XIP, SPI2 only) 16–24
Memory-Mapped Mode Error Status Bits 16–25
Memory-Mapped Programming Guidelines 16–25

SPI Interrupt Signals 16–28

Data Interrupts 16–28

Status Interrupts 16–29

Error Conditions 16–29

SPI Programming Concepts 16–30

Master Operation in Non-DMA Modes 16–30

Slave Operation in Non-DMA Modes 16–31

Configuring DMA Master Mode 16–32

Configuring DMA Slave Mode Operation 16–33

ADSP-SC58x SPI Register Descriptions 16–34

Clock Rate Register 16–36

Control Register 16–37

Delay Register 16–43

Masked Interrupt Condition Register 16–44

Masked Interrupt Clear Register 16–46

Interrupt Mask Register 16–49

Interrupt Mask Clear Register 16–51

Interrupt Mask Set Register 16–54

Memory Mapped Read Header (Only on SPI2) 16–57

SPI Memory Top Address (Only on SPI2) 16–61

Receive FIFO Data Register 16–62

Received Word Count Register 16–63

Received Word Count Reload Register 16–64

Receive Control Register 16–65

Slave Select Register 16–68

Status Register 16–71

Transmit FIFO Data Register 16–76

Transmitted Word Count Register 16–77

Transmitted Word Count Reload Register 16–78

Transmit Control Register 16–79

Universal Asynchronous Receiver/Transmitter (UART)

UART Features 17–1
UART Functional Description 17–2
ADSP-SC58x UART Register List 17–2
ADSP-SC58x UART Interrupt List 17–3
ADSP-SC58x UART Trigger List 17–4
ADSP-SC58x UART DMA Channel List 17–4
UART Block Diagram 17–5
UART Architectural Concepts 17–5
Internal Interface 17–5
External Interface 17–6
Hardware Flow Control 17–6
Bit Rate Generation 17–6
Autobaud Detection 17–7
UART Debug Features 17–9
UART Operating Modes 17–9
UART Mode 17–10
IrDA SIR Mode 17–10
Multi-Drop Bus Mode 17–10
UART Data Transfer Modes 17–12
UART Mode Transmit Operation (Core) 17–12
UART Mode LIN Break Command 17–12
UART Mode Receive Operation (Core) 17–13
IrDA Transmit Operation 17–14
IrDA Receive Operation 17–14
MDB Transmit Operation 17–15
MDB Receive Operation 17–16
DMA Mode 17–16
Mixing DMA and Core Modes 17–17
Setting Up Hardware Flow Control 17–17
UART Event Control 17–18

- Interrupt Masks 17–18
- Interrupt Servicing 17–18
- Transmit Interrupts 17–19
- Receive Interrupts 17–20
- Status Interrupts 17–21
- Multi-Drop Bus Events 17–22

UART Programming Model 17–22

- Detecting Autobaud 17–22
- Using Common Initialization Steps 17–23
- Using Core Transfers 17–23
- Using DMA Transfers 17–23
- Using Interrupts 17–23
- Setting Up Hardware Flow Control 17–23

ADSP-SC58x UART Register Descriptions 17–24

- Clock Rate Register 17–25
- Control Register 17–26
- Interrupt Mask Register 17–32
- Interrupt Mask Clear Register 17–36
- Interrupt Mask Set Register 17–38
- Receive Buffer Register 17–40
- Receive Shift Register 17–41
- Receive Counter Register 17–42
- Scratch Register 17–43
- Status Register 17–44
- Transmit Address/Insert Pulse Register 17–49
- Transmit Hold Register 17–50
- Transmit Shift Register 17–51
- Transmit Counter Register 17–52

Enhanced Parallel Peripheral Interface (EPPI)

EPPI Features 18–1

EPPI Functional Description 18–2

ADSP-SC58x EPPI Register List 18–3

ADSP-SC58x EPPI Interrupt List 18–3

ADSP-SC58x EPPI Trigger List 18–4

RGB Data Formats 18–4

Data Clipping 18–4

Data Mirroring 18–5

Windowing 18–6

Preamble, Blanking and Stripping Support 18–6

EPPI Definitions 18–7

EPPI Block Diagram 18–8

EPPI Architectural Concepts 18–8

EPPI Interface 18–8

Reset Operation 18–10

Frame Sync Polarity and Sampling Edge 18–10

Direct Memory Access (DMA) 18–11

EPPI Clock 18–12

EPPI Operating Modes 18–13

ITU-R 656 Modes 18–13

ITU-R 656 Background 18–13

ITU-R 656 Input Modes 18–16

Entire Field 18–17

Active Video 18–17

Vertical Blanking Interval (VBI) 18–17

ITU-R 656 Output in General-Purpose Transmit Modes 18–18

Frame Synchronization in ITU-R 656 Modes 18–19

General-Purpose EPPI Modes 18–20

General-Purpose 0 Frame Sync Mode 18–20

General-Purpose 1 Frame Sync Mode 18–20

General-Purpose 2 Frame Sync Mode 18–21

Data Enable in General-Purpose 2 Frame Sync Transmit Mode 18–21

General-Purpose 3 Frame Sync Mode 18–21

Supported Data Formats 18–22

Receive Data Formats 18–22

Transmit Data Formats 18–24

Data Transfer Modes 18–25

Data Packing for Receive Modes 18–25

Data Packing for Transmit Modes 18–26

Sign-Extended and Zero-Filled Data 18–26

Split Receive Modes 18–26

Split Transmit Modes 18–27

Clock Gating 18–27

Support for Delayed Start of EPPI Frame Syncs 18–27

Ignoring Premature External Frame Syncs for Data Consistency 18–28

EPPI Event Control 18–28

EPPI Status, Error, and Interrupt Signals 18–29

Frame and Line Track Errors 18–29

Line Track Errors 18–29

Frame Track Errors 18–29

Preamble Error Not Corrected Error 18–30

EPPI Programming Model 18–30

Receiving ITU-R 656 Frames 18–30

Transmitting ITU-R 656 Frames in GP Transmit Modes 18–30

Configuring Transfers in GP 0 FS Mode 18–31

Configuring Transfers in GP 1 FS Mode 18–31

Configuring Transfers in GP 2 FS Mode 18–32

Configuring Transfers in GP 3 FS Mode 18–33

Configuring the EPPI to Use the Windowing Feature 18–33

EPPI Mode Configuration 18–34

Configuring 8-Bit Receive Mode 18–34

Configuring 10/12/14-Bit Receive Modes 18–35

Configuring 16-Bit Receive Mode 18–37

Configuring 18-Bit Receive Mode 18–38

Configuring 8-Bit Split Receive Mode 18–39

Configuring 10/12/14/16-Bit Split Receive Mode with SPLTWRD=0 18–42

Configuring 16-Bit Split Receive Mode with SPLTWRD=1 18–43

Configuring 8-Bit Transmit Mode 18–44

Configuring 10/12/14-Bit Transmit Modes 18–45

Configuring 16-Bit Transmit Mode 18–45

Configuring 18-Bit Transmit Mode 18–46

Configuring 8-Bit Split Transmit Mode 18–46

Configuring 10/12/14/16-Bit Transmit Mode with SPLTWRD=0 18–49

Configuring 16-Bit Split Transmit Mode with SPLTWRD=1 18–51

EPPI Programming Concepts 18–52

ADSP-SC58x EPPI Register Descriptions 18–52

Clock Divide Register 18–54

Control Register 18–55

Control Register 2 Register 18–63

Clipping Register for EVEN (Luma) Data Register 18–64

Lines Per Frame Register 18–65

Frame Sync 1 Delay Value Register 18–66

FS1 Period Register / EPPI Active Samples Per Line Register 18–67

FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register 18–68

Frame Sync 2 Delay Value Register 18–69

FS2 Period Register / EPPI Active Lines Per Field Register 18–70

FS2 Width Register / EPPI Lines Of Vertical Blanking Register 18–71

Horizontal Transfer Count Register 18–72

Horizontal Delay Count Register 18–73

Interrupt Mask Register 18–74

Samples Per Line Register 18–76

Clipping Register for ODD (Chroma) Data Register 18–77

Status Register 18–78

Vertical Transfer Count Register 18–81
Vertical Delay Count Register 18–82

Pulse-Width Modulator (PWM)

PWM Features 19–1
Functional Description 19–1
ADSP-SC58x PWM Register List 19–2
ADSP-SC58x PWM Interrupt List 19–4
ADSP-SC58x PWM Trigger List 19–5
PWM Definitions 19–5
Architectural Concepts 19–6
Block Diagram 19–6
Timer Units 19–7
PWM Timer Period (PWM_TM) Registers 19–8
Timer Unit Operation 19–8
Phase Offset Control 19–10
Channel Timing Control Unit 19–13
Channel Control 19–13
Pulse Positioning and Duty Cycle Registers 19–13
Duty Cycle and Pulse Positioning Control 19–14
Channel Low Side Output Dependent Operation Mode and Dead Time 19–15
Channel High Side and Low Side Outputs, Independent Operation Mode 19–17
Switched Reluctance Motors Application 19–19
Switching Dead Time (PWM_DT) Register 19–20
Duty Cycle with Dead Time Control: Calculations for PULSEMODE 00 19–20
Special Consideration for PWM Operation in Over-Modulation 19–22
Gate Drive Unit 19–24
Output Control Feature Precedence 19–25
Operating Modes 19–25
Sync Operation Modes 19–25
External (Triggered) PWM Sync Generation 19–25

Output Disable and Cross-Over Modes 19–26

Brushless DC Motor (Electronically Commutated Motor) Control 19–27

Heightened-Precision Edge Placement 19–28

Sample Waveforms for High- and Low-Side with Precision Placement 19–30

Emulation Mode 19–32

Event Control 19–32

Trip Control Unit 19–33

Programming Model 19–34

Programming Model for Three-Phase AC Motor Control 19–34

PWM Initialization for Motor Control 19–36

PWM Enable for Motor Control 19–38

PWM Response to Sync Interrupt for Motor Control 19–38

PWM Disable (and Stop the Motor) for Motor Control 19–39

ADSP-SC58x PWM Register Descriptions 19–39

Channel A Control Register 19–43

Channel A-High Duty-0 Register 19–45

Channel A-High Heightened-Precision Duty-0 Register 19–46

Channel A-High Duty-1 Register 19–47

Channel A-High Heightened-Precision Duty-1 Register 19–48

Channel A-High Full Duty0 Register 19–49

Channel A-High Full Duty1 Register 19–50

Channel A-Low Duty-0 Register 19–51

Channel A-Low Heightened-Precision Duty-0 Register 19–52

Channel A-Low Duty-1 Register 19–53

Channel A-Low Heightened-Precision Duty-1 Register 19–54

Channel A-Low Full Duty0 Register 19–55

Channel A-Low Full Duty1 Register 19–56

Channel B Control Register 19–57

Channel B-High Duty-0 Register 19–59

Channel B-High Heightened-Precision Duty-0 Register 19–60

Channel B-High Duty-1 Register 19–61
Channel B-High Heightened-Precision Duty-1 Register 19–62
Channel B-High Full Duty0 Register 19–63
Channel B-High Full Duty1 Register 19–64
Channel B-Low Duty-0 Register 19–65
Channel B-Low Heightened-Precision Duty-0 Register 19–66
Channel B-Low Duty-1 Register 19–67
Channel B-Low Heightened-Precision Duty-1 Register 19–68
Channel B-Low Full Duty0 Register 19–69
Channel B-Low Full Duty1 Register 19–70
Channel C Control Register 19–71
Channel C-High Pulse Duty Register 0 19–73
Channel C-High Pulse Heightened-Precision Duty Register 0 19–74
Channel C-High Pulse Duty Register 1 19–75
Channel C-High Pulse Heightened-Precision Duty Register 1 19–76
Channel Configuration Register 19–77
Channel A Dead-time Register 19–84
Channel B Dead-time Register 19–85
Channel C Dead-time Register 19–86
Channel D Dead-time Register 19–87
Chop Configuration Register 19–88
Channel C-High Full Duty0 Register 19–89
Channel C-High Full Duty1 Register 19–90
Channel C-Low Pulse Duty Register 0 19–91
Channel C-Low Pulse Duty Register 1 19–92
Channel C-Low Duty-1 Register 19–93
Channel C-Low Heightened-Precision Duty-1 Register 19–94
Channel C-Low Full Duty0 Register 19–95
Channel C-Low Full Duty1 Register 19–96
Control Register 19–97

Channel D Control Register 19–101
Channel D-High Duty-0 Register 19–103
Channel D-High Pulse Heightened-Precision Duty Register 0 19–104
Channel D-High Pulse Duty Register 1 19–105
Channel D High Pulse Heightened-Precision Duty Register 1 19–106
Channel D-High Full Duty0 Register 19–107
Channel D-High Full Duty1 Register 19–108
Channel D-Low Pulse Duty Register 0 19–109
Channel D-Low Heightened-Precision Duty-0 Register 19–110
Channel D-Low Pulse Duty Register 1 19–111
Channel D-Low Heightened-Precision Duty-1 Register 19–112
Channel A Delay Register 19–113
Channel B Delay Register 19–114
Channel C Delay Register 19–115
Channel D Delay Register 19–116
Channel D-Low Full Duty0 Register 19–117
Channel D-Low Full Duty1 Register 19–118
Interrupt Latch Register 19–119
Interrupt Mask Register 19–121
Status Register 19–123
Sync Pulse Width Register 19–128
Timer 0 Period Register 19–129
Timer 1 Period Register 19–130
Timer 2 Period Register 19–131
Timer 3 Period Register 19–132
Timer 4 Period Register 19–133
Trip Configuration Register 19–134

General-Purpose Timer (TIMER)

GP Timer Features 20–1

ADSP-SC58x TIMER Register List 20–2

ADSP-SC58x TIMER Interrupt List 20–2

ADSP-SC58x TIMER Trigger List 20–3

Timer Block Diagram 20–4

Internal Interface 20–4

Internal Timer Connections 20–4

External Interface 20–5

GP Timer Operating Modes 20–5

General Operation 20–5

Period, Width and Delay Register Interaction 20–6

Single-Pulse PWMOUT Mode 20–7

Continuous PWMOUT Mode 20–7

Width Capture (WIDCAP) Mode 20–9

Width Capture Mode Overflow 20–12

Windowed Watchdog (WATCHDOG) Modes 20–14

Windowed Watchdog Width Mode 20–14

Windowed Watchdog Period Mode 20–16

Pin Interrupt (PININT) Mode 20–18

External Clock (EXTCLK) Mode 20–18

GP Timer Programming Concepts 20–19

Setting Up Constantly Changing Timer Conditions 20–19

Configuring, Enabling, and Disabling One or More Timers 20–20

Configuring Timer Data and Status Interrupts 20–20

Using the Timer Broadcast Feature 20–20

Timer Illegal States 20–21

Continuous PWMOUT Mode 20–22

Single Pulse PWMOUT Mode 20–23

WIDCAP Mode 20–23

EXTCLK Mode 20–23

WATCHDOG Events 20–24

ADSP-SC58x TIMER Register Descriptions 20–25
Broadcast Delay Register 20–26
Broadcast Period Register 20–27
Broadcast Width Register 20–28
Data Interrupt Latch Register 20–29
Data Interrupt Mask Register 20–30
Error Type Status Register 20–31
Run Register 20–33
Run Clear Register 20–34
Run Set Register 20–35
Status Interrupt Latch Register 20–36
Status Interrupt Mask Register 20–37
Stop Configuration Register 20–38
Stop Configuration Clear Register 20–39
Stop Configuration Set Register 20–40
Timer n Configuration Register 20–41
Timer n Counter Register 20–45
Timer n Delay Register 20–46
Timer n Period Register 20–47
Timer n Width Register 20–48
Trigger Slave Enable Register 20–49
Trigger Master Mask Register 20–50

Watchdog Timer (WDOG)

WDOG Features........ 21–1
WDOG Functional Description 21–2
ADSP-SC58x WDOG Register List........ 21–3
ADSP-SC58x WDOG Interrupt List 21–3
WDOG Block Diagram........ 21–3
Internal Interface 21–3

External Interface 21–4
ADSP-SC58x WDOG Register Descriptions 21–4
Count Register 21–5
Control Register 21–6
Watchdog Timer Status Register 21–7

Real Time Clock (RTC)

RTC Features 22–1
RTC Functional Description 22–1
ADSP-SC58x RTC Register List 22–2
ADSP-SC58x RTC Interrupt List 22–2
ADSP-SC58x RTC Trigger List 22–3
RTC Definitions 22–3
RTC Signal Descriptions 22–3
RTC Architectural Concepts 22–3
RTC Block Diagram 22–4
Power Supply Partitioning 22–4
Battery Life 22–5
Writes to the 1 Hz Registers 22–5
Reads From the 1 Hz Registers 22–6
RTC Operating Modes 22–6
Alarm 22–6
Day Alarm 22–6
Stopwatch 22–6
Digital Watch Mode 22–7
Calibration for Accuracy 22–7
Accuracy 22–8
RTC Event Control 22–8
RTC Status and Error Signals 22–9
RTC Programming Model 22–9
Power-Up, Power-Down and Reset 22–9

Register Access 22–10

ADSP-SC58x RTC Register Descriptions 22–11

RTC Alarm Register 22–12

RTC Clock Register 22–13

Interrupt Enable Register 22–14

RTC Initialization Register 22–16

RTC Initialization Status Register 22–17

RTC Status Register 22–18

RTC Stop Watch Register 22–21

General-Purpose Counter (CNT)

GP Counter Features 23–1

GP Counter Functional Description 23–2

ADSP-SC58x CNT Register List 23–2

ADSP-SC58x CNT Interrupt List 23–3

ADSP-SC58x CNT Trigger List 23–3

GP Counter Operating Modes 23–3

Quadrature Encoder Mode 23–4

Binary Encoder Mode 23–4

Up/Down Counter Mode 23–5

Direction Counter Mode 23–5

Timed Direction Mode 23–5

GP Counter Programming Model 23–5

GP Counter General Programming Flow 23–5

GP Counter Mode Configuration 23–6

Configuring GP Counter Push-Button Operation 23–6

Configuring Zero-Marker-Zeros-Counter Mode 23–6

Configuring Zero-Marker-Error Mode 23–6

Configuring Zero-Once Mode 23–7

Configuring Boundary Auto-Extend Mode 23–7

Configuring Boundary Capture Mode 23–7

Configuring Boundary Compare and Boundary Zero Modes 23–8
Configuring GP Counter Push-Button Operation 23–8
GP Counter Programming Concepts 23–8
CNT Input Noise Filtering 23–8
Capturing Counter Interval and CNT_CNTR Read Timing 23–9
Capturing Time Interval Between Successive Counter Events 23–11
GP Counter Event Control 23–11
Illegal Gray and Binary Code Events 23–12
Up/Down Count Events 23–12
Zero-Count Events 23–12
Overflow Events 23–12
Boundary Match Events 23–13
Zero Marker Events 23–13
ADSP-SC58x CNT Register Descriptions 23–13
Configuration Register 23–14
Command Register 23–17
Counter Register 23–20
Debounce Register 23–21
Interrupt Mask Register 23–23
Maximum Count Register 23–26
Minimum Count Register 23–27
Status Register 23–28

ADC Control Module (ACM)

ACM Features 24–2
ACM Functional Description 24–3
ADSP-SC58x ACM Register List 24–3
ADSP-SC58x ACM Interrupt List 24–4
ADSP-SC58x ACM Trigger List 24–4
ACM Event Handling Latency 24–5
ACM Timing Specifications 24–6

ACM External Pin Timing 24–6

ACM Architectural Concepts 24–10

Clocking 24–10

Block Diagram 24–10

Trigger Inputs 24–11

Timers 24–13

Event Register Pairs 24–14

Event Comparators Unit 24–14

Timing Generation Unit 24–15

Status Flags and Interrupts 24–15

Event Order Registers 24–16

ACM Operation 24–17

ACM Programming Concepts 24–18

Emulation Mode Use Case 24–20

ADSP-SC58x ACM Register Descriptions 24–22

Control Register 24–23

Event N Control Register 24–26

Event Complete Interrupt Mask Register 24–27

Event N Order Register 24–30

Event Complete Status Register 24–31

Event N Time Register 24–36

Missed Event Interrupt Mask Register 24–37

Missed Event Status Register 24–40

Status Register 24–44

Timing Configuration 0 Register 24–46

Timing Configuration 1 Register 24–47

Timer 0 Register 24–48

Timer 1 Register 24–49

Controller Area Network (CAN)

CAN Features 25–1

CAN Functional Description 25–2

  ADSP-SC58x CAN Register List 25–2

  ADSP-SC58x CAN Interrupt List 25–3

  External Interface 25–4

  Architectural Concepts 25–4

    Block Diagram 25–5

    Mailbox Control 25–6

    Protocol Fundamentals 25–7

CAN Operating Modes 25–8

  Data Transfer Modes 25–8

    Transmit Operations 25–8

      Retransmission 25–9

      Single-Shot Transmission 25–10

      Auto-Transmission 25–10

    Receive Operation 25–10

      Data Acceptance Filtering 25–12

    Watchdog Mode 25–12

    Time Stamps 25–13

    Remote Frame Handling 25–13

    Temporarily Disabling CAN Mailbox 25–14

  Bit Timing 25–15

  CAN Low Power Features 25–16

    Built-In Suspend Mode 25–16

    Built-In Sleep Mode 25–17

    Soft Reset 25–17

CAN Event Control 25–17

  CAN Interrupt Signals 25–17

    Mailbox Interrupts 25–17

    Global Interrupt 25–18

  Event Counter 25–20

  CAN Warnings and Errors 25–20

Programmable Warning Limits 25–20

Error Handling 25–20

Error Frames 25–21

Error Levels 25–22

CAN Debug and Test Modes 25–23

ADSP-SC58x CAN Register Descriptions 25–25

Abort Acknowledge 1 Register 25–28

Abort Acknowledge 2 Register 25–29

Acceptance Mask (H) Register 25–30

Acceptance Mask (L) Register 25–31

Error Counter Register 25–32

Clock Register 25–33

CAN Master Control Register 25–34

Debug Register 25–36

Error Status Register 25–38

Error Counter Warning Level Register 25–40

Global CAN Interrupt Flag Register 25–41

Global CAN Interrupt Mask Register 25–44

Global CAN Interrupt Status Register 25–46

Interrupt Pending Register 25–49

Mailbox Interrupt Mask 1 Register 25–51

Mailbox Interrupt Mask 2 Register 25–52

Mailbox Receive Interrupt Flag 1 Register 25–53

Mailbox Receive Interrupt Flag 2 Register 25–54

Temporary Mailbox Disable Register 25–55

Mailbox Transmit Interrupt Flag 1 Register 25–56

Mailbox Transmit Interrupt Flag 2 Register 25–57

Mailbox Word 0 Register 25–58

Mailbox Word 1 Register 25–59

Mailbox Word 2 Register 25–60

Mailbox Word 3 Register 25–61
Mailbox ID 0 Register 25–62
Mailbox ID 1 Register 25–63
Mailbox Length Register 25–64
Mailbox Time Stamp Register 25–65
Mailbox Configuration 1 Register 25–66
Mailbox Configuration 2 Register 25–67
Mailbox Direction 1 Register 25–68
Mailbox Direction 2 Register 25–69
Overwrite Protection/Single Shot Transmission 1 Register 25–70
Overwrite Protection/Single Shot Transmission 2 Register 25–71
Remote Frame Handling 1 Register 25–72
Remote Frame Handling 2 Register 25–73
Receive Message Lost 1 Register 25–74
Receive Message Lost 2 Register 25–75
Receive Message Pending 1 Register 25–76
Receive Message Pending 2 Register 25–77
Status Register 25–78
Transmission Acknowledge 1 Register 25–80
Transmission Acknowledge 2 Register 25–81
Timing Register 25–82
Transmission Request Reset 1 Register 25–83
Transmission Request Reset 2 Register 25–84
Transmission Request Set 1 Register 25–85
Transmission Request Set 2 Register 25–86
Universal Counter Configuration Mode Register 25–87
Universal Counter Register 25–89
Universal Counter Reload/Capture Register 25–90

Mobile Storage Interface (MSI)

MSI Features 26–1

MSI Functional Description 26–2

- ADSP-SC58x MSI Register List 26–2
- ADSP-SC58x MSI Interrupt List 26–4
- ADSP-SC58x MSI Trigger List 26–4
- MSI Block Diagram 26–4
- MSI Architectural Concepts 26–5
  - Bus Interface Unit (BIU) 26–6
    - Host Interface Unit (HIU) 26–6
    - Interrupt Controller Unit 26–6
    - Register Unit 26–6
    - FIFO Controller Unit 26–7
    - Power and Pull-up Control and Card Detection Unit 26–8
  - Internal Direct Memory Access Controller (IDMAC) 26–8
    - DMA Descriptors 26–8
    - Initialization 26–11
    - Host Bus Burst Access 26–11
    - Buffer Size Calculations 26–12
    - Data Transmit/Receive 26–12
    - Interrupts 26–13
    - Finite State Machine (FSM) 26–13
    - Abort Operation 26–14
    - FIFO Overflow and Underflow 26–15
  - Card Interface Unit 26–15
    - Command Path 26–16
    - Datapath 26–18
    - Auto-Stop 26–19
    - Non-Data Transfer Commands that Use Datapath 26–21
    - SDIO Interrupt Control 26–21
    - Clock Control 26–22

MSI Data Transfer Modes 26–23

- Data Transmit 26–23
- Data Receive 26–25
- Data Transfer Commands 26–27
- Transmission and Reception with Internal DMAC (IDMAC) 26–27

MSI Event Control 26–27
- MSI Status and Error Signals 26–28

MSI Programming Model 26–30
- MSI Programming Concepts 26–30
- Initializing the MSI 26–31
- Enumerating the Card Stack 26–32
- Identifying Card Types 26–33
- Programming Card Clocks 26–34
- Sending Non-Data Commands With or Without a Response Sequence 26–35
- Single-Block or Multiple-Block Read 26–36
- Single-Block or Multiple-Block Write 26–38
- Stream Reads and Writes 26–39
- Packed Commands 26–39
- Sending Stop or Abort in Middle of Transfer 26–40
- Suspend or Resume Sequence 26–41
- Read Wait Sequence 26–43
- Card Read Threshold 26–43

MSI Programming Model, Boot Operation 26–45
- Normal Boot Operation 26–45
  - Normal Boot Operation; eMMC 26–45
  - Normal Boot Operation; Removable MMC4.3, MMC4.4, and MMC4.41 Cards 26–47
- Alternate Boot Operation; eMMC 26–48
- Alternate Boot Operation; Removable MMC4.3 Card 26–51

ADSP-SC58x Product Specific Information 26–52

ADSP-SC58x MSI Register Descriptions 26–52
- Block Size Register 26–54

Current Buffer Descriptor Address Register **26–55**

Bus Mode Register **26–56**

Byte Count Register **26–58**

Card Detect Register **26–59**

Card Threshold Control Register **26–60**

Clock Divider Register **26–61**

Clock Enable Register **26–62**

Command Register **26–63**

Command Argument Register **26–69**

Control Register **26–70**

Card Type Register **26–73**

Descriptor List Base Address Register **26–74**

Debounce Count Register **26–75**

Current Host Descriptor Address Register **26–76**

Enable Phase Shift Register **26–77**

FIFO Threshold Watermark Register **26–78**

Internal DMA Interrupt Enable Register **26–79**

Internal DMA Status Register **26–81**

Interrupt Mask Register **26–84**

Raw Interrupt Status Register **26–88**

Masked Interrupt Status Register **26–92**

Poll Demand Register **26–96**

Response Register 0 .... **26–97**

Response Register 1 .... **26–98**

Response Register 2 .... **26–99**

Response Register 3 .... **26–100**

Status Register **26–101**

Transferred Host to BIU-FIFO Byte Count Register **26–104**

Transferred CIU Card Byte Count Register **26–105**

Timeout Register **26–106**

Universal Serial Bus (USB)

USB Features 27–1

USB Functional Description 27–2

USB Architectural Concepts 27–2

Multi-Point Support 27–3

On-Chip Bus Interfaces 27–4

FIFO Configuration 27–4

Clocking 27–5

UTMI Interface 27–5

ADSP-SC58x USB Register List 27–5

ADSP-SC58x USB Interrupt List 27–8

ADSP-SC58x USB Trigger List 27–9

USB Block Diagram 27–9

USB Definitions 27–9

USB References 27–11

USB Operating Modes 27–11

Peripheral Mode 27–12

Endpoint Setup 27–12

IN Transactions as a Peripheral 27–13

OUT Transactions as a Peripheral 27–14

High-Bandwidth Isochronous or Interrupt Transactions 27–15

High Bandwidth Isochronous or Interrupt IN Endpoints 27–16

High-Bandwidth Isochronous or Interrupt OUT Endpoints 27–17

Peripheral Transfer Work Flows 27–18

Control Transactions as a Peripheral 27–19

Write Requests 27–19

Read Requests 27–20

Zero Data Requests 27–21

Endpoint 0 States 27–22

Endpoint 0 Service Routine as Peripheral 27–22

Peripheral Mode, Bulk IN, Transfer Size Known 27–27

    - Peripheral Mode, Bulk IN, Transfer Size Unknown 27–27
    - Peripheral Mode, ISO IN, Small MaxPktSize 27–28
    - Peripheral Mode, ISO IN, Large MaxPktSize 27–28
    - Peripheral Mode, Bulk OUT, Transfer Size Known 27–29
    - Peripheral Mode, Bulk OUT, Transfer Size Unknown 27–29
    - Peripheral Mode, ISO OUT, Small MaxPktSize 27–30
    - Peripheral Mode, ISO OUT, Large MaxPktSize 27–30
  - Peripheral Mode Suspend 27–31
  - Start of Frame (SOF) Packets 27–31
  - Soft Connect/Soft Disconnect 27–31
  - Error Handling As a Peripheral 27–31
  - Stalls Issued to Control Transfers 27–32
  - Zero Length OUT Data Packets in Control Transfers 27–33
- Host Mode 27–33
  - Transaction Scheduling 27–33
  - Endpoint Setup and Data Transfer 27–34
  - Control Transaction as a Host 27–34
  - Set up Phase as a Host 27–34
  - IN Data Phase as a Host 27–35
  - OUT Data as a Host (Control) 27–36
  - IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase) 27–36
  - OUT Status Phase as a Host (Following IN Data Phase) 27–37
  - Host IN Transactions 27–38
  - Host OUT Transactions 27–38
  - Multi-Point Support 27–39
    - Allocating Devices to Endpoints 27–39
    - Multi-Point Operation 27–40
    - Multi-Point Bandwidth Considerations 27–41
  - Babble Interrupt 27–41
  - VBUS Events 27–41
    - Actions as an A Device 27–42
    - Actions as a B Device 27–42

Host Mode Reset 27–42

Host Mode Suspend 27–43

Suspending and Resuming the Controller 27–43

Suspend or Resume by Inactivity on the USB Bus (L0 to L2 State) in Peripheral Mode 27–43

Suspend or Resume by Inactivity on the USB Bus (L0 To L2 State) in Host Mode 27–44

Suspend or Resume by an LPM Transaction (L0 To L1 State) in Peripheral Mode 27–44

Suspend or Resume by an LPM Transaction (L0 to L1 State) in Host Mode 27–46

USB Event Control 27–47

Interrupt Signals 27–47

Interrupt Handling 27–48

Reset Signals 27–49

Reset in Peripheral Mode 27–49

USB Reset in Host Mode 27–50

USB Programming Model 27–50

Peripheral Mode Flow Charts 27–51

Host Mode Flow Charts 27–58

DMA Mode Flow Charts 27–62

OTG Session Request 27–66

Starting a Session 27–66

Detecting Activity 27–67

Host Negotiation Protocol 27–67

Data Transfer 27–68

Loading or Unloading Packets from Endpoints 27–68

DMA Master Channels 27–69

DMA Bus Cycles 27–70

Transferring Packets Using DMA 27–71

Individual Rx Endpoint Packet 27–71

Individual Tx Endpoint Packet 27–72

Multiple Rx Endpoint Packets 27–72

Multiple Tx Endpoint Packets 27–73

ADSP-SC58x USB Register Descriptions 27–74

Battery Charging Control Register 27–78

Host High-Speed Return to Normal Register 27–79

High-Speed Timeout Register 27–80

Chirp Timeout Register 27–81

Device Control Register 27–82

DMA Channel n Address Register 27–84

DMA Channel n Count Register 27–85

DMA Channel n Control Register 27–86

DMA Interrupt Register 27–89

EP0 Configuration Information Register 27–91

EP0 Number of Received Bytes Register 27–93

EP0 Configuration and Status (Host) Register 27–94

EP0 Configuration and Status (Peripheral) Register 27–98

EP0 NAK Limit Register 27–101

EP0 Connection Type Register 27–102

EP0 Configuration Information Register 27–103

EP0 Number of Received Bytes Register 27–105

EP0 Configuration and Status (Host) Register 27–106

EP0 Configuration and Status (Peripheral) Register 27–110

EP0 NAK Limit Register 27–113

EP0 Connection Type Register 27–114

Endpoint Information Register 27–115

EPn Number of Bytes Received Register 27–116

EPn Receive Configuration and Status (Host) Register 27–117

EPn Receive Configuration and Status (Peripheral) Register 27–122

EPn Receive Polling Interval Register 27–127

EPn Receive Maximum Packet Length Register 27–128

EPn Receive Type Register 27–129

EPn Transmit Configuration and Status (Host) Register 27–131

EPn Transmit Configuration and Status (Peripheral) Register 27–135

EPn Transmit Polling Interval Register 27–139
EPn Transmit Maximum Packet Length Register 27–140
EPn Transmit Type Register 27–141
EPn Number of Bytes Received Register 27–143
EPn Receive Configuration and Status (Host) Register 27–144
EPn Receive Configuration and Status (Peripheral) Register 27–149
EPn Receive Polling Interval Register 27–154
EPn Receive Maximum Packet Length Register 27–155
EPn Receive Type Register 27–156
EPn Transmit Configuration and Status (Host) Register 27–158
EPn Transmit Configuration and Status (Peripheral) Register 27–162
EPn Transmit Polling Interval Register 27–166
EPn Transmit Maximum Packet Length Register 27–167
EPn Transmit Type Register 27–168
Function Address Register 27–170
FIFO Byte (8-Bit) Register 27–171
FIFO Half-Word (16-Bit) Register 27–172
FIFO Word (32-Bit) Register 27–173
Frame Number Register 27–174
Full-Speed EOF 1 Register 27–175
High-Speed EOF 1 Register 27–176
ID Control 27–177
Common Interrupts Enable Register 27–178
Index Register 27–180
Receive Interrupt Register 27–181
Receive Interrupt Enable Register 27–184
Transmit Interrupt Register 27–187
Transmit Interrupt Enable Register 27–190
Common Interrupts Register 27–193
Link Information Register 27–195

LPM Attribute Register 27–196
LPM Control Register 27–197
LPM Function Address Register 27–199
LPM Interrupt Enable Register 27–200
LPM Interrupt Status Register 27–202
Low-Speed EOF 1 Register 27–205
MPn Receive Function Address Register 27–206
MPn Receive Hub Address Register 27–207
MPn Receive Hub Port Register 27–208
MPn Transmit Function Address Register 27–209
MPn Transmit Hub Address Register 27–210
MPn Transmit Hub Port Register 27–211
PHY Control Register 27–212
PLL and Oscillator Control Register 27–213
Power and Device Control Register 27–214
RAM Information Register 27–217
EPn Request Packet Count Register 27–218
Receive FIFO Address Register 27–219
Receive FIFO Size Register 27–220
Software Reset Register 27–222
Testmode Register 27–223
Transmit FIFO Address Register 27–224
Transmit FIFO Size Register 27–225
VBUS Control Register 27–227
VBUS Pulse Length Register 27–228

Media Local Bus (MLB)

Features 28–1
MLB Definitions 28–2
Clocking 28–3

Functional Description 28–3
  ADSP-SC58x MLB Register List 28–4
  ADSP-SC58x MLB Interrupt List 28–5
  Media LB Protocol 28–5
MLB Architectural Concepts 28–6
  MediaLB Block Diagram 28–6
  MediaLB Interface 28–7
  Routing Fabric 28–8
    Data Buffer RAM 28–8
    Channel Table RAM 28–8
      Address Mapping 28–8
      Channel Allocation Table 28–9
      Channel Set Up 28–10
      Channel Descriptor Tables 28–11
      AHB Descriptor Table (ADT) 28–15
  Interrupt Interface Block 28–19
Operating Modes 28–20
  Isochronous Data Exchange 28–20
  Asynchronous and Control Data Exchange 28–21
  Synchronous Data Exchange 28–22
Data Transfer 28–22
  DMA 28–23
Programming Model 28–23
  Channel Initialization 28–23
    Configure the Hardware 28–23
    Program the CAT and the CDT 28–23
    Program the ADT 28–24
  Service 28–25
    Servicing the DMA Channel Interrupts 28–26
    Servicing the MediaLB Status Interrupts 28–26
    Polling for MediaLB System Commands 28–27

ADSP-SC58x MLB Register Descriptions 28–27
Peripheral Channel Mask 0 Register 28–29
Peripheral Channel Mask 1 Register 28–30
Peripheral Channel Status 0 Register 28–31
Peripheral Channel Status 1 Register 28–32
Bus Control Register 28–33
MediaLB Control 0 Register 28–35
Control 1 Register 28–37
MLB Global Control Register 28–38
HBI Channel Busy 0 Register 28–39
HBI Channel Busy 1 Register 28–40
HBI Channel Error 0 Register 28–41
HBI Channel Error 1 Register 28–42
HBI Channel Mask 0 Register 28–43
HBI Channel Mask 1 Register 28–44
HBI Control Register 28–45
Memory Interface Address Register 28–46
Memory Interface Control Register 28–47
Memory Interface Control Data 0 Register 28–48
Memory Interface Control Data 1 Register 28–49
Memory Interface Control Data 2 Register 28–50
Memory Interface Control Data 3 Register 28–51
Memory Interface Control Data Write Enable 0 Register 28–52
Memory Interface Control Data Write Enable 1 Register 28–53
Memory Interface Control Data Write Enable 2 Register 28–54
Memory Interface Control Data Write Enable 3 Register 28–55
Interrupt Enable Register 28–56
Channel Status 0 Register 28–59
Channel Status 1 Register 28–60
System Data Register 28–61

System Status Register 28–62
MediaLB 6-pin Control 0 Register 28–64

PCI Express

PCIe Features 29–2
Functional Description 29–3
ADSP-SC58x PCIE Register List 29–3
ADSP-SC58x PCIE Interrupt List 29–10
ADSP-SC58x PCIE Trigger List 29–10
Definitions 29–10
Top-level Block Diagram and Description 29–12
Module-level Block Diagrams and Descriptions 29–13
Reset Clock and PHY Control Module (RSCKPHY) 29–14
Responding to a Warm or Hot Reset 29–15
Application Module (APP) 29–18
Architectural Concepts 29–20
Register Accesses (PCIe Link and SCB DBI Port) 29–20
SOC Address Space Accesses 29–21
PCIe Address Space Accesses 29–21
Message Signaled Interrupts (MSI) Controller 29–23
Internal Address Translation (iATU) 29–25
Embedded DMA Controller (eDMA) 29–27
Operations 29–29
Initialization 29–29
Port Configuration 29–29
BARs Disable Example 29–30
Link Training and Establishment 29–31
Downstream Device Enumeration by Root Complex 29–32
Host Software Writes to Bus Master Enable (BME), Memory Space Enable (MSE), and I/O Space Enable (ISE) Bits in PCI-Compatible Command Register 29–32
Configuration for Memory Read and Write Transactions 29–32

Inbound Programming Example (BAR Match Mode) 29–33
Inbound Programming Example (Address Match Mode) 29–34
Outbound Programming Example 29–35
Non DMA Transfers 29–36
Transmit Transaction Layer Packets (TLP) 29–36
Receive Transaction Layer Packets (TLP) 29–37
DMA Transfers 29–37
Read Transfer 29–38
Write Transfer 29–39
Flow Control 29–40
EP to EP Transactions 29–40
Gen2 5.0 GT/s Operation 29–40
Power Management 29–40
Power Management Software 29–40
Active State Power Management (ASPM) 29–42
Completion Timeout Ranges 29–42
PHY Registers Access 29–42
RCCKPHY 29–42
Event Control 29–42
Error Handling and Debug 29–42
System Cross Bar Master Interface to PCIe Link Error Mapping 29–42
PCIe Link Error Mapping to SCB Slave Interface 29–43
Messages 29–43
PCIe Interrupts 29–43
PCIe Interrupt Request 29–43
End Point 29–47
Root Complex 29–47
Local Interrupts 29–49
Reset Interrupt Request 29–50
DMA Interrupts 29–50
Programming Model 29–50

Enumeration 29–50
Enumeration Driver Example 29–51
DMA Transfers 29–52
DMA Write Transfer Example 29–52
DMA Read Transfer Example 29–53
Device Reset 29–53
Cold Reset 29–54
Hot Reset 29–54
ADSP-SC58x Product Specific Information 29–55
ADSP-SC58x PCIE Register Descriptions 29–56
Acknowledge Frequency and L0-L1 ASPM Control Register 29–64
Acknowledge Latency Timer and Replay Timer Register 29–67
Advanced Error Capabilities and Control Register 29–68
Advanced Error Reporting Extended Capability Header Register 29–69
Correctable Error Message Requester ID Register 29–70
Application Control Register 29–71
Diagnostic Control Register 29–75
Diagnostic Status Register 29–76
Fatal Error Message Requester ID Register 29–77
Application Interrupts Status Register 29–78
Non Fatal Error Message Requester ID Register 29–82
Power Management Acknowledge Message Requester ID Register 29–83
Power Management Event Message Requester ID Register 29–84
Power Management Turn Off Message Requester ID Register 29–85
Application Status Register 29–86
Unlock Message Requester ID Register 29–88
Vendor Message Requester ID Register 29–89
Vendor Message Header Bytes 8 to 11 Register 29–90
Vendor Message Header Bytes 12 to 15 Register 29–91
Auxiliary Clock Frequency Control Register 29–92

Peripheral and SCB Bridge Slave Error Response Register 29–93
Bus Order Manager Watchdog Off Register 29–94
Bus Multiple Outbound Decomposition SubReq Control Register 29–95
ACE Cache Coherency Control Register 1 29–96
ACE Cache Coherency Control Register 3 29–97
PCIe Capabilities, ID, Next Pointer Register 29–98
Correctable Error Mask Register 29–100
Correctable Error Status Register 29–101
Device Capabilities 2 Register 29–103
Device Control 2 and Status 2 Register 29–107
Device Control and Status Register 29–108
DMA Read Arbitration Weight Low Off Register 29–113
DMA Read Channel 1 and 0 IMWr Data Register 29–114
DMA Control 1 Read Channel Register 29–115
DMA Destination Address High Read Channel Register 29–118
DMA Destination Address Low Read Channel Register 29–119
DMA Read Done IMWr Address High Register 29–120
DMA Read Done IMWr Address Low Register 29–121
DMA Read Doorbell Register 29–122
DMA Read Engine Enable Register 29–123
DMA Read Error Status High Register 29–125
DMA Read Error Status Low Register 29–127
DMA Read Interrupt Clear Register 29–128
DMA Read Interrupt Mask Register 29–129
DMA Read Abort IMWr Address High Register 29–130
DMA Read Abort IMWr Address Low Register 29–131
DMA Read Interrupt Status Register 29–132
DMA LLP High Read Channel Register 29–133
DMA LLP Low Read Channel Register 29–134
DMA Read Linked List Error Enable Register 29–135

DMA SAR High Read Channel Register 29–136
DMA SAR Low Read Channel Register 29–137
DMA Transfer Size Read Channel Register 29–138
DMA Write Abort IMWr Address High Register 29–139
DMA Write Abort IMWr Address Low Register 29–140
DMA Write Channel Arbitration Weight Low Off Register 29–141
DMA Write Channel 1 and 0 IMWr Data Register 29–142
DMA Control 1 Write Channel Register 29–143
DMA Dar High Write Channel Register 29–146
DMA DAR Low Write Channel Register 29–147
DMA Write Done IMWr Interrupt Address High Register 29–148
DMA Write Done IMWr Address Low Register 29–149
DMA Write Doorbell Register 29–150
DMA Write Engine Enable Register 29–151
DMA Write Error Status Register 29–152
DMA Write Interrupt Clear Register 29–153
DMA Write Interrupt Mask Register 29–154
DMA Write Interrupt Status Register 29–155
DMA LLP High Write Channel Register 29–156
DMA LLP Low Write Channel Register 29–157
DMA Write Linked List Error Enable Register 29–158
DMA Write Posted Request Deadlock Timer Register 29–159
DMA SAR High Write Channel Register 29–160
DMA SAR Low Write Channel Register 29–161
DMA Transfer Size Write Channel Register 29–162
DMA Number of Channels Register 29–163
DMA Channel Context Index Register 29–164
Endpoint Base Address Mask Register 0 29–165
Endpoint Base Address Register 0 29–166
Endpoint Base Address Mask Register 1 29–167

Endpoint Base Address Register 1 29–168
Endpoint Base Address Mask Register 2 29–169
Endpoint Base Address Register 2 29–170
Endpoint Base Address Mask Register 3 29–171
Endpoint Base Address Register 3 29–172
Endpoint Base Address Mask Register 4 29–173
Endpoint Base Address Register 4 29–174
Endpoint Base Address Mask Register 5 29–175
End Point Base Address Register 5 29–176
Capability Pointer Register 29–177
Class Code and Revision ID Register 29–178
End Point Configuration Register 29–179
CardBus CIS Pointer Register 29–180
Device Capabilities Register 29–181
Device ID and Vendor ID Register 29–184
Interrupt Line and Pin Register 29–185
End Point Expansion ROM Base Address Register 29–186
Subsystem ID and Subsystem Vendor ID Register 29–187
Command and Status Register 29–188
Error Source Identification Register 29–192
Filter Mask 2 Register 29–193
Link Width and Speed Change Control Register 29–194
Header Log Register 0 29–197
Header Log Register 1 29–198
Header Log Register 2 29–199
Header Log Register 3 29–200
iATU Region Control 1 Inbound Register 29–201
iATU Region Control 1 Outbound Register 29–203
iATU Region Control 2 Inbound Register 29–205
iATU Region Control 2 Register Outbound 29–209

iATU Region Control 3 Register 29–211
iATU Region Control 3 Register 29–212
IATU Lower Base Inbound Address Register 29–213
IATU Lower Base Outbound Address Register 29–214
IATU Inbound Limit Address Register 29–215
IATU Outbound Limit Address Register 29–216
IATU Lower Target Address Inbound Register 29–217
IATU Lower Target Address Outbound Register 29–218
IATU Upper Base Address Inbound Register 29–219
IATU Upper Base Address Outbound Register 29–220
iATU Upper Target Address Inbound Register 29–221
iATU Upper Target Address Outbound Register 29–222
IATU View Port Register 29–223
Lane Skew Register 29–224
Link Capabilities 2 Register 29–226
Link Capabilities Register 29–227
Link Control 2 and Status 2 Register 29–232
Link Control and Status Register 29–236
DBI Read-Only Write Enable Register 29–239
MSI Capability ID, Next Pointer and Control Register 29–240
MSI Capability Offset Register 29–242
MSI Capability Offset Register 29–243
MSI Capability Offset Register 29–244
MSI Controller Lower Address Register 29–245
MSI Controller Upper Address Register 29–246
MSI Controller General-Purpose IO Register 29–247
MSI Controller Interrupt 0 Enable Register 29–248
MSI Controller Interrupt 1 Enable Register 29–249
MSI Controller Interrupt 2 Enable Register 29–250
MSI Controller Interrupt 3 Enable Register 29–251

MSI Controller Interrupt 4 Enable Register 29–252
MSI Controller Interrupt 5 Enable Register 29–253
MSI Controller Interrupt 6 Enable Register 29–254
MSI Controller Interrupt 7 Enable Register 29–255
MSI Controller Interrupt 0 Mask Register 29–256
MSI Controller Interrupt 1 Mask Register 29–257
MSI Controller Interrupt 2 Mask Register 29–258
MSI Controller Interrupt 3 Mask Register 29–259
MSI Controller Interrupt 4 Mask Register 29–260
MSI Controller Interrupt 5 Mask Register 29–261
MSI Controller Interrupt 6 Mask Register 29–262
MSI Controller Interrupt 7 Mask Register 29–263
MSI Controller Interrupt 0 Status Register 29–264
MSI Controller Interrupt 1 Status Register 29–265
MSI Controller Interrupt 2 Status Register 29–266
MSI Controller Interrupt 3 Status Register 29–267
MSI Controller Interrupt 4 Status Register 29–268
MSI Controller Interrupt 5 Status Register 29–269
MSI Controller Interrupt 6 Status Register 29–270
MSI Controller Interrupt 7 Status Register 29–271
TX De-emphasis Parameters Register 29–272
TX Launch Amplitude Register 29–273
Port Logic Link Control Register 29–274
Port Logic Debug0 Register 29–276
Port Logic Debug1 Register 29–279
Port Force Link Register 29–281
Port Control PHY Control Register 29–283
Port Control PHY Status Register 29–284
Power Management Capabilities Register 29–285
Power Management Control and Status Register 29–287

Queue Status Register 29–290
Interrupt Bridge Line and Pin Control Register 29–291
Capability Pointer Register. 29–294
Class Code and Revision ID Register 29–295
Root Complex Configuration Register 29–296
Root Control and Capabilities Register 29–297
Device Capabilities Register 29–299
Root Complex Error Command Register 29–301
Root Error Status Register 29–302
Device ID and Vendor ID Register 29–305
Root Complex I/O Base and Limit Upper 16 bits Register 29–306
Memory Base and Memory Limit Register 29–307
Prefetchable Memory Base and Limit Register 29–308
Prefetchable Base Upper 32 Bits Register 29–309
Prefetchable Limit Upper 32 Bits Register 29–310
Root Complex Expansion ROM Base Address Register 29–311
Secondary Status and I/O Base and Limit Register 29–312
Command and Status Register 29–315
Root Status Register 29–319
Latency Timer Register 29–320
Reset and Clock Control Register 29–321
Reset and Clock Status 29–323
Symbol Timer Filter 1 Off 29–325
Timer Control and Max Function Number Register 29–326
Transmit Completion FC Credit Status Register 29–327
Transmit Non-Posted FC Credit Status Register 29–328
Transmit Posted FC Credit Status Register 29–329
Uncorrectable Error Mask Register 29–330
Uncorrectable Error Severity Register 29–332
Uncorrectable Error Status Register 29–334

Segmented-Buffer VC0 Completion Receive Queue Control Register 29–336
Segmented-Buffer VC0 Non-Posted Receive Queue Control Register 29–337
Segmented-Buffer VC0 Posted Receive Queue Control Register 29–338
Virtual Channel Transmit Arbitration Register 1 29–340
Virtual Channel Transmit Arbitration Register 2 29–342
Vendor Specific DLLP Register 29–344

Two-Wire Interface (TWI)

TWI Features 30–1
TWI Functional Description 30–2
ADSP-SC58x TWI Register List 30–2
ADSP-SC58x TWI Interrupt List 30–2
TWI Block Diagram 30–3
External Interface 30–3
Serial Clock Signal (SCL) 30–3
Serial Data Signal (SDA) 30–4
Internal Interface 30–4
TWI Architectural Concepts 30–5
TWI Protocol 30–5
Clock Generation and Synchronization 30–6
Bus Arbitration 30–6
Start and Stop Conditions 30–7
General Call Support 30–7
Fast Mode 30–8
TWI Operating Modes 30–8
Repeated Start 30–8
Transmit Receive Repeated Start 30–8
Receive Transmit Repeated Start 30–9
Clock Stretching 30–9
Clock Stretching During FIFO Underflow 30–10
Clock Stretching During FIFO Overflow 30–10

Clock Stretching During Repeated Start 30–11

TWI Programming Model 30–12

General Setup 30–12

Slave Mode 30–13

Master Mode Program Flow 30–14

Master Mode Clock Setup 30–15

Master Mode Transmit 30–15

Master Mode Receive 30–16

ADSP-SC58x TWI Register Descriptions 30–17

SCL Clock Divider Register 30–18

Control Register 30–19

FIFO Control Register 30–21

FIFO Status Register 30–23

Interrupt Mask Register 30–24

Interrupt Status Register 30–26

Master Mode Address Register 30–29

Master Mode Control Registers 30–30

Master Mode Status Register 30–33

Rx Data Double-Byte Register 30–36

Rx Data Single-Byte Register 30–37

Slave Mode Address Register 30–38

Slave Mode Control Register 30–39

Slave Mode Status Register 30–41

Tx Data Double-Byte Register 30–42

Tx Data Single-Byte Register 30–43

Ethernet Media Access Controller (EMAC)

EMAC Features 31–1

EMAC Functional Description 31–3

ADSP-SC58x EMAC Register List 31–4

ADSP-SC58x EMAC Interrupt List 31–10

ADSP-SC58x EMAC Trigger List 31–11

EMAC Definitions 31–11

EMAC Block Diagram and Interfaces 31–12

EMAC CORE Subblocks 31–14

EMAC PHY Interface 31–16

RGMII Board Design Recommendations 31–17

Clock Sources 31–19

EMAC Architectural Concepts 31–20

EMAC Feature Summary 31–20

EMAC System Crossbar Interface (EMAC SCB) 31–20

Priority of SCB Requests 31–21

SCB Interface Programming Options 31–22

DMA Bursts Using the SCB Interface 31–23

SCB Bus Transaction Status 31–24

Fatal Bus Error 31–24

DMA Controller (EMAC DMA) 31–24

DMA Related Registers 31–25

DMA Descriptors 31–27

OWN Bit (Ownership) Semaphore 31–43

Application Data Buffer Alignment 31–44

Buffer Size Calculations 31–44

EMAC FIFO Layer (EMAC MFL) 31–44

FIFO Layer Transmit Path 31–45

FIFO Layer Receive Path 31–46

EMAC CORE 31–47

EMAC CORE Transmission Engine 31–49

Source Address, VLAN, and CRC Insertion, Replacement, or Deletion 31–53

EMAC CORE Reception Engine 31–55

EMAC Station Management Interface (SMI) 31–65

MDC Clock Frequency 31–66

SMI Write Operation 31–67

- SMI Read Operation 31–68
- EMAC Management Counters (MMC) 31–68
  - MMC Receive Interrupt Register 31–70
  - MMC Transmit Interrupt Register 31–70
  - MMC Receive Checksum Offload Interrupt Register 31–70
- EMAC Precision Time Protocol (PTP) Engine 31–70
  - IEEE1588 and the PTP Engine 31–70
  - Block Diagram 31–74
  - PTP Module Clock 31–75
  - Time Stamp Module 31–77
  - System Time 31–84
  - Target Time Trigger (Alarm) 31–87
  - Pulse-Per-Second (PPS) 31–87
  - PTP Interrupts 31–90
- Audio Video Data Transmission 31–90
  - Transmit Path Functions 31–91
  - Receive Path Functions 31–92
  - DMA Arbiter 31–93
  - Slot Number Function 31–94
  - Interrupts 31–94
  - Credit-Based Shaper Algorithm Functions 31–94
- Energy-Efficient Ethernet 31–96
  - Transmit Path Functions 31–96
  - Receive Path Functions 31–97
  - LPI Timers 31–99

EMAC Event Control 31–99

- EMAC Interrupt Signals 31–100
  - PHYINT Interrupt Signal 31–101

EMAC Programming Model 31–102

- EMAC Programming Steps 31–102
  - DMA Initialization 31–102
  - EMAC CORE Initialization 31–103

Performing Normal Transmit and Receive Operations 31–104

Stopping and Starting Transfers 31–104

Interrupts and Interrupt Service Routines 31–105

Enabling Checksum for Transmit and Receive 31–106

Programming the System Time Module 31–106

Programming the PTP for Frame Detection and Time Stamping 31–107

Programming for Auxiliary Time Stamps 31–108

Programming Fixed Pulse-Per-Second Output 31–108

Programming Flexible Pulse-Per-Second Output 31–108

EMAC Programming Concepts 31–109

IEEE 802.3 Ethernet Packet Structure 31–109

Frame Size Statistics for Application Software 31–110

Software Visualization of Programmable Packet Size 31–110

Ethernet Packet Structure in C 31–111

DMA Descriptor Implementation in C 31–111

PTP Header Structure in C 31–112

ADSP-SC58x EMAC Register Descriptions 31–112

MAC Address 0 High Register 31–120

MAC Address 0 Low Register 31–121

MAC Address 1 High Register 31–122

MAC Address 1 Low Register 31–123

Debug Register 31–124

DMA SCB Bus Mode Register 31–127

DMA SCB Status Register 31–129

DMA Bus Mode Register 31–130

DMA Interrupt Enable Register 31–133

DMA Missed Frame Register 31–136

DMA Operation Mode Register 31–137

DMA Rx Buffer Current Register 31–141

DMA Rx Descriptor List Address Register 31–142

DMA Rx Descriptor Current Register 31–143
DMA Rx Interrupt Watch Dog Register 31–144
DMA Rx Poll Demand register 31–145
DMA Status Register 31–146
DMA Tx Buffer Current Register 31–151
DMA Tx Descriptor List Address Register 31–152
DMA Tx Descriptor Current Register 31–153
DMA Tx Poll Demand Register 31–154
DMA Bus Mode Register 31–155
Channel 1 Credit Shaping Control Register 31–158
Channel 1 Average Traffic Transmitted Register 31–159
Channel 1 High Credit Value Register 31–160
Channel 1 Idle Slope Credit Value Register 31–161
Channel 1 Low Credit Value Register 31–162
Channel 1 Control Bits for Slot Function Register 31–163
Channel 1 Send Slope Credit Value Register 31–164
DMA Interrupt Enable Register 31–165
DMA Missed Frame Register 31–168
DMA Operation Mode Register 31–169
DMA Rx Buffer Current Register 31–173
DMA Rx Descriptor List Address Register 31–174
DMA Rx Descriptor Current Register 31–175
DMA Rx Interrupt Watch Dog Register 31–176
DMA Rx Poll Demand Register 31–177
DMA Status Register 31–178
DMA Tx Buffer Current Register 31–183
DMA Tx Descriptor List Address Register 31–184
DMA Tx Descriptor Current Register 31–185
DMA Tx Poll Demand Register 31–186
DMA Bus Mode Register 31–187

Channel 2 Credit Shaping Control Register 31–190
Channel 2 Avg Traffic Transmitted Status Register 31–191
Channel 2 High Credit Value Register 31–192
Channel 2 Idle Slope Credit Value Register 31–193
Channel 2 Low Credit Value Register 31–194
Channel 2 Control Bits for Slot Function Register 31–195
Channel 2 Send Slope Credit Value Register 31–196
DMA Interrupt Enable Register 31–197
DMA Missed Frame Register 31–200
DMA Operation Mode Register 31–201
DMA Rx Buffer Current Register 31–205
DMA Rx Descriptor List Address Register 31–206
DMA Rx Descriptor Current Register 31–207
DMA Rx Interrupt Watch Dog Register 31–208
DMA Rx Poll Demand register 31–209
DMA Status Register 31–210
DMA Tx Buffer Current Register 31–215
DMA Tx Descriptor List Address Register 31–216
DMA Tx Descriptor Current Register 31–217
DMA Tx Poll Demand Register 31–218
FLow Control Register 31–219
RGMII Control and Status Register 31–221
Hash Table High Register 31–222
Hash Table Low Register 31–223
Interrupt Mask Register 31–224
MMC IPC Rx Interrupt Mask Register 31–225
MMC IPC Rx Interrupt Register 31–231
Interrupt Status Register 31–236
Layer3 and Layer4 Control Register 31–238
Layer 3 Address0 Register 31–240

Layer 3 Address1 Register 31–241

Layer 3 Address2 Register 31–242

Layer 3 Address3 Register 31–243

Layer 4 Address Register 31–244

Low Power Idle Control and Status Register 31–245

Low Power Idle Timeout Register 31–247

MAC Configuration Register 31–248

MAC Rx Frame Filter Register 31–253

AV MAC Control Register 31–256

MMC Control Register 31–258

MMC Rx Interrupt Mask Register 31–260

MMC Rx Interrupt Register 31–264

MMC TX Interrupt Mask Register 31–268

MMC Tx Interrupt Register 31–272

Rx 1024- to Max-Byte Frames (Good/Bad) Register 31–276

Rx 128- to 255-Byte Frames (Good/Bad) Register 31–277

Rx 256- to 511-Byte Frames (Good/Bad) Register 31–278

Rx 512- to 1023-Byte Frames (Good/Bad) Register 31–279

Rx 64-Byte Frames (Good/Bad) Register 31–280

Rx 65- to 127-Byte Frames (Good/Bad) Register 31–281

Rx alignment Error Register 31–282

Rx Broadcast Frames (Good) Register 31–283

Rx CRC Error Register 31–284

Rx Good Control Frames Register 31–285

Rx FIFO Overflow Register 31–286

Rx Frame Count (Good/Bad) Register 31–287

Rx ICMP Error Frames Register 31–288

Rx ICMP Error Octets Register 31–289

Rx ICMP Good Frames Register 31–290

Rx ICMP Good Octets Register 31–291

Rx IPv4 Datagrams Fragmented Frames Register 31–292
Rx IPv4 Datagrams Fragmented Octets Register 31–293
Rx IPv4 Datagrams (Good) Register 31–294
Rx IPv4 Datagrams Good Octets Register 31–295
Rx IPv4 Datagrams Header Errors Register 31–296
Rx IPv4 Datagrams Header Errors Register 31–297
Rx IPv4 Datagrams No Payload Frame Register 31–298
Rx IPv4 Datagrams No Payload Octets Register 31–299
Rx IPv4 UDP Disabled Frames Register 31–300
Rx IPv4 UDP Disabled Octets Register 31–301
Rx IPv6 Datagrams Good Frames Register 31–302
Rx IPv6 Good Octets Register 31–303
Rx IPv6 Datagrams Header Error Frames Register 31–304
Rx IPv6 Header Errors Register 31–305
Rx IPv6 Datagrams No Payload Frames Register 31–306
Rx IPv6 No Payload Octets Register 31–307
Rx Jab Error Register 31–308
Rx Length Error Register 31–309
Rx Multicast Frames (Good) Register 31–310
Rx Octet Count (Good) Register 31–311
Rx Octet Count (Good/Bad) Register 31–312
Rx Out Of Range Type Register 31–313
Rx Oversize (Good) Register 31–314
Rx Pause Frames Register 31–315
Rx Error Frames Received Register 31–316
Rx Runt Error Register 31–317
Rx TCP Error Frames Register 31–318
Rx TCP Error Octets Register 31–319
Rx TCP Good Frames Register 31–320
Rx TCP Good Octets Register 31–321

Rx Unicast Frames (Good) Register 31–322
Rx UDP Error Frames Register 31–323
Rx UDP Error Octets Register 31–324
Rx UDP Good Frames Register 31–325
Rx UDP Good Octets Register 31–326
Rx Undersize (Good) Register 31–327
Rx VLAN Frames (Good/Bad) Register 31–328
Rx Watch Dog Error Register 31–329
SMI Address Register 31–330
SMI Data Register 31–332
Time Stamp Addend Register 31–333
Time Stamp Auxiliary TS Nano Seconds Register 31–334
Time Stamp Auxiliary TM Seconds Register 31–335
Time Stamp Control Register 31–336
Time Stamp High Second Register 31–342
Time Stamp Nanoseconds Register 31–343
Time Stamp Nanoseconds Update Register 31–344
Time Stamp PPS Interval Register 31–345
Time Stamp Target Time Nanoseconds Register 31–346
Time Stamp Target Time Seconds Register 31–347
PPS Width Register 31–348
PPS 1 Interval Register 31–349
PPS 1 Target Time Nanoseconds Register 31–350
PPS 1 Target Time Seconds Register 31–351
PPS 1 Width Register 31–352
PPS 2 Interval Register 31–353
PPS 2 Target Time Nanoseconds Register 31–354
PPS 2 Target Time Seconds Register 31–355
PPS 2 Width Register 31–356
PPS 3 Interval Register 31–357

PPS 3 Target Time Nanoseconds Register 31–358
PPS 3 Target Time Seconds Register 31–359
PPS 3 Width Register 31–360
PPS Control Register 31–361
Time Stamp Low Seconds Register 31–364
Time Stamp Seconds Update Register 31–365
Time Stamp Status Register 31–366
Time Stamp Sub Second Increment Register 31–369
Tx 1024- to Max-Byte Frames (Good/Bad) Register 31–370
Tx 128- to 255-Byte Frames (Good/Bad) Register 31–371
Tx 256- to 511-Byte Frames (Good/Bad) Register 31–372
Tx 512- to 1023-Byte Frames (Good/Bad) Register 31–373
Tx 64-Byte Frames (Good/Bad) Register 31–374
Tx 65- to 127-Byte Frames (Good/Bad) Register 31–375
Tx Broadcast Frames (Good) Register 31–376
Tx Broadcast Frames (Good/Bad) Register 31–377
Tx Carrier Error Register 31–378
Tx Deferred Register 31–379
Tx Excess Collision Register 31–380
Tx Excess Deferral Register 31–381
Tx Frame Count (Good) Register 31–382
Tx Frame Count (Good/Bad) Register 31–383
Tx Late Collision Register 31–384
Tx Multicast Frames (Good) Register 31–385
Tx Multicast Frames (Good/Bad) Register 31–386
Tx Multiple Collision (Good) Register 31–387
Tx Octet Count (Good) Register 31–388
Tx OCT Count (Good/Bad) Register 31–389
Number of Tx Frames (Good) greater than maxsize 31–390
Tx Pause Frame Register 31–391

Tx Single Collision (Good) Register 31–392

Tx Unicast Frames (Good/Bad) Register 31–393

Tx Underflow Error Register 31–394

Tx VLAN Frames (Good) Register 31–395

VLAN Tag Register 31–396

VLAN Hash Table Register 31–398

VLAN Tag Inclusion or Replacement Register 31–399

Watchdog Timeout Register 31–400

Sinus Cardinalis (SINC) Filter

SINC Filter Features 32–1

SINC Functional Description 32–2

ADSP-SC58x SINC Register List 32–2

ADSP-SC58x SINC Interrupt List 32–3

ADSP-SC58x SINC Trigger List 32–3

SINC Definitions 32–4

SINC Block Diagram 32–4

SINC Architectural Concepts 32–6

Digital Filter 32–6

DC Gain and Data Resolution 32–7

Frequency Response 32–7

Output Scaling 32–8

SINC Operating Modes 32–9

SINC Data Transfer Modes 32–10

SINC Signal Modes 32–10

SINC Event Control 32–11

SINC Interrupt Signals 32–11

SINC Status and Error Signals 32–12

SINC Programming Model 32–12

SINC Programming Concepts 32–13

Channel Configuration 32–13

Trigger Masking 32–14

Interrupt Masking 32–14

Modulator Clock 32–14

Filter Configuration 32–14

Primary Filter Parameters 32–15

Primary DMA Configuration and Data Interrupts 32–15

Secondary Filter Parameters 32–16

Overload Detection 32–16

ADSP-SC58x SINC Register Descriptions 32–17

Bias for Group 0 Register 32–18

Bias for Group 1 Register 32–19

Clock Control Register 32–20

Control Register 32–22

History Status Register 32–26

Level Control for Group 0 Register 32–28

Level Control for Group 1 Register 32–31

(Amplitude) Limits for Secondary Filter 0 Register 32–34

(Amplitude) Limits for Secondary Filter 1 Register 32–35

(Amplitude) Limits for Secondary Filter 2 Register 32–36

(Amplitude) Limits for Secondary Filter 3 Register 32–37

Pair 0 Secondary (Filter) History n Register 32–38

Pair 1 Secondary (Filter) History n Register 32–39

Pair 2 Secondary (Filter) History n Register 32–40

Pair 3 Secondary (Filter) History n Register 32–41

Primary (Filters) Head for Group 0 Register 32–42

Primary (Filters) Head for Group 1 Register 32–43

Primary (Filters) Pointer for Group 0 Register 32–44

Primary (Filters) Pointer for Group 1 Register 32–45

Primary (Filters) Tail for Group 0 Register 32–46

Primary (Filters) Tail for Group 1 Register 32–47

Rate Control for Group 0 Register 32–48
Rate Control for Group 1 Register 32–50
Status Register 32–52

Digital Audio Interface (DAI)

SRU Features 33–1
Functional Description 33–2
ADSP-SC58x DAI Register List 33–2
ADSP-SC58x DAI Interrupt List 33–4
DAI Block Diagram 33–4
DAI Signal Naming Conventions 33–5
I/O Pin Buffers 33–5
Pin Buffer Signals 33–6
Pin Buffer Input Signal 33–6
Pin Buffer Enable Signal 33–6
Pin Buffer Input Level State 33–6
Pin Buffer Functions 33–7
Pin Buffers as Signal Input 33–7
Pin Buffers As Signal Output 33–7
DAI Pin Buffer Status 33–8
DAIn Peripherals 33–8
Output Signals With Pin Buffer Enable Control 33–8
Output Signals Without Pin Buffer Enable Control 33–8
Signal Routing Units (SRUs) 33–8
Cross Mode Connections 33–9
Signal Routing Matrix by Groups 33–10
DAI Group Routing 33–11
Rules for SRU Connections 33–12
Miscellaneous Buffers and Functions 33–13
DAI Routing Capabilities 33–13
DAI Default Routing 33–15

Unused DAI Connections 33–18
DAI Operating Modes 33–18
  DAI Pin Buffer Polarity 33–18
  DAI Miscellaneous Buffer Polarity 33–18
DAI System Interrupt Controller (SIC) 33–19
Signal Routing Unit Effect Latency 33–21
DAI Programming Model 33–21
Debug Features 33–21
DAI Sources Overview 33–22
  DAI0 Group A – Clock Routing Source Signals 33–22
  DAI0 Group B – Serial Data Source Signals 33–23
  DAI0 Group C – Frame Sync Source Signals 33–25
  DAI0 Group D – Pin Signal Assignment Source Signals 33–26
  DAI0 Group E – Miscellaneous Source Signals 33–29
  DAI0 Group F – Pin Output Enable Source Signals 33–31
  DAI1 Group A – Clock Routing Source Signals 33–32
  DAI1 Group B – Serial Data Source Signals 33–34
  DAI1 Group C – Frame Sync Source Signals 33–35
  DAI1 Group D – Pin Signal Assignment Source Signals 33–37
  DAI1 Group E – Miscellaneous Source Signals 33–40
  DAI1 Group F – Pin Output Enable Source Signals 33–41
DAI Destination Registers Overview 33–43
ADSP-SC58x DAI Register Descriptions 33–49
  Clock Routing Control Register 0 33–52
  Clock Routing Control Register 1 33–54
  Clock Routing Control Register 2 33–56
  Clock Routing Control Register 3 33–57
  Clock Routing Control Register 4 33–58
  Clock Routing Control Register 5 33–59
  Serial Data Routing Control Register 0 33–60

Serial Data Routing Control Register 1 33–61
Serial Data Routing Control Register 2 33–62
Serial Data Routing Control Register 3 33–63
Serial Data Routing Control Register 4 33–64
Serial Data Routing Control Register 5 33–65
Serial Data Routing Control Register 6 33–66
Frame Sync Routing Control Register 0 33–67
Frame Sync Routing Control Register 1 33–69
Frame Sync Routing Control Register 2 33–71
Frame Sync Routing Control Register 4 33–72
Falling-Edge Interrupt Mask Register 33–73
Core Interrupt Priority Assignment Register 33–76
Rising-Edge Interrupt Mask Register 33–79
High Priority Interrupt Latch Register 33–82
Shadow High Priority Interrupt Latch Register 33–85
Low Priority Interrupt Latch Register 33–89
Shadow Low Priority Interrupt Latch Register 33–92
Miscellaneous Control Register 0 33–96
Miscellaneous Control Register 1 33–98
Pin Buffer Enable Register 0 33–100
Pin Buffer Enable Register 1 33–101
Pin Buffer Enable Register 2 33–102
Pin Buffer Enable Register 3 33–103
Pin Buffer Assignment Register 0 33–104
Pin Buffer Assignment Register 1 33–105
Pin Buffer Assignment Register 2 33–106
Pin Buffer Assignment Register 3 33–107
Pin Buffer Assignment Register 4 33–108
Pin Status Register 33–110

Serial Port (SPORT)

Features 34–1

Signal Descriptions 34–2

SRU Programming 34–6

Functional Description 34–6

- ADSP-SC58x SPORT Register List 34–6
- ADSP-SC58x SPORT Interrupt List 34–7
- ADSP-SC58x SPORT Trigger List 34–9
- ADSP-SC58x SPORT DMA Channel List 34–10
- Block Diagram 34–11
- Architectural Concepts 34–12
  - Multiplexer Logic 34–13
- Data Types and Companding 34–16
  - Companding as a Function 34–17
- Transmit Path 34–17
- Receive Path 34–19

Operating Modes and Options 34–19

- Serial Word Length 34–21
- Clock Sample and Drive Edges 34–21
- Frame Sync Options 34–23
  - Data-Dependent versus Data-Independent Frame Syncs 34–23
  - Support for Edge-Detected and Level-Sensitive Frame Syncs 34–23
  - Early versus Late Frame Syncs 34–24
  - Framed versus Unframed Frame Syncs 34–25
  - Frame Sync Polarity 34–26
  - Premature Frame Sync Error Detection 34–26
- Mode Selection 34–27
  - Standard DSP Serial Mode 34–27
  - Stereo Modes 34–28
    - I^2S Mode 34–29

Left-Justified Mode 34–30
Right-Justified Mode 34–31
Multichannel (TDM) Mode 34–33
Packed I^{2}S Mode 34–36
Gated Clock Mode 34–37
Data Transfers and Interrupts 34–38
Data Buffers 34–38
Data Buffer Status 34–40
Single-Word (Core) Transfers 34–40
DMA Transfers 34–41
Data Transfer Interrupt 34–42
Error Detection (Status) Interrupt 34–43
SPORT Programming Model 34–44
Initializing Core-Driven (Non-MCM) Transfers 34–45
Initializing Multichannel Transfers 34–46
Using DMA for SPORT Transfers 34–47
Using Companding as a Function 34–47
ADSP-SC58x SPORT Register Descriptions 34–48
Half SPORT 'A' Multichannel 0-31 Select Register 34–50
Half SPORT 'B' Multichannel 0-31 Select Register 34–51
Half SPORT 'A' Multichannel 32-63 Select Register 34–52
Half SPORT 'B' Multichannel 32-63 Select Register 34–53
Half SPORT 'A' Multichannel 64-95 Select Register 34–54
Half SPORT 'B' Multichannel 64-95 Select Register 34–55
Half SPORT 'A' Multichannel 96-127 Select Register 34–56
Half SPORT 'B' Multichannel 96-127 Select Register 34–57
Half SPORT 'A' Control 2 Register 34–58
Half SPORT 'B' Control 2 Register 34–59
Half SPORT 'A' Control Register 34–60
Half SPORT 'B' Control Register 34–68

Half SPORT 'A' Divisor Register 34–77
Half SPORT 'B' Divisor Register 34–78
Half SPORT 'A' Error Register 34–79
Half SPORT 'B' Error Register 34–81
Half SPORT 'A' Multichannel Control Register 34–83
Half SPORT 'B' Multichannel Control Register 34–85
Half SPORT 'A' Multichannel Status Register 34–87
Half SPORT 'B' Multichannel Status Register 34–88
Half SPORT 'A' Rx Buffer (Primary) Register 34–89
Half SPORT 'B' Rx Buffer (Primary) Register 34–90
Half SPORT 'A' Rx Buffer (Secondary) Register 34–91
Half SPORT 'B' Rx Buffer (Secondary) Register 34–92
Half SPORT 'A' Tx Buffer (Primary) Register 34–93
Half SPORT 'B' Tx Buffer (Primary) Register 34–94
Half SPORT 'A' Tx Buffer (Secondary) Register 34–95
Half SPORT 'B' Tx Buffer (Secondary) Register 34–96

Precision Clock Generators (PCG)

Features 35–1
Functional Description 35–1
  ADSP-SC58x PCG Register List 35–2
  Internal Interface 35–2
  Serial Clock 35–3
  Frame Sync 35–3
    Frame Sync Output 35–3
    Divider Mode Selection 35–3
    Phase Shift 35–4
    Pulse Width 35–5
    Default Pulse Width 35–5
    Input Clock Source Considerations 35–5
    Timing Example for I^2S Mode 35–6

Cross Mode Connections 35–6

Operating Modes 35–6

Normal Mode 35–6

Bypass Mode 35–7

One-Shot Mode 35–7

PCG Event Control 35–8

External Event Trigger 35–8

External Event Trigger Delay 35–8

Audio System Example 35–9

Clock Configuration Examples 35–10

Programming Model 35–10

Frame Sync Phase Setting 35–11

External Event Trigger 35–11

Debug Features 35–11

ADSP-SC58x PCG Register Descriptions 35–11

Precision Clock A Control 0 Register 35–13

Precision Clock A Control 1 Register 35–14

Precision Clock B Control 0 Register 35–15

Precision Clock B Control 1 Register 35–16

Precision Clock C Control 0 Register 35–17

Precision Clock C Control 1 Register 35–18

Precision Clock D Control 0 Register 35–19

Precision Clock D Control 1 Register 35–20

Precision Clock Pulse Width Control 1 Register 35–21

Precision Clock Pulse Width Control 2 Register 35–23

Precision Clock Frame Sync Synchronization 1 Register 35–25

Precision Clock Frame Sync Synchronization 2 Register 35–27

Asynchronous Sample Rate Converter (ASRC)

Features 36–1

Functional Description 36–2
  ADSP-SC58x ASRC Register List 36–2
  ASRC Interrupt List 36–2
  ASRC Block Diagram 36–2
  SRU Programming 36–3
  Clocking 36–3
  I/O Ports 36–3
  De-Emphasis Filter 36–4
  Mute Control 36–4
  SRC Core 36–4
    RAM FIFO 36–4
    Digital Servo Loop 36–5
    FIR Filter 36–5
    Sample Rate Sensing 36–5
    Digital Filter Group Delay 36–5
  Data Format 36–6
Operating Modes 36–6
  TDM Input Mode 36–6
  TDM Output Mode 36–6
  Matched-Phase Mode 36–7
  Bypass Mode 36–8
  De-Emphasis Mode 36–8
  Dithering Mode 36–8
  Muting Modes 36–9
    Soft Mute 36–9
    Hard Mute 36–9
    Auto Mute 36–9
Interrupts 36–9
  Sources 36–10
    SRC Mute Out 36–10

Masking 36–10
Service 36–10
Programming Model 36–10
Debug Features 36–10
ADSP-SC58x ASRC Register Descriptions 36–10
Control Register for ASRC 0 and 1 36–12
Control Register for ASRC 2 and 3 36–17
Mute Register 36–22
Ratio Register for ASRC 0 and 1 36–23
Ratio Register for ASRC 2 and 3 36–25

Sony/Philips Digital Interface (S/PDIF)

Features 37–1
ADSP-SC58x SPDIF Register List 37–1
SRU Programming 37–3
S/PDIF Interrupt List 37–3
Clocking 37–3
S/PDIF Transmitter 37–3
Functional Description 37–4
Input Data Formats 37–5
Operating Modes 37–6
Full Serial Mode 37–6
Standalone Mode 37–6
Data Output Mode 37–7
S/PDIF Receiver 37–7
Functional Description 37–8
Clock Recovery 37–9
Output Data Format 37–9
Channel Status 37–9
Operating Modes 37–9
Compressed or Non-linear Audio Data 37–9

        Emphasized Audio Data 37–10
        Single-Channel Double-Frequency Mode 37–10
    Clock Recovery Modes 37–10
Interrupts 37–11
  Sources 37–11
    Transmit Block Start 37–11
    Receiver Status 37–11
    Receiver Error 37–11
  Masking 37–11
  Service 37–12
Programming Model 37–12
  Programming the Transmitter 37–12
  Programming the Receiver 37–12
  Interrupted Data Streams on the Receiver 37–13
Debug Features 37–13
  Loopback Routing 37–13
ADSP-SC58x SPDIF Register Descriptions 37–13
  Receive Control 37–15
  Receive Status Register 37–17
  Receive Status A0 Register 37–20
  Receive Status B0 Register 37–21
  Receive Status A1 Register 37–22
  Receive Status B1 Register 37–23
  Transmit Control Register 37–24
  Transmit Status A0 Register 37–27
  Transmit Status A1 Register 37–28
  Transmit Status A2 Register 37–29
  Transmit Status A3 Register 37–30
  Transmit Status A4 Register 37–31
  Transmit Status A5 Register 37–32

Transmit Status B0 Register 37–33
Transmit Status B1 Register 37–34
Transmit Status B2 Register 37–35
Transmit Status B3 Register 37–36
Transmit Status B4 Register 37–37
Transmit Status B5 Register 37–38
Transmit User Buffer A0 Register 37–39
Transmit User Buffer A1 Register 37–40
Transmit User Buffer A2 Register 37–41
Transmit User Buffer A3 Register 37–42
Transmit User Buffer A4 Register 37–43
Transmit User Buffer A5 Register 37–44
Transmit User Buffer B0 Register 37–45
Transmit User Buffer B1 Register 37–46
Transmit User Buffer B2 Register 37–47
Transmit User Buffer B3 Register 37–48
Transmit User Buffer B4 Register 37–49
Transmit User Buffer B5 Register 37–50
User Bit Update Register 37–51

Direct Memory Access (DMA)

DMA Channel Features 38–1
DMA Channel Functional Description 38–3
  ADSP-SC58x DMA Register List 38–3
  ADSP-SC58x DMA Channel List 38–4
  DMA Definitions 38–5
  Block Diagram 38–7
  Architectural Concepts 38–8
    DMA Channel SCB Interface 38–9
      SCB Interface Signals 38–9
      SCB Burst Transfers 38–9

- Data Address Alignment 38–10
- Descriptor Set Address Alignment 38–10
- DMA Channel Peripheral DMA Bus 38–11
  - Peripheral Control Commands 38–11
  - Peripheral-Control Command Restrictions 38–13
- Memory DMA and Triggering 38–14
- Medium Band Width DMA Channel MMR Access Bus 38–16
- DMA Channel Operation Flow 38–17
  - Startup Flow 38–17
  - Refresh Flow 38–18
  - Work Unit Transition Flow 38–19
  - Transfer Termination and Shutdown Flow 38–22
- DMA Channel Errors 38–23
  - Status and Debug Errors 38–23
  - DMA Configuration Register Errors 38–24
  - Illegal Register Write During Run 38–25
  - Address Alignment Error 38–25
  - Memory Access Error 38–25
  - Trigger Overrun Error 38–25
  - Bandwidth-Monitor Error 38–25
  - Control Interface Error 38–25

DMA Operating Modes 38–25
- Register-Based Flow Modes 38–26
  - Stop Mode 38–26
  - Autobuffer Mode 38–26
- Descriptor-Based Flow Modes 38–26
  - Descriptor-Array Mode 38–27
  - Descriptor-List Mode 38–28
  - Descriptor-On-Demand Modes 38–29
- Data Transfer Modes 38–29
  - Two-Dimensional DMA 38–30

DMA Channel Event Control 38–31

  - Event Signals 38–31
  - Work Unit State Events 38–32
  - Peripheral Interrupt Request Events 38–32
  - Peripheral Data Request Events 38–33
  - DMA Channel Triggers 38–33
  - Issuing Triggers 38–33
  - Waiting For Triggers 38–34
- DMA Channel Programming Model 38–35
  - Mode Configuration 38–35
    - Register-Based Linear-Buffer Stop Flow Mode 38–35
    - Register-Based Autobuffer Flow Mode 38–36
    - Descriptor-Array Flow Mode 38–37
    - Descriptor-List Flow Mode 38–38
    - Register-Based Memory-to-Memory Transfer in Stop Flow Mode 38–39
  - Programming Concepts 38–40
    - Synchronization of Software and DMA 38–40
      - Interrupt and Trigger Event-Based Synchronization 38–41
      - Register Polling Based Synchronization 38–41
    - Descriptor Queues 38–42
      - Queues Using Event Generation for Every Descriptor Set 38–42
      - Queues Using Minimal Events 38–43
- ADSP-SC58x DMA Register Descriptions 38–44
  - Start Address of Current Buffer Register 38–45
  - Current Address Register 38–46
  - Bandwidth Limit Count Register 38–47
  - Bandwidth Limit Count Current Register 38–48
  - Bandwidth Monitor Count Register 38–49
  - Bandwidth Monitor Count Current Register 38–50
  - Configuration Register 38–51
  - Current Descriptor Pointer Register 38–59
  - Pointer to Next Initial Descriptor Register 38–60

Previous Initial Descriptor Pointer Register 38–61
Status Register 38–62
Inner Loop Count Start Value Register 38–65
Current Count (1D) or Intra-row XCNT (2D) Register 38–66
Inner Loop Address Increment Register 38–67
Outer Loop Count Start Value (2D only) Register 38–68
Current Row Count (2D only) Register 38–69
Outer Loop Address Increment (2D only) Register 38–70

Extended Memory DMA (EMDMA)

EMDMA Features 39–1
EMDMA Functional Description 39–1
ADSP-SC58x EMDMA Register List 39–2
ADSP-SC58x EMDMA Interrupt List 39–2
ADSP-SC58x EMDMA Trigger List 39–3
DMA Addressing 39–3
DMA Burst Transfers 39–3
Transfer Control Block (TCB) Memory Storage 39–4
Chain Assignment 39–4
Starting Chain Loading 39–4
Buffered Chain Loading Register 39–4
TCB Storage 39–5
EMDMA Operating Modes 39–7
Standard DMA 39–8
Circular Buffered DMA 39–8
Chained DMA Mode 39–9
Data Direction On-the-Fly 39–10
Write Back Circular Index Pointer 39–10
Scatter/Gather DMA 39–10
Pre Modified Read/Write Index 39–12

Delay Line DMA 39–13
ADSP-SC58x EMDMA Register Descriptions 39–15
External Base Address Register 39–17
Circular Buffer Length Register 39–18
Chain Pointer Register 39–19
Internal Count Register 39–20
External Count Register 39–21
External Memory DMA Control Register 39–22
Internal Index Register 39–26
External Index Register 39–27
Internal Modifier Register 39–28
External Modifier Register 39–29
Delay Line Tap Count Register 39–30
Tap List Pointer Register 39–31

Cyclic Redundancy Check (CRC)

CRC Features 40–1
CRC Functional Description 40–2
ADSP-SC58x CRC Register List 40–2
ADSP-SC58x CRC Interrupt List 40–3
CRC Definitions 40–3
CRC Block Diagram 40–4
Peripheral DMA Bus 40–5
MMR Access Bus 40–5
Mirror Block 40–5
Data FIFO 40–5
DMA Request Generator 40–5
CRC Engine 40–5
Compare Logic 40–5
CRC Architectural Concepts 40–6

Look-up Table 40–6
Data Mirroring 40–6
FIFO Status and Data Requests 40–7
CRC Operating Modes 40–8
Data Transfer Modes 40–8
Memory Scan Compute-and-Compare Mode 40–9
Memory Scan Data Verify 40–10
Memory Transfer Compute-and-Compare Mode 40–10
Memory Transfer Data Fill Mode 40–10
CRC Event Control 40–11
Interrupt Signals 40–11
CRC Programming Model 40–11
CRC Mode Configuration 40–11
Look-up Table Generation 40–12
Core Driven Memory Scan Compute-and-Compare Mode 40–12
DMA Driven Memory Scan Compute-and-Compare Mode 40–14
Core Driven Memory Scan Data Verify Mode 40–15
DMA Driven Memory Scan Data Verify Mode 40–17
Core Driven Memory Transfer Compute-and-Compare Mode 40–18
DMA Driven Memory Transfer Compute-and-Compare Mode 40–20
DMA Driven Memory Transfer Data Fill Mode 40–21
ADSP-SC58x CRC Register Descriptions 40–23
Data Compare Register 40–24
Control Register 40–25
Data Word Count Register 40–28
Data Count Capture Register 40–29
Data Word Count Reload Register 40–30
Data FIFO Register 40–31
Fill Value Register 40–32
Interrupt Enable Register 40–33
Interrupt Enable Clear Register 40–34

Interrupt Enable Set Register 40–35
Polynomial Register 40–36
CRC Current Result Register 40–37
CRC Final Result Register 40–38
Status Register 40–39

Housekeeping ADC (HADC)

HADC Features 41–1
HADC Functional Description 41–1
ADSP-SC58x HADC Register List 41–2
ADSP-SC58x HADC Interrupt List 41–2
ADSP-SC58x HADC Trigger List 41–2
HADC Definitions 41–3
HADC Block Diagram 41–4
HADC Signal Descriptions 41–5
HADC Architectural Concepts 41–5
Converter Operation 41–5
Auto-Scan 41–6
Channel Sequence Programming 41–7
ADC Transfer Function 41–7
Results 41–7
HADC Operating Modes 41–7
HADC Event Control 41–8
HADC Programming Model 41–8
ADSP-SC58x HADC Register Descriptions 41–8
Channel Mask Register 41–10
Control Register 41–11
Channel Data Registers 41–13
Interrupt Mask Register 41–14
Status Register 41–15

System Security

Security Features 42–1

Security Functional Description 42–2

Security Mode Configuration 42–4

Status and Error Signals 42–4

System Protection Unit (SPU)

SPU Features 43–1

SPU Functional Description 43–1

  ADSP-SC58x SPU Register List 43–1

  ADSP-SC58x SPU Interrupt List 43–2

  Peripheral Register Write Protection 43–2

SPU Block Diagram 43–5

SPU Architectural Concepts 43–5

SPU Event Control 43–5

SPU Programming Model 43–6

SPU Mode Configuration 43–7

  Locking Write-Protect Registers 43–7

  Protecting a Peripheral 43–7

Configuring Security Privileges of a Peripheral 43–8

ADSP-SC58x SPU Register Descriptions 43–8

  Control Register 43–9

  Secure Check Register 43–10

  Secure Control Register 43–11

  Secure Core Registers 43–13

  Secure Peripheral Register 43–14

  Status Register 43–15

  Write Protect Register n 43–16

ADSP-SC5xx Write-Protect, Secure Peripheral and Secure Core Registers 43–16

ADSP-SC5xx Specific Information 43–23

Security Packet Engine (PKTE)

PKTE Features 44–1

PKTE Functional Description 44–1

ADSP-SC58x PKTE Register List 44–1

ADSP-SC58x PKTE Interrupt List 44–3

PKTE Definitions 44–4

Cipher Module 44–5

Hash Module 44–5

Pseudo-Random Number Generator 44–6

Packet Engine Processing Details 44–8

Crypto Padding 44–8

Pad Generation and Insertion 44–9

Pad Types 44–9

Pad Length 44–10

Pad Verification and Consumption 44–12

Crypto and Hash Algorithms 44–14

IV Processing 44–17

ARC4 Processing 44–18

Hash State Loading 44–19

Sequence Number Processing 44–19

Sequence Number Processing in Extended SSL/TLS 44–19

Sequence Number Processing in DTLS 44–20

PKTE Block Diagram 44–23

PKTE Architectural Concepts 44–24

Packet Engine 44–24

Input/Output FIFO Buffers 44–25

Parallel Operations 44–25

DMA Controller 44–25

Interrupt Controller 44–25

Clock Controller 44–25

PKTE Operating Modes 44–26

Autonomous Ring Mode (ARM) 44–26

Target Command Mode (TCM) 44–27

Direct Host Mode (DHM) 44–27

PKTE Event Control 44–28

PKTE Interrupt Signals 44–28

PKTE Programming Model 44–30

PKTE Mode Configuration 44–33

PKTE Programming Concepts 44–33

Packet Engine Descriptor 44–33

Descriptor Processing 44–35

Ownership of the Descriptor 44–36

Description and Use of the SA Record and State Record Structure 44–37

SA Record Structure 44–37

SA State Structure 44–39

ARC4 State Structure 44–40

Configuring Operations in the PKTE 44–41

Basic Operations and Decoding 44–41

Error Code Description 44–42

Extended Error Codes 44–42

Number Format 44–45

PKTE Programming Examples 44–45

Calculating SHA in Direct Host Mode 44–45

Performing AES Decryption in Direct Host Mode 44–47

ADSP-SC58x PKTE Register Descriptions 44–48

Packet Engine ARC4 State Record Address 44–50

Starting Entry of 256-byte ARC4 State Buffer 44–51

Packet Engine Buffer Pointer Register 44–52

Packet Engine Buffer Threshold Register 44–53

Packet Engine Command Descriptor Ring Base Address 44–55

Packet Engine Command Descriptor Count Register 44–56

Packet Engine Command Descriptor Count Increment Register 44–57

Packet Engine Configuration Register 44–58
PE Clock Control Register 44–61
PKTE Continue Register 44–63
Packet Engine Control Register 44–64
Starting Entry of 256-byte Data Input/Output Buffer 44–69
Packet Engine Destination Address 44–70
Packet Engine DMA Configuration Register 44–71
Packet Engine Endian Configuration Register 44–73
Packet Engine Halt Control Register 44–75
Packet Engine Halt Status Register 44–77
Interrupt Mask Disable Register 44–80
Interrupt Mask Enable Register 44–82
Interrupt Masked Status Register 44–84
Packet Engine Input Buffer Count Register 44–86
Packet Engine Input Buffer Count Increment Register 44–87
Interrupt Configuration Register 44–88
Interrupt Clear Register 44–89
Interrupt Enable Register 44–91
Interrupt Unmasked Status Register 44–93
Packet Engine Length Register 44–95
Packet Engine Output Buffer Count Register 44–97
Packet Engine Output Buffer Count Decrement Register 44–98
Packet Engine Result Descriptor Ring Base Address 44–99
Packet Engine Result Descriptor Count Registers 44–100
Packet Engine Result Descriptor Count Decrement Registers 44–101
Packet Engine Ring Configuration 44–102
Packet Engine Ring Pointer Status 44–103
Packet Engine Ring Status 44–104
Packet Engine Ring Threshold Registers 44–105
Packet Engine SA Address 44–107

ARC4 i and j Pointer Register 44–108

SA Command 0 44–109

SA Command 1 44–114

SA Inner Hash Digest Registers 44–118

SA Key Registers 44–119

SA Initialization Vector Register 44–120

SA Outer Hash Digest Registers 44–121

SA Ready Indicator 44–122

SA Sequence Number Register 44–123

SA Sequence Number Mask Registers 44–124

SA SPI Register 44–125

Packet Engine Source Address 44–126

Packet Engine Status Register 44–127

Packet Engine State Record Address 44–131

State Hash Byte Count Registers 44–132

State Inner Digest Registers 44–133

State Initialization Vector Registers 44–134

Packet Engine User ID 44–135

Public Key Accelerator (PKA)

PKA Features 45–1

PKA Functional Description 45–1

ADSP-SC58x PKA Register List 45–2

PKA Definitions 45–2

PKA Architectural Concepts 45–3

PKA Block Diagram 45–3

PKCP Vector Operations 45–4

Modular Exponentiation Operations 45–6

Modular Inversion 45–13

Modular Inversion with an Even Modulus 45–14

Modular Inversion with a Prime Modulus 45–14
ECC Operations 45–14
ADSP-SC58x PKA Register Descriptions 45–20
PKA Vector_A Length 45–21
PKA Vector_A Address 45–22
PKA Vector_B Length 45–23
PKA Vector_B Address 45–24
PKA Compare Result 45–25
PKA Vector_C Address 45–26
PKA Most-Significant-Word of Divide Remainder 45–27
PKA Vector_D Address 45–28
PKA Function 45–29
Start of PKA RAM space 45–32
PKA Most-Significant-Word of Result Vector 45–33
PKA Bit Shift Value 45–34

Public Key Interrupt Controller (PKIC)

PKIC Functional Description 46–1
ADSP-SC58x PKIC Register List 46–1
ADSP-SC58x PKIC Interrupt List 46–2
PKIC Programming Model 46–2
PKIC Programming Concepts 46–2
ADSP-SC58x PKIC Register Descriptions 46–3
Acknowledge Register 46–4
Enable Clear Register 46–5
Enable Control Register 46–6
Enable Set Register 46–7
Enabled Status Register 46–8
Polarity Control Register 46–9
Raw Status Register 46–10

Type Control Register 46–11

True Random Number Generator (TRNG)

TRNG Features 47–1
TRNG Functional Description 47–1
ADSP-SC58x TRNG Register List 47–1
Random Number Generation 47–2
Locking Detection and Prevention 47–3
Run Testing 47–3
Monobit Testing 47–4
Poker Testing 47–5
Data for Tests 47–5
TRNG Block Diagram 47–6
TRNG Architectural Concepts 47–6
TRNG Operating Modes 47–7
TRNG Data Transfer Modes 47–8
TRNG Event Control 47–8
TRNG Interrupt Signals 47–8
ADSP-SC58x TRNG Register Descriptions 47–9
TRNG Alarm Counter Register 47–11
TRNG Alarm Mask Register 47–13
TRNG Alarm Stop Register 47–14
TRNG Configuration Register 47–15
Counter Register 47–17
TRNG Control Register 47–18
TRNG FRO De-tune Register 47–20
TRNG FRO Enable Register 47–21
TRNG Input Registers 47–22
TRNG Interrupt Acknowledge Register 47–23
TRNG LFSR Access Register 47–25

TRNG LFSR Access Register 47–26
TRNG LFSR Access Register 47–27
TRNG Monobit Test Result Register 47–28
TRNG Output Registers 47–29
TRNG Poker Test Result Registers 47–30
TRNG Run Count Registers 47–31
TRNG Run Test State and Result Registers 47–32
TRNG Status Register 47–34
TRNG Test Register 47–36

Thermal Monitoring Unit (TMU)

TMU Features 48–1
TMU Functional Description 48–1
ADSP-SC58x TMU Register List 48–1
ADSP-SC58x TMU Interrupt List 48–2
ADSP-SC58x TMU Trigger List 48–2
TMU Definitions 48–2
TMU Block Diagram 48–3
TMU Architectural Concepts 48–3
TMU Event Control 48–4
Status and Error Signals 48–4
TMU Programming Guidelines 48–5
ADSP-SC58x TMU Register Descriptions 48–5
Alert High Limit Register 48–6
Alert Low Limit Register 48–7
Averaging Register 48–8
TMU Control Register 48–9
Fault High Limit Register 48–10
Fault Low Limit Register 48–11
Gain Value Register 48–12

Interrupt Mask Register 48–13
Offset Register 48–14
Status Register 48–15
Temperature Value Register 48–16

Harmonic Analysis Engine (HAE)

HAE Features........ 49–1
HAE Functional Description 49–2
ADSP-SC58x HAE Register List 49–2
ADSP-SC58x HAE Interrupt List 49–3
ADSP-SC58x HAE Trigger List........ 49–3
ADSP-SC58x HAE DMA Channel List 49–4
HAE Block Diagram........ 49–4
HAE Architectural Concepts 49–4
Harmonic Engine........ 49–4
Harmonic Analyzer 49–5
Data Transfer Module 49–7
Results Memory 49–9
HAE Results Upper Byte ID 49–9
HAE Result Ranges and Formats 49–10
HAE Operating Modes........ 49–11
HAE Data Transfer Modes 49–11
HAE Event Control 49–11
HAE Interrupt Signals........ 49–11
HAE Status and Error Signals........ 49–11
HAE Programming Model........ 49–12
HAE Programming Concepts 49–13
Theory of Operation 49–13
Initialization........ 49–14
Harmonic Calculations........ 49–15
Configuring Harmonic Calculations Update Rate 49–16

ADSP-SC58x HAE Register Descriptions 49–17
Configuration 0 Register 49–18
Configuration 1 Register 49–19
Configuration 2 Register 49–20
Configuration 3 Register 49–22
Configuration 4 Register 49–23
DIDT Coefficient Register 49–24
DIDT Gain Register 49–25
Harmonic n Index Register 49–26
I (Current) Sample Register 49–27
I (Current) Waveform Register 49–28
Run Register 49–29
Status Register 49–30
Voltage Level Register 49–31
V (Voltage) Sample Register 49–32
V (Voltage) Waveform Register 49–33

FFT Accelerator (FFTA)

FFTA Features........ 50–1
FFTA Functional Description 50–2
ADSP-SC58x FFTA Register List 50–2
FFTA Definitions 50–3
ADSP-SC58x FFTA Interrupt List 50–3
ADSP-SC58x FFTA Trigger List........ 50–4
FFTA Block Diagram........ 50–4
FFTA Programming Model........ 50–5
FFTA Use Cases........ 50–5
Single-Shot FFT 50–5
Pipelined Small FFT........ 50–6
Pipelined Small Interleaved FFT and IFFT Operations 50–6

Fast Convolution 50–8
ADSP-SC58x FFTA Register Descriptions 50–9
Control Register 50–10
Instruction Memory Register 50–11
Loop Counter Value Register 50–12
Program Counter Register 50–13
FFT/IFFT Scale Factor Register 50–14
Status Register 50–15
Thread Count Offset Register 50–18
Wrapper Control Register 50–19
Load/Dump Transfer Left Register 50–20

FIR Accelerator (FIR)

Features 51–1
Clocking 51–1
Functional Description 51–2
ADSP-SC58x FIR Register List 51–3
ADSP-SC58x FIR Interrupt List 51–3
ADSP-SC58x FIR Trigger List 51–4
Compute Block 51–4
Partial Sum Register 51–4
Delay Line Memory 51–5
Coefficient Memory 51–5
Prefetch Data Buffer 51–5
Processing Output 51–5
System Memory Storage 51–6
Coefficients and Input Buffer Storage 51–7
Single Rate Input Filtering 51–7
Decimation 51–7
Interpolation 51–7
Operating Modes 51–8

Single Rate Processing 51–8

Single Iteration 51–8

Floating-point Multi-Iteration 51–8

Window Processing 51–8

Multi-Rate Processing 51–9

Decimation 51–9

Interpolation 51–9

Channel Processing 51–10

Floating-Point Data Format 51–11

Fixed-Point Data Format 51–11

Data Transfer 51–12

Chain Assignment 51–12

DMA Access 51–13

Accelerator TCB 51–13

Chain Pointer DMA 51–13

Programming Model 51–14

Single Channel Processing 51–14

Multichannel Processing 51–14

Dynamic Coefficient Processing Notes 51–15

Debug Mode 51–16

Write to Local Memory 51–16

Read from Local Memory 51–16

Single-Step Mode 51–16

FIR Programming Example 51–16

Computing FIR Output, Tap Length Greater than 4096 51–17

Debug Features 51–18

Local Memory Access 51–19

Single-Step Mode 51–19

Emulation Considerations 51–19

Interrupts 51–19

Sources 51–19

Window Complete 51–19

All Channels Complete 51–20

MAC Status 51–20

Service 51–20

Effect Latency 51–21

Write Effect Latency 51–21

FIR Throughput 51–21

ADSP-SC58x FIR Register Descriptions 51–21

FIR Chain Pointer Register 51–23

FIR Coefficient Count Register 51–24

FIR Coefficient Index Register 51–25

FIR Coefficient Modifier Register 51–26

FIR Global Control Register 51–27

FIR Channel Control Register 51–29

Debug Address Register 51–30

FIR Debug Control Register 51–31

FIR Debug Data Read Register 51–32

FIR Debug Data Write Register 51–33

FIR DMA Status Register 51–34

FIR Input Data Base Register 51–36

FIR Input Data Count Register 51–37

FIR Input Data Index Register 51–38

FIR Input Data Modifier Register 51–39

FIR MAC Status Register 51–40

FIR Output Data Base Register 51–43

FIR Output Data Count Register 51–44

FIR Output Data Index Register 51–45

FIR Output Data Modifier Register 51–46

IIR Accelerator (IIR)

Features 52–1

Clocking 52–1

Functional Description 52–1

- ADSP-SC58x IIR Register List 52–3
- ADSP-SC58x IIR Interrupt List 52–4
- ADSP-SC58x IIR Trigger List 52–4
- Multiply and Accumulate (MAC) Unit 52–4
- Input Data and Biquad State 52–5
- Coefficient Memory 52–5
- Internal Memory Storage 52–5
  - Coefficient Memory Storage 52–5

Operating Modes 52–6

- Window Processing Mode 52–6
- 40-Bit Floating-Point Mode 52–6
- Save Biquad State Mode 52–6

Data Transfers 52–7

- IIR Accelerator TCB 52–7
  - DMA Access 52–8
    - Chain Pointer DMA 52–8
- Effect Latency 52–9
  - Write Effect Latency 52–9
- IIR Throughput 52–9

Interrupts 52–10

- Sources 52–10
  - Window Complete 52–10
  - All Channels Complete 52–10
  - Chained DMA 52–10
  - MAC Status 52–10
- Service 52–11

Programming Model 52–11

Dynamic Coefficient Processing Notes 52–12

Writing to Local Memory 52–13

Reading from Local Memory 52–13

Single Step Mode 52–13

Save Biquad State of the IIR 52–13

Programming Example 52–14

ADSP-SC58x IIR Register Descriptions 52–15

Chain Pointer Register 52–16

Coefficient Buffer Index Register 52–17

Coefficient Buffer Length Register 52–18

Coefficient Index Modifier Register 52–19

Global Control Register 52–20

Channel Control Register 52–22

IIR Debug Address Register 52–23

IIR Debug Control Register 52–24

IIR Debug Read Data High Register 52–25

IIR Debug Read Data Low Register 52–26

IIR Debug Write Data High Register 52–27

IIR Debug Write Data Low Register 52–28

DMA Status Register 52–29

Input Buffer Base Register 52–31

Input Data Index Register 52–32

Input Data Buffer Length Register 52–33

Input Data Index Modifier Register 52–34

MAC Status Register 52–35

Output Buffer Base Register 52–36

Output Data Buffer Index Register 52–37

IIR Output Data Buffer Length Register 52–38

IIR Output Data Index Modifier Register 52–39

Boot ROM and Booting the Processor

SRAM Requirements 53–1

Preboot Operations 53–3

  Start-up Sequence 53–3

    Core Reset Sequencing 53–3

    Core 0 Start-up 53–5

    Core 1 Start-up 53–6

    Core 2 Start-up 53–7

  Idle On Entry 53–8

  Fault Configuration 53–8

  SPU Configuration 53–9

  SMPU Configuration 53–9

  Secure Debug Key Processing 53–10

  CGU Configuration 53–11

  Releasing All Cores From Reset 53–13

  L1/L2 Memory Initialization 53–13

  Default Application Entry Points 53–14

  Memory Faults Configuration 53–14

  NO-BOOT Processing 53–15

  SYS_RESOUTb Processing 53–15

  DMC Configuration 53–16

  Bypassing the Boot Process 53–17

  Boot Mode Disable 53–17

  Boot Command Customization 53–17

  Boot Mode Specific SPU Configuration 53–18

  Executing the Boot Mode 53–18

Boot Modes 53–19

  No-Boot Mode 53–20

  SPI Master Boot Mode 53–20

  SPI Slave Boot Mode 53–27

- Link Port Slave Boot Mode 53–31
- UART Slave Boot Mode 53–32
Boot Loader Stream 53–35
- Block Types 53–39
- Single-Block Boot Streams 53–46
- Direct Code Execution 53–47
- Multi-Application Boot Streams 53–48
- CRC32 Protection 53–49
Secure Boot 53–49
- Terminology 53–51
- Secure Boot Image Signing 53–52
- Secure Boot Image Types 53–52
- Secure Boot Image Format 53–53
- Secure Boot Image Attributes 53–56
- Secure Boot Streams 53–57
- Secure Debug Access 53–59
- Errors and Failures 53–60
Boot ROM Programming Model 53–60
- Boot Mode Driver API 53–60
- Error Handler 53–62
- Page Mode 53–62
- Boot Hook Function 53–63
  - enum ROM_HOOK_CALL_CAUSE 53–63
- Boot Return Feature 53–64
- Boot Termination and Application Execution 53–64
- API Reference 53–65
  - adi_rom_Boot() 53–65
  - adi_rom_BootKernel() 53–67
  - adi_rom_Crc32Init() 53–68
  - adi_rom_Crc32Poly() 53–69

adi_rom_GetAddress() 53–69

adi_rom_MemCompare() 53–70

adi_rom_MemCopy() 53–71

adi_rom_MemCrc() 53–72

adi_rom_MemDma() 53–72

adi_rom_MemFill() 53–75

adi_rom_PeriphDma() 53–76

adi_rom_otp_cfg() 53–77

adi_rom_otp_get() 53–77

adi_rom_otp_lock() 53–78

adi_rom_otp_pgm() 53–78

callback() 53–79

initcode() 53–81

Data Structures 53–81

struct ADI_ROM_BOOT_BUFFER 53–81

struct ADI_ROM_BOOT_CONFIG 53–82

struct ADI_ROM_BOOT_CUSTOM 53–91

struct ADI_ROM_BOOT_HEADER 53–92

struct ADI_ROM_BOOT_INTER_BUFFER 53–93

struct ADI_ROM_BOOT_INTER_BUFFERS 53–93

struct ADI_ROM_BOOT_LINKPORT 53–94

struct ADI_ROM_BOOT_MODES 53–95

struct ADI_ROM_BOOT_REGISTRY 53–95

struct ADI_ROM_BOOT_SPI 53–96

struct ADI_ROM_BOOT_UART 53–99

struct ADI_ROM_OTP_BOOT_CFG 53–100

struct ADI_ROM_OTP_BOOT_CGU_INFO 53–102

struct ADI_ROM_OTP_BOOT_CMD_INFO 53–106

struct ADI_ROM_OTP_BOOT_INFO 53–107

struct ADI_ROM_OTP_DMC_CONFIG 53–108

struct ROM_BOOT_DMA_INSTANCE 53–111

struct ROM_BOOT_MDMA 53–111

struct ROM_BOOT_MDMA_REGS 53–112

struct ROM_DMA_MDMA_CONFIG 53–113

struct ROM_DMA_PDMA_CONFIG 53–114

struct otp_data 53–115

Enumerations 53–118

enum ADI_ROM_BOOT_KEY_TYPE 53–118

enum ADI_ROM_BOOT_TYPE 53–118

enum OTPCMD 53–119

enum ROM_BOOT_MDMA_CRC_SUPPORT 53–121

enum ROM_BOOT_RESULT 53–121

enum ROM_CORE_ID 53–128

enum ROM_DMA_DONE_DETECT_METHOD 53–129

enum ROM_DMA_MDMA_ID 53–129

enum ROM_DMA_MDMA_OPERATION 53–130

enum ROM_DMA_PDMA_OPERATION 53–131

enum ROM_GETADDR_VALUE 53–131

enum ROM_HOOK_CALL_CAUSE 53–132

enum ROM_SB_IMAGE_TYPE 53–132

enum ROM_SPI_PROTOCOL 53–134

System Crossbars (SCB)

SCB Features 54–1

SCB Functional Description 54–1

ADSP-SC58x SCB0 Register List 54–2

ADSP-SC58x SCB1 Register List 54–2

ADSP-SC58x SCB3 Register List 54–2

SCB Architectural Concepts 54–2

SCB Block Diagrams 54–3

SCB Block Diagram 54–4

System Crossbars 54–9

Clock Domain Synchronization 54–10

SCB Bus Master IDs 54–11

SCB Programming Model........54–13
SCB Programming Concepts........54–15
ADSP-SC58x SCB0 Register Descriptions........54–20
Read Quality of Service for Master n........54–21
Write Quality of Service for Master n........54–22
ADSP-SC58x SCB1 Register Descriptions........54–22
Mst00 Interface Block Sync Mode........54–23
ADSP-SC58x SCB3 Register Descriptions........54–23
DCLK0 Interface Block APB WR Tidemark........54–24
SYNC Mode........54–25

System Watchpoint Unit (SWU)

SWU Features........55–1
SWU Functional Description........55–1
ADSP-SC58x SWU Register List........55–1
ADSP-SC58x SWU Interrupt List........55–2
ADSP-SC58x SWU Trigger List........55–3
SWU Definitions........55–4
SWU Architectural Concepts........55–5
SWU-to-SCB Interface........55–5
SWU Block Diagram........55–5
SCB Interface Block........55–5
MMR Interface Block........55–5
SWU Operating Modes........55–5
Bandwidth Mode........55–5
Watchpoint Mode........55–6
Match Block........55–6
Scaling........55–6
SWU Event Control........55–6
SWU Interrupts........55–7
SWU Status and Errors........55–7

Triggers 55–7
SWU Programming Model 55–7
SWU Mode Configuration 55–8
Configuring the SWU for Bandwidth Mode 55–8
Configuring the SWU for Watchpoint Mode 55–9
ADSP-SC58x SWU Register Descriptions 55–10
Count Register n 55–11
Control Register n 55–12
Current Register n 55–17
Global Control Register 55–18
Global Status Register 55–19
Bandwidth History Register n 55–23
ID Register n 55–24
Lower Address Register n 55–25
Target Register n 55–26
Upper Address Register n 55–27

System Debug and Trace Unit (DBG)

DBG Features 56–1
DBG Functional Description 56–2
ADSP-SC58x CSPFT Register List 56–2
ADSP-SC58x TAPC Register List 56–3
ADSP-SC58x TAPC Register List 56–3
ADSP-SC58x STM Trigger List 56–4
ADSP-SC58x CTI Trigger List 56–5
ADSP-SC58x CTI Interrupt List 56–6
DBG Block Diagram 56–6
DBG Definitions 56–7
Test Access Port Controller (TAPC) 56–8
Embedded Trace Macrocell (ETM) 56–8

Debug Access Ports 56–8

Trace Unit 56–9

Programmable Flow Trace (CSPFT) 56–9

System Trace Module (STM) 56–10

Embedded Cross Trigger (ECT) 56–10

CTI Debug Trigger Tables 56–12

ADSP-SC58x CSPFT Register Descriptions 56–14

Address Comparator Access Type Register 56–16

Address Comparator Value Register 56–17

Authentication Status Register 56–18

Configuration Code Extension Register 56–19

Component ID0 Register 56–20

Component ID1 Register 56–21

Component ID2 Register 56–22

Component ID3 Register 56–23

Context ID Comparator Mask Register 56–24

Context ID Comparator Value 56–25

Claim Tag Clear Register 56–26

Claim Tag Set Register 56–27

Counter Enable Event Register 56–28

Counter Reload Event Register 56–31

Counter Reload Value Register 56–34

Counter Value Register 56–35

Main Control Register 56–36

Device Type Identifier Register 56–38

External Output Event Register 56–39

Hardware Feature Register 56–42

Lock Access Register 56–44

Lock Status Register 56–45

Peripheral ID0 Register 56–46

Peripheral ID1 Register 56–47
Peripheral ID2 Register 56–48
Peripheral ID3 Register 56–49
Peripheral ID4 Register 56–50
Status Register 56–51
Synchronization Frequency Register 56–52
TraceEnable Control Register 56–53
TraceEnable Event Register 56–54
CoreSight Trace ID Register 56–57
Trigger Event Register 56–58
TraceEnable Start/Stop Control Register 56–61
ADSP-SC58x TAPC Register Descriptions 56–62
Debug Control Register 56–64
IDCODE Register 56–66
Secure Debug Key 0 Register 56–67
Secure Debug Key 1 Register 56–68
Secure Debug Key 2 Register 56–69
Secure Debug Key 3 Register 56–70
Secure Debug Key Control Register 56–71
Secure Debug Key Status Register 56–72
USERCODE Register 56–73

ADSP-SC58x Register List

Preface

Thank you for purchasing and developing systems using an ADSP-SC58x SHARC+ processor from Analog Devices, Inc.

The ADSP-SC58x processor features two high-performance, floating-point SHARC+® cores (ADSP-2158x) and one ARM® Cortex-A5 core. The cores deliver 24 GFLOPS performance at under 2W for advanced, real-rime audio and industrial applications. The processor is designed for a wide variety of automotive, consumer, professional audio energy distribution systems, and other computationally intensive real-time applications.

Purpose of This Manual

The *ADSP-SC58x SHARC Processor Hardware Reference* provides architectural information about the ADSP-SC58x processors. This hardware reference provides the main architectural information about these processors. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes that they support. For information about programming the ARM core in the ADSP-SC58x processor, visit the ARM Information Center at:

http://infocenter.arm.com.

For timing, electrical, and package specifications, see the processor data sheet.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. The manual assumes the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts, such as programming reference books and data sheets, that describe their target architecture.

What's New in This Manual

This is the first released to production revision (1.0) of the *ADSP-SC58x SHARC+ Processor Hardware Reference*.

Technical or Customer Support

You can reach customer and technical support for processors from Analog Devices in the following ways:

- Post your questions in the processors and DSP support community at ***EngineerZone***:

  http://ez.analog.com/community/dsp

- Submit your questions to technical support at ***Connect with ADI Specialists***:

http://www.analog.com/support

- E-mail your questions about software/hardware development tools to:

  processor.tools.support@analog.com

- E-mail your questions about processors and DSPs to:

  processor.support@analog.com (world wide support)

  processor.china@analog.com (China support)

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

  http://www.analog.com/adi-sales

- Send questions by mail to:

  Analog Devices, Inc.

  Three Technology Way

  P.O. Box 9106

  Norwood, MA 02062-9106 USA

Product Information

Product information can be obtained from the Analog Devices Web site and CrossCore Embedded Studio online Help system.

Analog Devices Web Site

The Analog Devices Web site, http://www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to: http://www.analog.com/processors/technical_library The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Supported Processors

The following is the list of Analog Devices, Inc. processors supported by the CrossCore Embedded Studio® development tools suite.

Blackfin+® (ADSP-BF7xx) Processors

The name *Blackfin+* refers to the enhanced fixed-point Blackfin core architecture featured by the ADSP-BF70x processor product line, which is a family of 16-bit embedded processors.

Blackfin® (ADSP-BF6xx/BF5xx) Processors

The name *Blackfin* refers to the fixed-point core architecture featured on the following processors: ADSP-BF5xx and ADSP-BF6xx.

SHARC® (ADSP-21xxx) Processors

The name *SHARC* refers to the high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-2116x, ADSP-2126x, ADSP-213xx, and ADSP-214xx. These processors can be used in speech, sound, graphics, and imaging applications.

SHARC+® (ADSP-SC5xx, ADSP-215xx) Processors

The name *SHARC+* refers to the enhanced high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-215xx/ADSP-SC5xx. The connected SHARC+ ADSP-SC5xx processors also contain an ARM® Cortex-A5® core. These products can be used in speech, sound, graphics, and imaging applications.

The following is the list of Analog Devices, Inc. processors supported by the IAR Embedded WorkBench® development tools. For information about the IAR Embedded WorkBench product and software download, go to http://www.iar.com/en/Products/IAR-Embedded-Workbench .

Mixed-Signal Control Processors

The ADSP-CM40x processors are based on the ARM Cortex®-M4 core and are designed for motor control and industrial applications.

The ADSP-CM41x processors are based on the ARM Cortex-M4 and ARM Cortex-M0 cores and are designed for motor control and industrial applications.

How to Use this Manual

This section describes how this manual is organized and where you can find specific types of information.

This manual is organized such that it aligns with the Figure 1-1 ADSP-SC58x Functional Block Diagram. This structure is:

ARM Cortex Overview

Power and Clock Management

- Clock Generation Unit (CGU)
- Dynmic Power Management (DPM)
- Reset Control Unit (RCU)

System Interrupts and Triggers

- System Event Control (SEC)
- Trigger Routing Unit (TRU)

System Memory (L2CTL/DMC/SMC/OTPC/SMPU)

- L2 Controller (L2CTL)
- Dynamic Memory Congtroller (DMC)
- Static Memory Controller (SMC)
- One-Time Programmable Memory Controller (OTPC)
- System Memory Protection Unit (SMPU)

Peripherals:

- GPIO
- Timers and Counters
  - General-Purpose Counter (CNT)
  - General-Purpose Timer (TMR)
  - Watchdog Timer (WDOG)

    - Real-Time Counter (RTC)
- Serial Interfaces
    - Serial Ports (SPORTs)
    - Serial Peripheral Interface (SPI, SPIHP)
    - Universal Asynchronous Receiver/Transmitter (UART)
    - Two-Wire Interface (TWI)
    - Control Area Network (CAN)
    - Universal Serial Bus (USB)
- Parallel Interfaces
    - Enhanced Parallel Peripheral Interface (EPPI)
    - Mobile Storage Interface (MSI)

Direct Memory Access

- Dynamic Memory Controller (DMC)
- Memory DMA Controller (MDMA)
- Cyclic Redundancy Check (CRC)

Analog Subsystem (Housekeeping ADC)

Security and Protection (SPU/PKTE/PKIC/PKA/TRNG)

- System Protection Unit (SPU)
- Security Packet Engine (PKTE)
- Public Key Interrupt Controller (PKIC)
- Public Key Accelerator (PKA)
- True Random Number Generator (TRNG)

Booting

System Debug

- System Crossbars (SCB)
- System Watchpoint Unit (SWU)
- System Debug and Trace Unit (DBG)
- Test Access Port Controller (TAPC)
- Trace Unit (CSPFT)

- System Trace Module (STM)

The following list provides information on the organization and contents of each chapter of this manual. Note that not every chapter follows this organization exactly nor does every chapter contain all of the topics listed.

- ***Overview.*** Provides very general information on what the module does. Also may list relevent industry standards that the modules support or comply with.
- ***Features.*** Provides a list of module features such as modes of operation, clocking options, and throughput.
- ***Functional Description.*** Provides details on how the module functions including descriptions of module blocks, connectivity and a glossary. This section also contains a register list and lists of interrupts, triggers and DMA channels (if applicable).
- ***Operating Modes.*** Describes the available modes of operation, how to configure them and how they work.
- ***Event Control.*** Provides information on interrupt and fault generation and how to manage them.
- ***Programming Model.*** Provides general guidelines on how to configure and enable the module. May also provide specific examples for configuring a mode of operation.
- ***Debug Features.*** Describes debugging operations.
- ***Register Descriptions.*** Provides detailed descriptions of all of the registers and bits used by the module.

This manual also contains a comprehensive appendix of memory-mapped register addresses and names. The modules in this appendix are presented in alphabetical order.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
File > ***Close***	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the ***Close*** command appears on the ***File*** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this, …]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `Letter Gothic` font.

Example	Description
`filename`	Non-keyword placeholders appear in text with italic style format.
NOTE:	*NOTE:* For correct operation, ... A note provides supplementary information on a related topic. In the online version of this book, the word *NOTE:* appears instead of this symbol.
CAUTION:	*CAUTION:* Incorrect device operation may result if ... *CAUTION:* Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word *CAUTION:* appears instead of this symbol.
ATTENTION:	*ATTENTION:* Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word *ATTENTION:* appears instead of this symbol.
Registers/Bits	All registers and bits in this manual are linked (clickable) to their respective descriptions in the "Register Descriptions" of each chapter.
Miscellaneous Conventions	Interrupt and internal signals are shown in all caps with no other formatting. For example the SPDIFn_RX or SCLK signal or the PKTE0_IRQ interrupt. An overbar denotes an active-low signal as in $\overline{\text{SYS\_FAULT}}$.

Register Documentation Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top with the short form of the name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name.
- The reset value appears in binary in the individual bits and in hexadecimal to the left of the register.
- Bits marked *X* have an unknown reset value. Consequently, the reset value of registers that contain such bits is undefined or dependent on pin values at reset.
- Shaded bits are reserved

NOTE: To ensure upward compatibility with future implementations, write back the value that is read for reserved bits in a register, unless otherwise specified.

Register description tables use the following conventions:

- Each bit's or bit field's access type appears beneath the bit number in the table in the form (read-access/write-access). The access types include:
  - R = read, RC = read clear, RS = read set, R0 = read zero, R1 = read one, Rx = read undefined
  - W = write, NW = no write, W1C = write one to clear, W1S = write one to set, W0C = write zero to clear, W0S = write zero to set, WS = write to set, WC = write to clear, W1A = write one action

- Many bit and bit field descriptions include enumerations, identifying bit values and related functionality. Unless otherwise indicated (with a prefix), these enumerations are decimal values.

1 Introduction

The ADSP-SC58x and ADSP-215xx processors are members of the SHARC+ family of products. The ADSP-SC58x processor is based on the SHARC+ dual-core processor and the ARM Cortex-A5 processor cores.

As shown in the ***ADSP-SC58x Functional Block Diagram***, by integrating a rich set of industry-leading system peripherals and memory (see product data sheet), the ARM/SHARC+ processor is the platform of choice for next-generation applications that require RISC-like programmability, multimedia support, and leading-edge signal processing in one integrated package. These applications span a wide array of markets, from automotive and pro-audio to industrial-based applications that require high floating-point performance.

NOTE: For specific product configurations (available cores and peripherals), see the SHARC+ Dual Core DSP with ARM Cortex-A5 ADSP-SC582/SC583/SC584/SC587/SC589/ADSP-21583/21584/21587 data sheet.

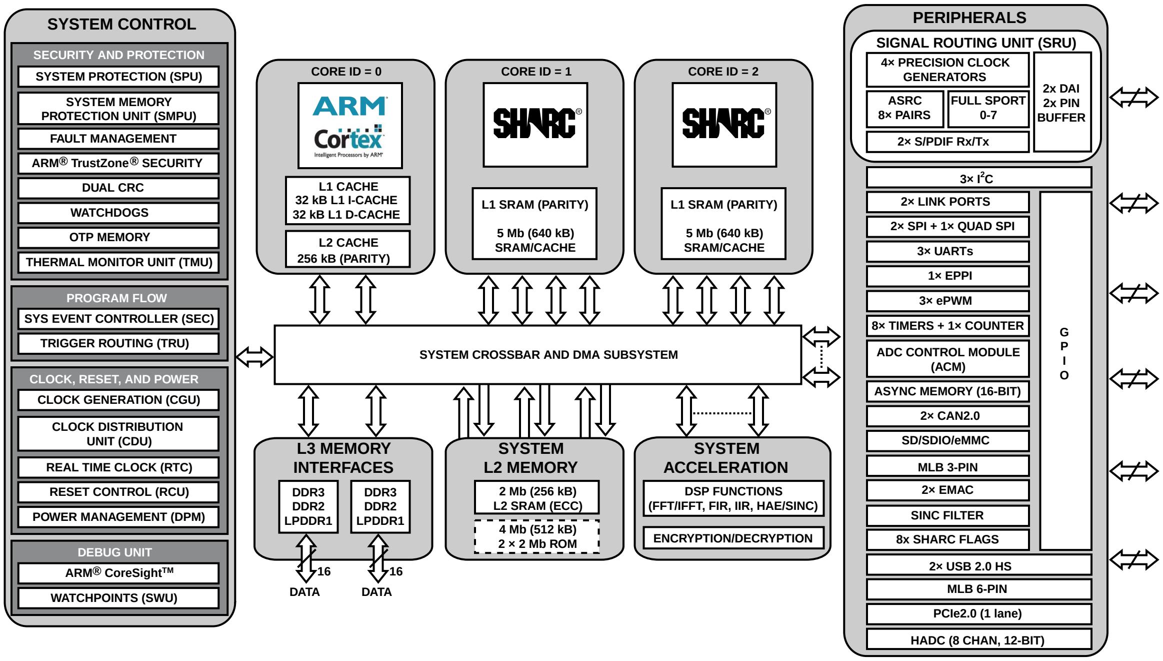

SYSTEM CONTROL
SECURITY AND PROTECTION
SYSTEM PROTECTION (SPU)
SYSTEM MEMORY PROTECTION UNIT (SMPU)
FAULT MANAGEMENT
ARM® TrustZone® SECURITY
DUAL CRC
WATCHDOGS
OTP MEMORY
THERMAL MONITOR UNIT (TMU)
PROGRAM FLOW
SYS EVENT CONTROLLER (SEC)
TRIGGER ROUTING (TRU)
CLOCK, RESET, AND POWER
CLOCK GENERATION (CGU)
CLOCK DISTRIBUTION UNIT (CDU)
REAL TIME CLOCK (RTC)
RESET CONTROL (RCU)
POWER MANAGEMENT (DPM)
DEBUG UNIT
ARM® CoreSight™
WATCHPOINTS (SWU)
CORE ID = 0
ARM Cortex
L1 CACHE 32 kB L1 I-CACHE 32 kB L1 D-CACHE
L2 CACHE 256 kB (PARITY)
CORE ID = 1
SHARC
L1 SRAM (PARITY)
5 Mb (640 kB) SRAM/CACHE
CORE ID = 2
SHARC
L1 SRAM (PARITY)
5 Mb (640 kB) SRAM/CACHE
SYSTEM CROSSBAR AND DMA SUBSYSTEM
L3 MEMORY INTERFACES
DDR3 DDR2 LPDDR1
DDR3 DDR2 LPDDR1
16
DATA
16
DATA
SYSTEM L2 MEMORY
2 Mb (256 kB) L2 SRAM (ECC)
4 Mb (512 kB) 2 × 2 Mb ROM
SYSTEM ACCELERATION
DSP FUNCTIONS (FFT/IFFT, FIR, IIR, HAE/SINC)
ENCRYPTION/DECRYPTION
PERIPHERALS
SIGNAL ROUTING UNIT (SRU)
4× PRECISION CLOCK GENERATORS
ASRC 8× PAIRS
FULL SPORT 0-7
2× S/PDIF Rx/Tx
2x DAI 2x PIN BUFFER
3× I²C
2× LINK PORTS
2× SPI + 1× QUAD SPI
3× UARTs
1× EPPI
3× ePWM
8× TIMERS + 1× COUNTER
ADC CONTROL MODULE (ACM)
ASYNC MEMORY (16-BIT)
2× CAN2.0
SD/SDIO/eMMC
MLB 3-PIN
2× EMAC
SINC FILTER
8x SHARC FLAGS
GPIO
2× USB 2.0 HS
MLB 6-PIN
PCIe2.0 (1 lane)
HADC (8 CHAN, 12-BIT)

Figure 1-1: ADSP-SC58x Functional Block Diagram

ARM Cortex-A5 Processor Core

The ARM Cortex-A5 processor (shown in the Figure 2-1 A5 Sub-System Block Diagram) is a high performance processor with an L1 cache and a virtual Memory Management Unit. This processor is available on the ADSP-SC5xx processors and has a core ID = 0.

SHARC+ Processor Cores

The ADSP-SC58x/ADSP-215xx SHARC+ processors are members of the SIMD SHARC+ family of DSPs that feature Analog Devices Super Harvard Architecture. These 32-bit/40-bit/64-bit floating-point processors are optimized for high performance audio/floating-point applications with their large on-chip SRAM, multiple internal buses to eliminate I/O bottlenecks, and innovative digital audio interfaces (DAI). New enhancements to the SHARC+ core add cache enhancements, branch prediction and other instruction set improvements all while maintaining instruction set compatibility to previous SHARC products.

The ADSP-SC58x/ADSP-2158x processors feature two SHARC+ cores (SHARC1 and SHARC2), which have core IDs of 1 and 2, respectively.

Power and Clock Management (DPM/RCU/CGU/CDU)

The processor contains four modules that control power and clocking.

Dynamic Power Management (DPM)

The Dynamic Power Management (DPM) unit of the processor controls transitions between different power-saving modes.

Reset Control Unit (RCU)

The Reset Control Unit (RCU) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. The RCU supports separate reset control for various chip sub-systems. For deterministic operation programmers should ensure there is no activity between separate chip sub-systems during reset activity. More global reset options are also supported. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. This functionality is important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset).

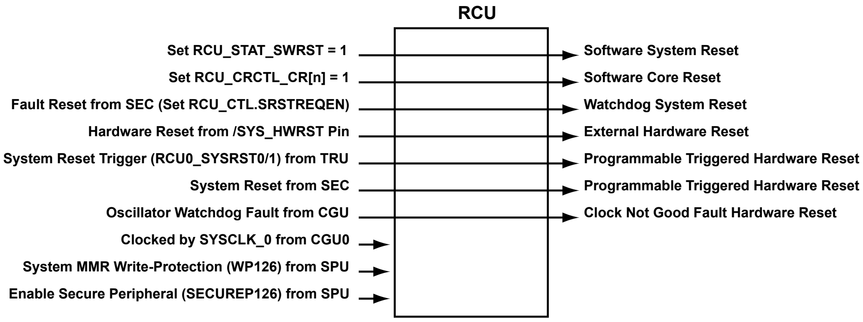

RCU
Set RCU_STAT_SWRST = 1
Set RCU_CRCTL_CR[n] = 1
Fault Reset from SEC (Set RCU_CTL.SRSTREQEN)
Hardware Reset from /SYS_HWRST Pin
System Reset Trigger (RCU0_SYSRST0/1) from TRU
System Reset from SEC
Oscillator Watchdog Fault from CGU
Clocked by SYSCLK_0 from CGU0
System MMR Write-Protection (WP126) from SPU
Enable Secure Peripheral (SECUREP126) from SPU
Software System Reset
Software Core Reset
Watchdog System Reset
External Hardware Reset
Programmable Triggered Hardware Reset
Programmable Triggered Hardware Reset
Clock Not Good Fault Hardware Reset

Figure 1-2: RCU System Diagram

Clock Generation Unit (CGU)

The Clock Generation Unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock that runs at a frequency that is a multiple of the SYS_CLKINx input clock frequency. It also generates all on-chip clocks and synchronization signals. The PCU allows the application software to control the PLL module operation. All clocks derived from the CGU are then forwarded to the CDU for distribution.

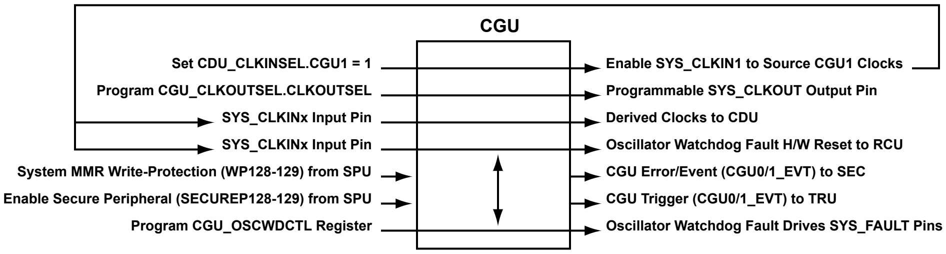

CGU
Set CDU_CLKINSEL.CGU1 = 1
Program CGU_CLKOUTSEL.CLKOUTSEL
SYS_CLKINx Input Pin
SYS_CLKINx Input Pin
System MMR Write-Protection (WP128-129) from SPU
Enable Secure Peripheral (SECUREP128-129) from SPU
Program CGU_OSCWDCTL Register
Enable SYS_CLKIN1 to Source CGU1 Clocks
Programmable SYS_CLKOUT Output Pin
Derived Clocks to CDU
Oscillator Watchdog Fault H/W Reset to RCU
CGU Error/Event (CGU0/1_EVT) to SEC
CGU Trigger (CGU0/1_EVT) to TRU
Oscillator Watchdog Fault Drives SYS_FAULT Pins

Figure 1-3: CGU System Diagram

Clock Distribution Unit (CDU)

The Clock Distribution Unit (CDU) consists of an array of software-configurable multiplexors that select clocks originated from up to four different clock sources that are generated from the CGUs. Unused input clocks are grounded internally and never selected. The output clock signal for each multiplexor is assigned to one or more destinations within the processor.

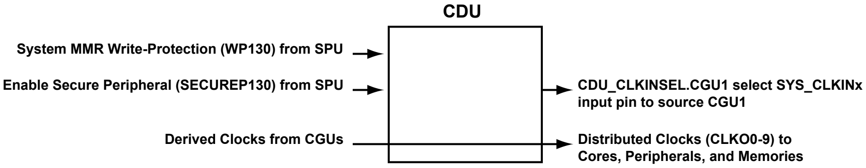

CDU
System MMR Write-Protection (WP130) from SPU
Enable Secure Peripheral (SECUREP130) from SPU
Derived Clocks from CGUs
CDU_CLKINSEL.CGU1 select SYS_CLKINx input pin to source CGU1
Distributed Clocks (CLKO0-9) to Cores, Peripherals, and Memories

Figure 1-4: CDU System Diagram

System Interrupts and Triggers (SEC/TRU)

The SEC and TRU modules provide interrupt and trigger management functions for the processor.

System Event Controller (SEC)

There are two interrupt controllers— the System Event Controller (SEC) and Generic Interrupt Controller (GIC). The generic interrupt controller is used for the ARM core, and the system event controller is used for the SHARC+ cores.

The SEC manages the configuration of all system event sources and the propagation of system events to all connected cores and the system fault interface. The SEC arbitrates among all pending system interrupt requests and presents the highest priority interrupt to the core(s) for processing.

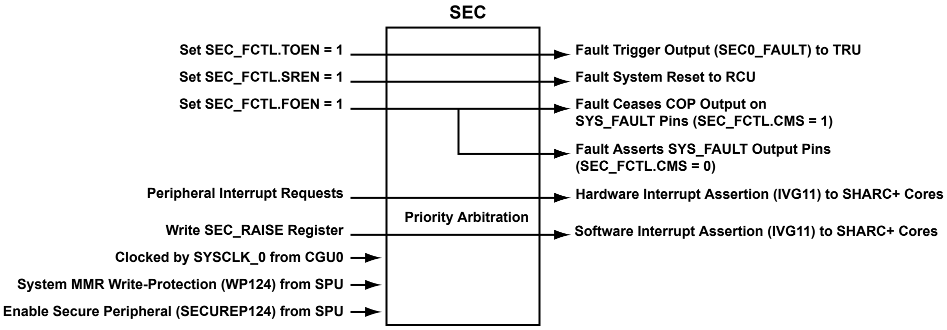

SEC
Set SEC_FCTL.TOEN = 1
Set SEC_FCTL.SREN = 1
Set SEC_FCTL.FOEN = 1
Peripheral Interrupt Requests
Write SEC_RAISE Register
Clocked by SYSCLK_0 from CGU0
System MMR Write-Protection (WP124) from SPU
Enable Secure Peripheral (SECUREP124) from SPU
Priority Arbitration
Fault Trigger Output (SEC0_FAULT) to TRU
Fault System Reset to RCU
Fault Ceases COP Output on SYS_FAULT Pins (SEC_FCTL.CMS = 1)
Fault Asserts SYS_FAULT Output Pins (SEC_FCTL.CMS = 0)
Hardware Interrupt Assertion (IVG11) to SHARC+ Cores
Software Interrupt Assertion (IVG11) to SHARC+ Cores

Figure 1-5: SEC System Diagram

Trigger Routing Unit (TRU)

The Trigger Routing Unit (TRU) provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering
- Synchronization of concurrent activities

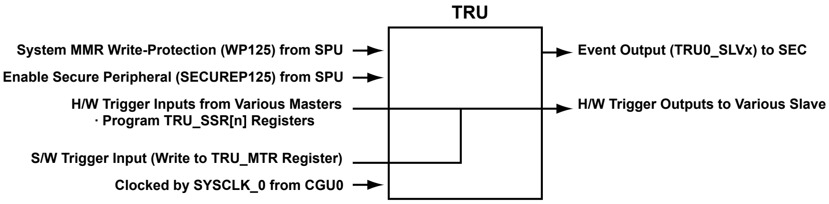

TRU
System MMR Write-Protection (WP125) from SPU
Enable Secure Peripheral (SECUREP125) from SPU
H/W Trigger Inputs from Various Masters
· Program TRU_SSR[n] Registers
S/W Trigger Input (Write to TRU_MTR Register)
Clocked by SYSCLK_0 from CGU0
Event Output (TRU0_SLVx) to SEC
H/W Trigger Outputs to Various Slave

Figure 1-6: TRU System Diagram

System Memory (L2CTL/DMC/SMC/OTPC/SMPU)

The following sections describe the memory architecture of the ADSP-SC58x processors. More information can be found in the processor data sheet.

L2 Memory Controller (L2CTL)

L2 System Memory has significant bandwidth for core accesses, but it is important to note that L2 responds slower to core accesses than L1 memories. L2 SRAM is the ideal storage for multiple processor cores to share data and instruction resources, such as semaphores, shared buffers, and code libraries. Due to sophisticated data integrity protection and write protection, L2 SRAM is also ideal for data and instructions critical for safe operation of the application.

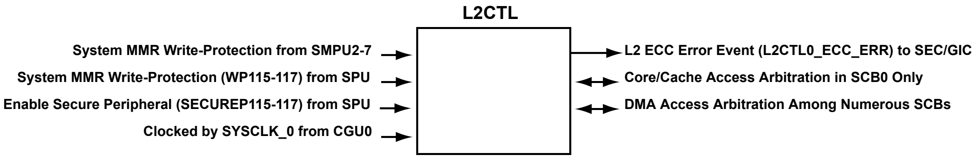

L2CTL
System MMR Write-Protection from SMPU2-7
System MMR Write-Protection (WP115-117) from SPU
Enable Secure Peripheral (SECUREP115-117) from SPU
Clocked by SYSCLK_0 from CGU0
L2 ECC Error Event (L2CTL0_ECC_ERR) to SEC/GIC
Core/Cache Access Arbitration in SCB0 Only
DMA Access Arbitration Among Numerous SCBs

Figure 1-7: L2CTL System Diagram

Dynamic Memory Controller (DMC)

The Dynamic Memory Controller (DMC) provides a glueless interface between DDR3/DDR2/LPDDR SDRAMs and the system crossbar interface (SCB). The DMC enables execution of instructions from, as well as transfer of data to and from, DDR3, DDR2 SDRAM or LPDDR SDRAM, respectively.

The DMC is partitioned in a manner that allows reconfiguration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the protocol controller. An access and operation reordering mechanism is incorporated as an efficiency controller. An SCB slave interface is provided to interface with the on-chip interconnect. This interface results in an efficient slave implementation owing to its out-of-order transaction capabilities.

The DMC supports access to the external memory by core and DMA accesses.

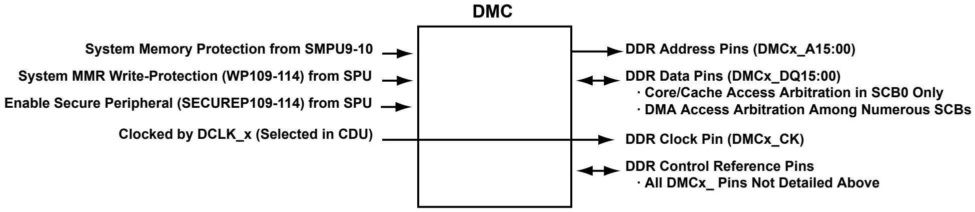

DMC
System Memory Protection from SMPU9-10
System MMR Write-Protection (WP109-114) from SPU
Enable Secure Peripheral (SECUREP109-114) from SPU
Clocked by DCLK_x (Selected in CDU)
DDR Address Pins (DMCx_A15:00)
DDR Data Pins (DMCx_DQ15:00)
· Core/Cache Access Arbitration in SCB0 Only
· DMA Access Arbitration Among Numerous SCBs
DDR Clock Pin (DMCx_CK)
DDR Control Reference Pins
· All DMCx_ Pins Not Detailed Above

Figure 1-8: DMC System Diagram

Static Memory Controller (SMC)

The Static Memory Controller (SMC) is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to various external memories and peripheral devices, including SRAM, ROM, EPROM, NOR flash memory and FPGA/ASIC devices.

The SMC acts as an SCB slave. The processor SCB interconnect fabric arbitrates accesses to the SMC. The SMC connects to signal pins for memory control (such as read, write, output enable, and memory select lines).

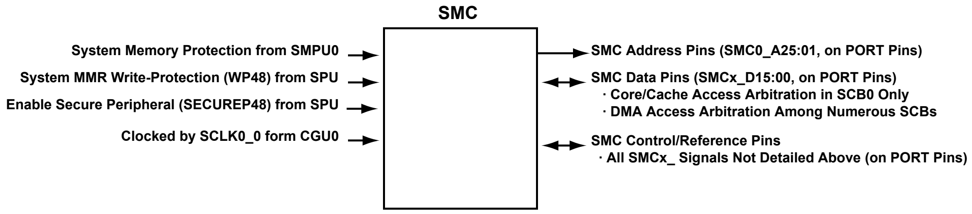

SMC
System Memory Protection from SMPU0
System MMR Write-Protection (WP48) from SPU
Enable Secure Peripheral (SECUREP48) from SPU
Clocked by SCLK0_0 form CGU0
SMC Address Pins (SMC0_A25:01, on PORT Pins)
SMC Data Pins (SMCx_D15:00, on PORT Pins)
· Core/Cache Access Arbitration in SCB0 Only
· DMA Access Arbitration Among Numerous SCBs
SMC Control/Reference Pins
· All SMCx_ Signals Not Detailed Above (on PORT Pins)

Figure 1-9: SMC System Diagram

One-Time Programmable Memory Controller (OTPC)

The One-Time Programmable Memory Controller (OTPC) module is a complete system integrating an OTP memory core with a programming controller, charge pump, and voltage regulator. A built-in Hamming Code Error Correction (ECC), and a fully implemented double-redundant program or read scheme protect the OTP data.

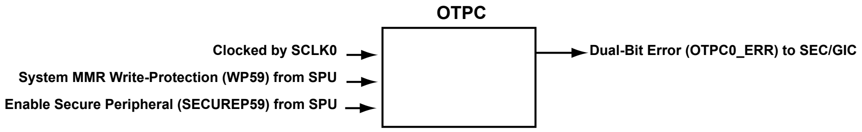

OTPC
Clocked by SCLK0
System MMR Write-Protection (WP59) from SPU
Enable Secure Peripheral (SECUREP59) from SPU
Dual-Bit Error (OTPC0_ERR) to SEC/GIC

Figure 1-10: OTPC System Diagram

System Memory Protection Unit (SMPU)

The System Memory Protection Unit (SMPU) provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

On the ADSP-SC58x, 10 SMPU instances are available to protect the L2, external memory (DMC/SMC), and memory-mapped I/O (PCIe) interfaces. Six instances are allotted to protect the L2 memory, two instances for DMC0 and DMC1, one instance for SMC and one instance for PCIe.

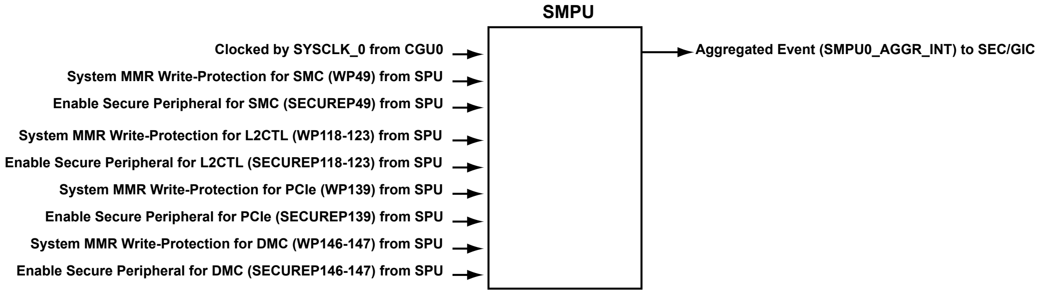

SMPU
Clocked by SYSCLK_0 from CGU0
System MMR Write-Protection for SMC (WP49) from SPU
Enable Secure Peripheral for SMC (SECUREP49) from SPU
System MMR Write-Protection for L2CTL (WP118-123) from SPU
Enable Secure Peripheral for L2CTL (SECUREP118-123) from SPU
System MMR Write-Protection for PCIe (WP139) from SPU
Enable Secure Peripheral for PCIe (SECUREP139) from SPU
System MMR Write-Protection for DMC (WP146-147) from SPU
Enable Secure Peripheral for DMC (SECUREP146-147) from SPU
Aggregated Event (SMPU0_AGGR_INT) to SEC/GIC

Figure 1-11: SMPU System Diagram

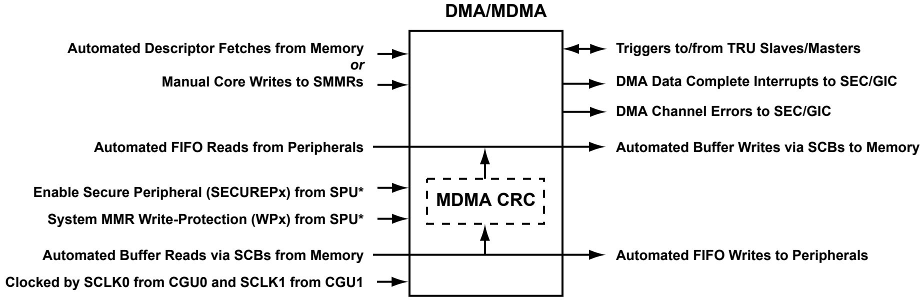
Direct Memory Access (DMA/MDMA/EMDMA/CRC)

DMA Controller (DMA)

The processors use Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processors can specify data transfer operations and return to normal processing while the fully integrated DMA controller carries out the data transfers independent of processor activity.

DMA transfers can occur between memory and a peripheral or between one memory and another memory. Each Memory-to-memory DMA stream uses two channels, where one channel is the source channel, and the second is the destination channel. Most peripherals have at least one dedicated DMA channel associated with them.

All DMAs can transport data to and from all on-chip and off-chip memories. Programs can use two types of DMA transfers, descriptor-based or register-based. Register-based DMA allows the processors to directly program DMA control registers to initiate a DMA transfer. On completion, the control registers may be automatically updated with their original setup values for continuous transfer. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based DMA transfers allow multiple DMA sequences to be chained together and a DMA channel can be programmed to automatically set up and start another DMA transfer after the current sequence completes.


DMA/MDMA
Automated Descriptor Fetches from Memory
or
Manual Core Writes to SMMRs
Automated FIFO Reads from Peripherals
Enable Secure Peripheral (SECUREPx) from SPU*
System MMR Write-Protection (WPx) from SPU*
Automated Buffer Reads via SCBs from Memory
Clocked by SCLK0 from CGU0 and SCLK1 from CGU1
MDMA CRC
Triggers to/from TRU Slaves/Masters
DMA Data Complete Interrupts to SEC/GIC
DMA Channel Errors to SEC/GIC
Automated Buffer Writes via SCBs to Memory
Automated FIFO Writes to Peripherals

** See Table 43-12 for values of x associated with specific DMA channels*

Figure 1-12: DMA System Diagram

Memory DMA Controllers (MDMA)

The processor supports a variety of Memory DMA and Triggering memory-to-memory DMA operations which include:

- Two standard memory DMA channels with CRC protection (32-bit bus width, run on SCLK0)
- One enhanced memory DMA channel (32-bit bus width, runs on SYSCLK)
- Two memory DMA channels (64-bit bus width, run on SYSCLK, one channel may be assigned to the FFT accelerator)
- Two EMDMA channels (32-bit bus width, run on SCLK0)

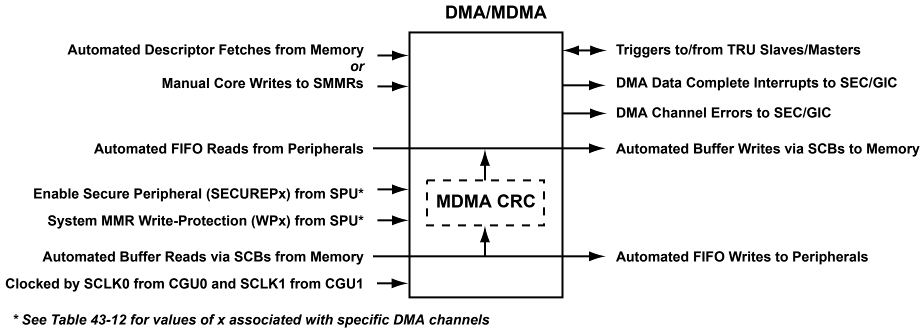

DMA/MDMA
Automated Descriptor Fetches from Memory
or
Manual Core Writes to SMMRs
Automated FIFO Reads from Peripherals
Enable Secure Peripheral (SECUREPx) from SPU*
System MMR Write-Protection (WPx) from SPU*
Automated Buffer Reads via SCBs from Memory
Clocked by SCLK0 from CGU0 and SCLK1 from CGU1
MDMA CRC
Triggers to/from TRU Slaves/Masters
DMA Data Complete Interrupts to SEC/GIC
DMA Channel Errors to SEC/GIC
Automated Buffer Writes via SCBs to Memory
Automated FIFO Writes to Peripherals
* See Table 43-12 for values of x associated with specific DMA channels

Figure 1-13: MDMA System Diagram

Extended Memory DMA (EMDMA)

The Extended Memory DMA (EMDMA) engine can be used in applications that copy data in a non-sequential manner. This includes delay lines, scatter and gather, and circular access types.

Unlike previous SHARC processors which contained external port DMA, the current EPDMA module can access all memory locations (L1/L2/L3) for source and destination DMA operations.

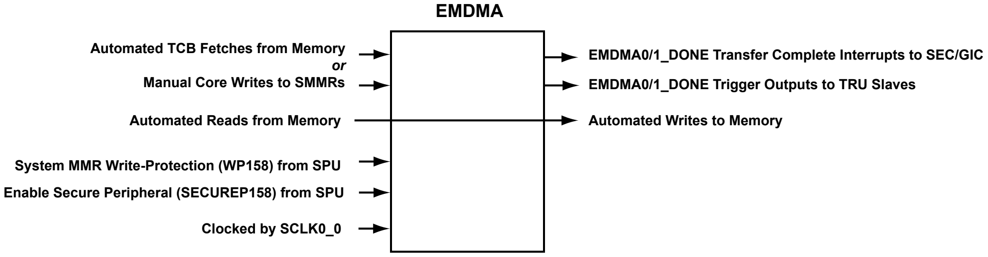

EMDMA
Automated TCB Fetches from Memory
or
Manual Core Writes to SMMRs
Automated Reads from Memory
System MMR Write-Protection (WP158) from SPU
Enable Secure Peripheral (SECUREP158) from SPU
Clocked by SCLK0_0
EMDMA0/1_DONE Transfer Complete Interrupts to SEC/GIC
EMDMA0/1_DONE Trigger Outputs to TRU Slaves
Automated Writes to Memory

Figure 1-14: EMDMA System Diagram

Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check (CRC) peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error. The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by core write transactions.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the look-up table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

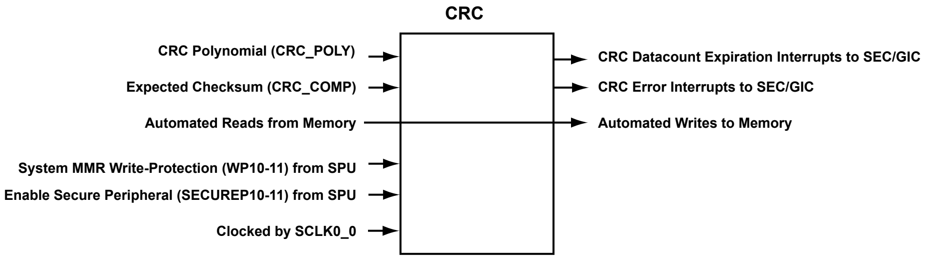

CRC
CRC Polynomial (CRC_POLY)
Expected Checksum (CRC_COMP)
Automated Reads from Memory
System MMR Write-Protection (WP10-11) from SPU
Enable Secure Peripheral (SECUREP10-11) from SPU
Clocked by SCLK0_0
CRC Datacount Expiration Interrupts to SEC/GIC
CRC Error Interrupts to SEC/GIC
Automated Writes to Memory

Figure 1-15: CRC System Diagram

Peripherals

The SHARC+ processor contains a rich set of industry leading system peripherals. The processor is the platform of choice for applications that require RISC-like programmability, multimedia support, and leading edge signal processing in one integrated package. These applications span a wide array of markets, including automotive, pro audio, and industrial-based applications that require high floating-point performance. These peripherals are described in the following sections.

- General-Purpose I/O (GPIO) Peripherals
- DAI/SRU Peripherals
- Dedicated Pin Peripherals

General-Purpose I/O (GPIO) Peripherals

The SHARC+ processors feature up to 102 general-purpose I/O pins mapped across up to seven ports (PORT A through PORT G). Each pin can be configured individually to serve as a GPIO pin or as a peripheral-specific pin.

GPIO Ports (PORT)

When configured in the default GPIO mode, the PORT pins allow for the processor to interface to system components to provide handshaking functionality as either inputs or outputs. When in output mode, open-drain output is supported. A single MMR access can be used to sense or set individual pins or a complete port of 16 pins.

Additionally, each GPIO pin can optionally be configured to raise a system interrupt on the processor via a dedicated pin interrupt (PINT) block, and all peripheral functions are controlled via a set of port multiplexing registers, with specific settings defined in the processor data sheet.

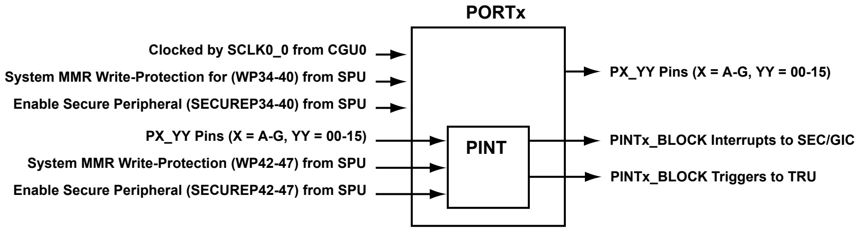

PORTx
Clocked by SCLK0_0 from CGU0
System MMR Write-Protection for (WP34-40) from SPU
Enable Secure Peripheral (SECUREP34-40) from SPU
PX_YY Pins (X = A-G, YY = 00-15)
System MMR Write-Protection (WP42-47) from SPU
Enable Secure Peripheral (SECUREP42-47) from SPU
PINT
PX_YY Pins (X = A-G, YY = 00-15)
PINTx_BLOCK Interrupts to SEC/GIC
PINTx_BLOCK Triggers to TRU

Figure 1-16: PORT System Diagram

Link Ports (LP)

The Link Port (LP) allow the processor to connect to other processors or peripheral link ports using a simple communication protocol for high-speed parallel data transfer. This peripheral allows various I/O peripheral interconnection schemes to I/O peripheral devices, as well as co-processing and multiprocessing schemes.

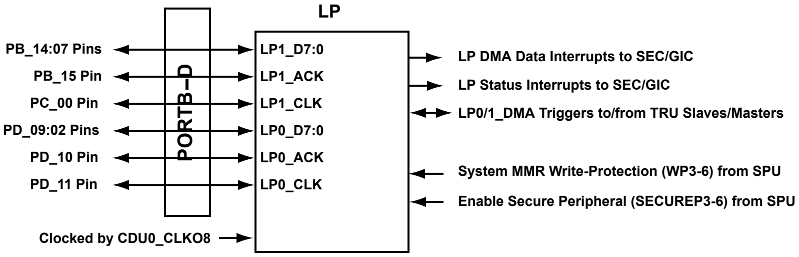

LP
PB_14:07 Pins
PB_15 Pin
PC_00 Pin
PD_09:02 Pins
PD_10 Pin
PD_11 Pin
PORTB–D
LP1_D7:0
LP1_ACK
LP1_CLK
LP0_D7:0
LP0_ACK
LP0_CLK
Clocked by CDU0_CLKO8
LP DMA Data Interrupts to SEC/GIC
LP Status Interrupts to SEC/GIC
LP0/1_DMA Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP3-6) from SPU
Enable Secure Peripheral (SECUREP3-6) from SPU

Figure 1-17: LP System Diagram

Serial Peripheral Interface Ports (SPI)

The Serial Peripheral Interface (SPI) is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

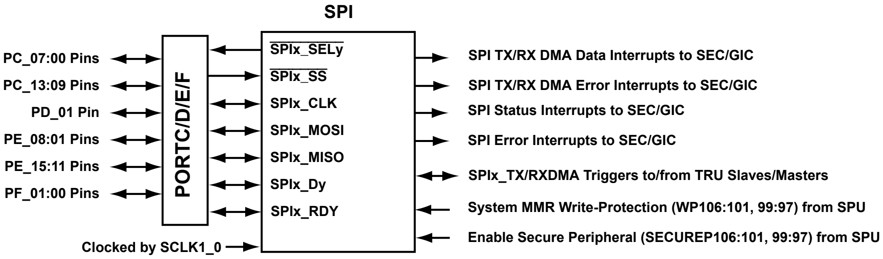

SPI
PC_07:00 Pins
PC_13:09 Pins
PD_01 Pin
PE_08:01 Pins
PE_15:11 Pins
PF_01:00 Pins
PORTC/D/E/F
$\overline{\text{SPIx\_SELy}}$
$\overline{\text{SPIx\_SS}}$
SPIx_CLK
SPIx_MOSI
SPIx_MISO
SPIx_Dy
SPIx_RDY
Clocked by SCLK1_0
SPI TX/RX DMA Data Interrupts to SEC/GIC
SPI TX/RX DMA Error Interrupts to SEC/GIC
SPI Status Interrupts to SEC/GIC
SPI Error Interrupts to SEC/GIC
SPIx_TX/RXDMA Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP106:101, 99:97) from SPU
Enable Secure Peripheral (SECUREP106:101, 99:97) from SPU

Figure 1-18: SPI System Diagram

Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. The UART includes interrupt-handling hardware. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

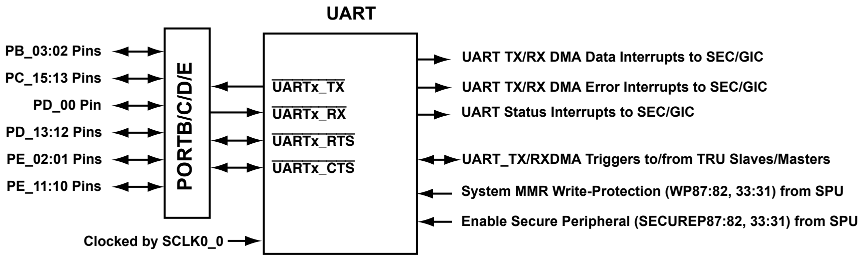

UART
PB_03:02 Pins
PC_15:13 Pins
PD_00 Pin
PD_13:12 Pins
PE_02:01 Pins
PE_11:10 Pins
PORTB/C/D/E
UARTx_TX
UARTx_RX
UARTx_RTS
UARTx_CTS
Clocked by SCLK0_0
UART TX/RX DMA Data Interrupts to SEC/GIC
UART TX/RX DMA Error Interrupts to SEC/GIC
UART Status Interrupts to SEC/GIC
UART_TX/RXDMA Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP87:82, 33:31) from SPU
Enable Secure Peripheral (SECUREP87:82, 33:31) from SPU

Figure 1-19: UART System Diagram

Enhanced Parallel Peripheral Interface (EPPI)

The Enhanced Parallel Peripheral Interface (EPPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins. It can support direct connections to active TFT LCDs, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. Each EPPI has two DMA channels associated with it. Moreover, in some modes, an EPPI can use an extra DMA channel.

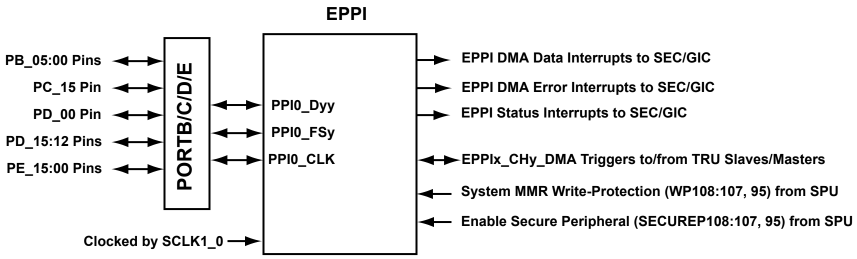

EPPI
PB_05:00 Pins
PC_15 Pin
PD_00 Pin
PD_15:12 Pins
PE_15:00 Pins
PORTB/C/D/E
PPI0_Dyy
PPI0_FSy
PPI0_CLK
Clocked by SCLK1_0
EPPI DMA Data Interrupts to SEC/GIC
EPPI DMA Error Interrupts to SEC/GIC
EPPI Status Interrupts to SEC/GIC
EPPIx_CHy_DMA Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP108:107, 95) from SPU
Enable Secure Peripheral (SECUREP108:107, 95) from SPU

Figure 1-20: EPPI System Diagram

Pulse-Width Modulator (PWM)

The Pulse-Width Modulator (PWM) module is a flexible and programmable waveform generator.

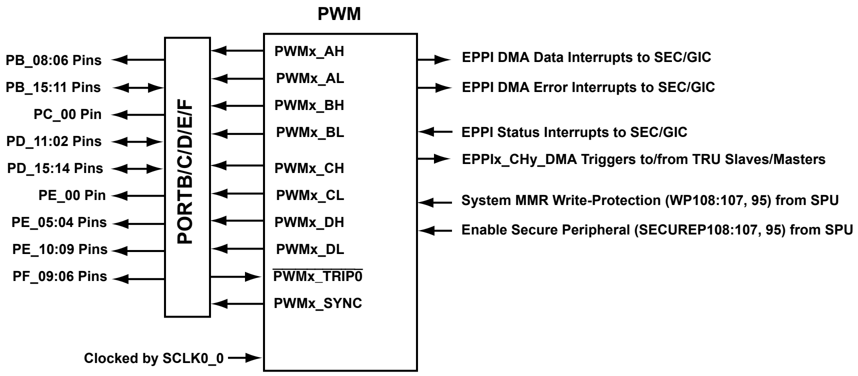

PWM
PB_08:06 Pins
PB_15:11 Pins
PC_00 Pin
PD_11:02 Pins
PD_15:14 Pins
PE_00 Pin
PE_05:04 Pins
PE_10:09 Pins
PF_09:06 Pins
PORTB/C/D/E/F
PWMx_AH
PWMx_AL
PWMx_BH
PWMx_BL
PWMx_CH
PWMx_CL
PWMx_DH
PWMx_DL
PWMx_TRIP0
PWMx_SYNC
Clocked by SCLK0_0
EPPI DMA Data Interrupts to SEC/GIC
EPPI DMA Error Interrupts to SEC/GIC
EPPI Status Interrupts to SEC/GIC
EPPIx_CHy_DMA Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP108:107, 95) from SPU
Enable Secure Peripheral (SECUREP108:107, 95) from SPU

Figure 1-21: PWM System Diagram

General-Purpose Counter (CNT)

The General-Purpose Counter (CNT) converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

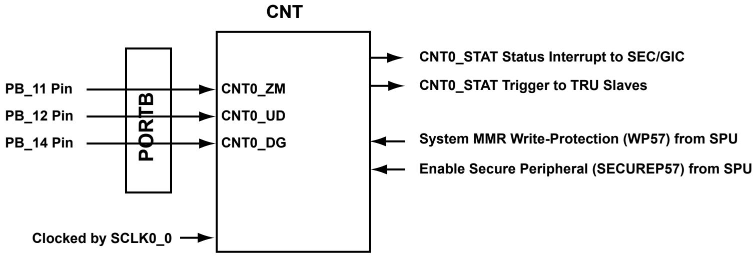

CNT
PB_11 Pin
PB_12 Pin
PB_14 Pin
PORTB
CNT0_ZM
CNT0_UD
CNT0_DG
CNT0_STAT Status Interrupt to SEC/GIC
CNT0_STAT Trigger to TRU Slaves
System MMR Write-Protection (WP57) from SPU
Enable Secure Peripheral (SECUREP57) from SPU
Clocked by SCLK0_0

Figure 1-22: CNT System Diagram

ADC Control Module (ACM)

The processor includes an ADC Control Module (ACM) that provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The processor initiates analog-to-digital conversions, based on either external or internal events.

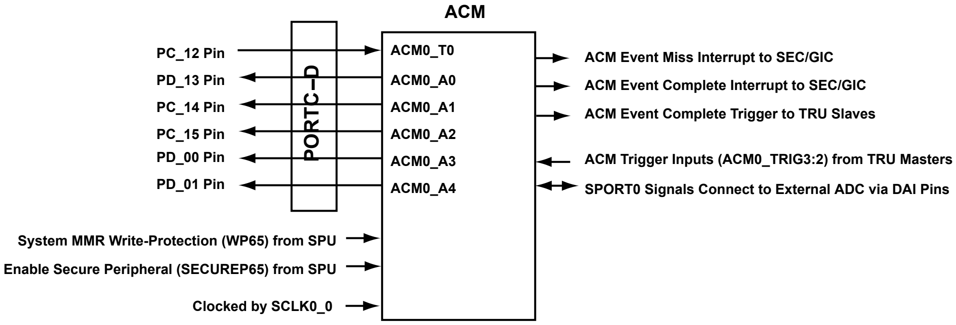

ACM
PC_12 Pin
PD_13 Pin
PC_14 Pin
PC_15 Pin
PD_00 Pin
PD_01 Pin
PORTC–D
ACM0_T0
ACM0_A0
ACM0_A1
ACM0_A2
ACM0_A3
ACM0_A4
ACM Event Miss Interrupt to SEC/GIC
ACM Event Complete Interrupt to SEC/GIC
ACM Event Complete Trigger to TRU Slaves
ACM Trigger Inputs (ACM0_TRIG3:2) from TRU Masters
SPORT0 Signals Connect to External ADC via DAI Pins
System MMR Write-Protection (WP65) from SPU
Enable Secure Peripheral (SECUREP65) from SPU
Clocked by SCLK0_0

Figure 1-23: ACM System Diagram

Controller Area Network (CAN)

The processor contains a Controller Area Network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is compatible with the control applications. It can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

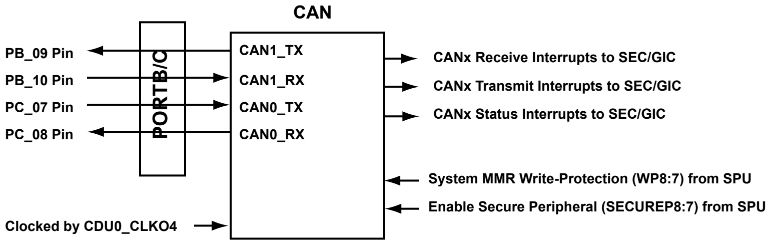

CAN
PB_09 Pin
PB_10 Pin
PC_07 Pin
PC_08 Pin
PORTB/C
CAN1_TX
CAN1_RX
CAN0_TX
CAN0_RX
CANx Receive Interrupts to SEC/GIC
CANx Transmit Interrupts to SEC/GIC
CANx Status Interrupts to SEC/GIC
System MMR Write-Protection (WP8:7) from SPU
Enable Secure Peripheral (SECUREP8:7) from SPU
Clocked by CDU0_CLKO4

Figure 1-24: CAN System Diagram

Mobile Storage Interface (MSI)

The Mobile Storage Interface (MSI) is a fast, synchronous controller that uses various protocols to communicate with MMC, SD, and SDIO cards. It addresses the growing storage need in embedded systems, hand held, and consumer electronics applications that require low power.

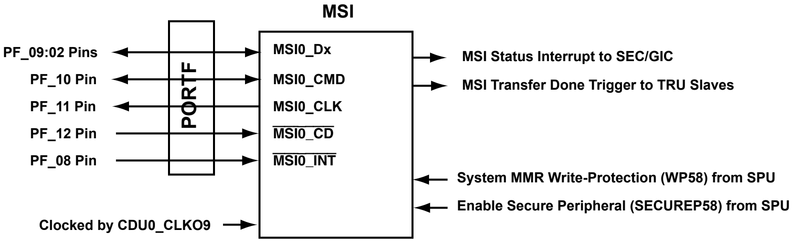

MSI
PF_09:02 Pins
PF_10 Pin
PF_11 Pin
PF_12 Pin
PF_08 Pin
PORTF
MSI0_Dx
MSI0_CMD
MSI0_CLK
MSI0_CD
MSI0_INT
MSI Status Interrupt to SEC/GIC
MSI Transfer Done Trigger to TRU Slaves
System MMR Write-Protection (WP58) from SPU
Enable Secure Peripheral (SECUREP58) from SPU
Clocked by CDU0_CLKO9

Figure 1-25: MSI System Diagram

Media Local Bus (MLB)

The Media Local Bus (MLB)® is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The MLB topology supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

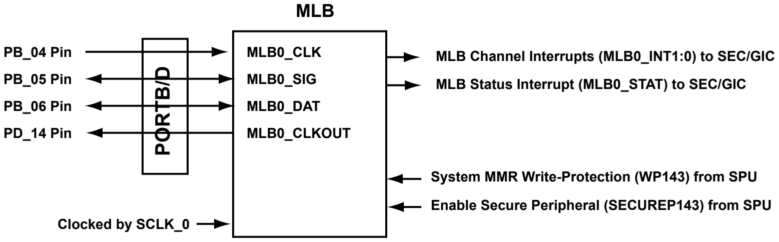

MLB
PB_04 Pin
PB_05 Pin
PB_06 Pin
PD_14 Pin
PORTB/D
MLB0_CLK
MLB0_SIG
MLB0_DAT
MLB0_CLKOUT
MLB Channel Interrupts (MLB0_INT1:0) to SEC/GIC
MLB Status Interrupt (MLB0_STAT) to SEC/GIC
System MMR Write-Protection (WP143) from SPU
Enable Secure Peripheral (SECUREP143) from SPU
Clocked by SCLK_0

Figure 1-26: MLB System Diagram

Ethernet Media Access Controller (EMAC)

The Ethernet Media Access Controller (EMAC) peripheral in the processor enables network connectivity to applications through an Ethernet interface.

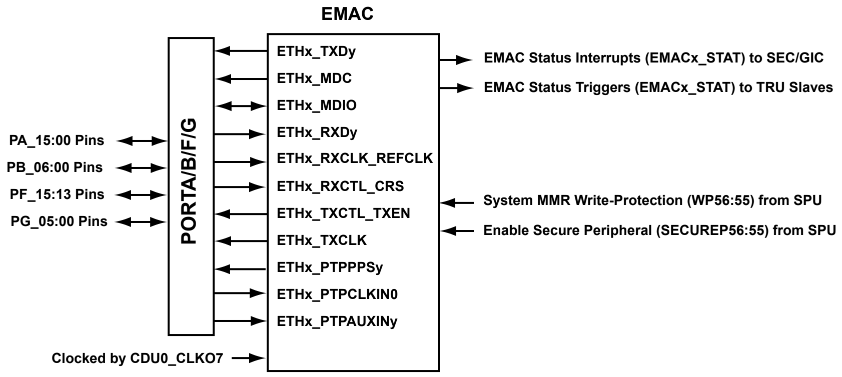

EMAC
ETHx_TXDy
ETHx_MDC
ETHx_MDIO
ETHx_RXDy
ETHx_RXCLK_REFCLK
ETHx_RXCTL_CRS
ETHx_TXCTL_TXEN
ETHx_TXCLK
ETHx_PTPPPSy
ETHx_PTPCLKIN0
ETHx_PTPAUXINy
PORTA/B/F/G
PA_15:00 Pins
PB_06:00 Pins
PF_15:13 Pins
PG_05:00 Pins
Clocked by CDU0_CLKO7
EMAC Status Interrupts (EMACx_STAT) to SEC/GIC
EMAC Status Triggers (EMACx_STAT) to TRU Slaves
System MMR Write-Protection (WP56:55) from SPU
Enable Secure Peripheral (SECUREP56:55) from SPU

Figure 1-27: EMAC System Diagram

Sinus Cardinalis (SINC) Filter

The sinus cardinalis (SINC) filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. See System Accelerators (FFT/FIR/IIR/HAE/SINC)The following sections provide information about the high-performance acceleration engines on the processor..

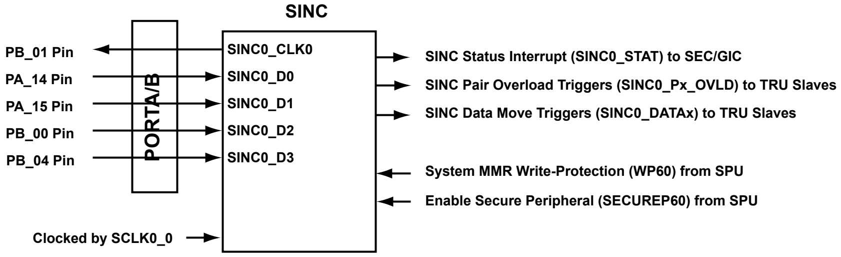

SINC
SINC0_CLK0
SINC0_D0
SINC0_D1
SINC0_D2
SINC0_D3
PORTA/B
PB_01 Pin
PA_14 Pin
PA_15 Pin
PB_00 Pin
PB_04 Pin
Clocked by SCLK0_0
SINC Status Interrupt (SINC0_STAT) to SEC/GIC
SINC Pair Overload Triggers (SINC0_Px_OVLD) to TRU Slaves
SINC Data Move Triggers (SINC0_DATAx) to TRU Slaves
System MMR Write-Protection (WP60) from SPU
Enable Secure Peripheral (SECUREP60) from SPU

Figure 1-28: SINC System Diagram

DAI/SRU Peripherals

The Digital Audio Interface (DAI) (DAIn) are comprised of groups of identical peripherals and their respective Signal Routing Units (SRUn). The SRU connects inputs and outputs of the DAI peripherals with each other and to the external pins. This configuration allows peripherals to be interconnected to accomodate a wide variety of systems without making external pin connections.

The DAI Routing Capabilities section provides an overview of the different routing capabilities for the DAI unit. An example for the eight SPORTs is shown in the following figure.

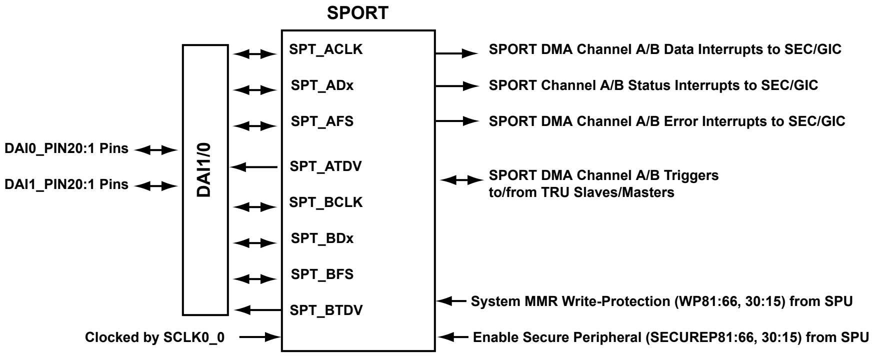

SPORT
SPT_ACLK
SPT_ADx
SPT_AFS
SPT_ATDV
SPT_BCLK
SPT_BDx
SPT_BFS
SPT_BTDV
DAI1/0
DAI0_PIN20:1 Pins
DAI1_PIN20:1 Pins
Clocked by SCLK0_0
SPORT DMA Channel A/B Data Interrupts to SEC/GIC
SPORT Channel A/B Status Interrupts to SEC/GIC
SPORT DMA Channel A/B Error Interrupts to SEC/GIC
SPORT DMA Channel A/B Triggers to/from TRU Slaves/Masters
System MMR Write-Protection (WP81:66, 30:15) from SPU
Enable Secure Peripheral (SECUREP81:66, 30:15) from SPU

Figure 1-29: SPORT Block Diagram

The following additional peripherals are connected using the DAI/SRU.

- Eight Asynchronous Sample Rate Converter (ASRC) blocks
- Two Sony/Philips Digital Interface (S/PDIF) transmit/receive blocks
- Four Precision Clock Generators (PCG)

Dedicated Pin Peripherals

The following peripherals have dedicated pins on the processor.

Two-Wire Interface (TWI)

The processor has a Two-Wire Interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

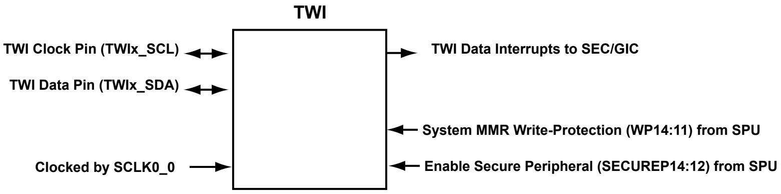

TWI
TWI Clock Pin (TWIx_SCL)
TWI Data Pin (TWIx_SDA)
Clocked by SCLK0_0
TWI Data Interrupts to SEC/GIC
System MMR Write-Protection (WP14:11) from SPU
Enable Secure Peripheral (SECUREP14:12) from SPU

Figure 1-30: TWI System Diagram

Universal Serial Bus (USB)

The Universal Serial Bus (USB) controller provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras, and MP3 players. It allows these devices to transfer data using a point-to-point USB connection without the need for a personal computer host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification.

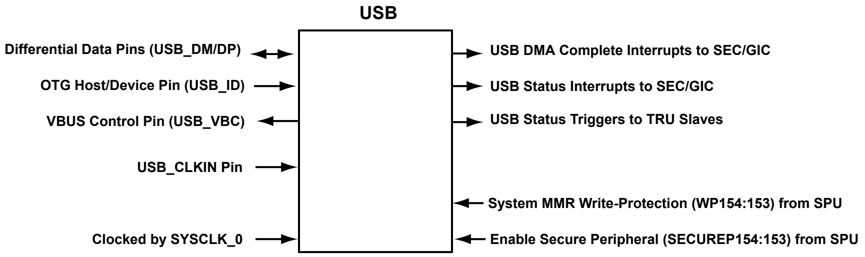

USB
Differential Data Pins (USB_DM/DP)
OTG Host/Device Pin (USB_ID)
VBUS Control Pin (USB_VBC)
USB_CLKIN Pin
Clocked by SYSCLK_0
USB DMA Complete Interrupts to SEC/GIC
USB Status Interrupts to SEC/GIC
USB Status Triggers to TRU Slaves
System MMR Write-Protection (WP154:153) from SPU
Enable Secure Peripheral (SECUREP154:153) from SPU

Figure 1-31: USB System Diagram

Media Local Bus (6-pin) (MLB)

The Media Local Bus (MLB) supports the MOST25, MOST50 and MOST150 standards and this document assumes familiarity with these standards. For more information, refer to the Media Local Bus specification version 4.2.

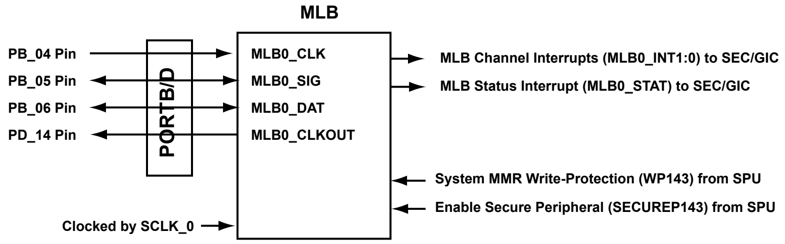

MLB
PB_04 Pin
PB_05 Pin
PB_06 Pin
PD_14 Pin
PORTB/D
MLB0_CLK
MLB0_SIG
MLB0_DAT
MLB0_CLKOUT
Clocked by SCLK_0
MLB Channel Interrupts (MLB0_INT1:0) to SEC/GIC
MLB Status Interrupt (MLB0_STAT) to SEC/GIC
System MMR Write-Protection (WP143) from SPU
Enable Secure Peripheral (SECUREP143) from SPU

Figure 1-32: MLB System Diagram

PCI Express

PCI Express is a high performance, general purpose I/O interconnect defined for a wide variety of computing and communication platforms.

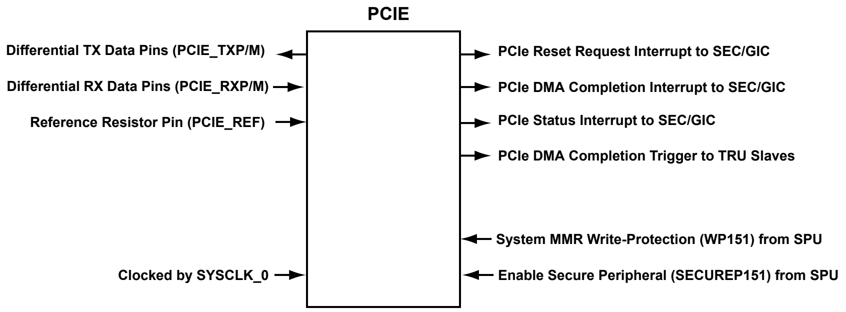

PCIE
Differential TX Data Pins (PCIE_TXP/M)
Differential RX Data Pins (PCIE_RXP/M)
Reference Resistor Pin (PCIE_REF)
Clocked by SYSCLK_0
PCIe Reset Request Interrupt to SEC/GIC
PCIe DMA Completion Interrupt to SEC/GIC
PCIe Status Interrupt to SEC/GIC
PCIe DMA Completion Trigger to TRU Slaves
System MMR Write-Protection (WP151) from SPU
Enable Secure Peripheral (SECUREP151) from SPU

Figure 1-33: PCIE System Diagram

Housekeeping ADC (HADC)

The Housekeeping ADC (HADC) is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters like voltages, temperatures in the system or for any general-purpose use as well.

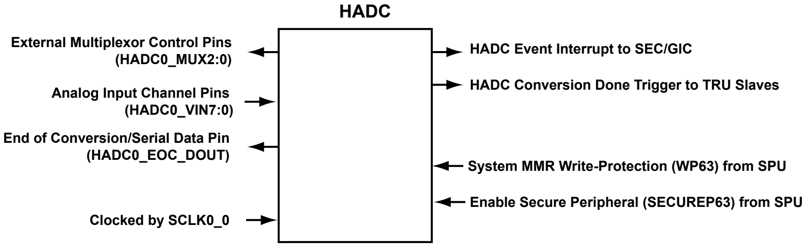

HADC
External Multiplexor Control Pins (HADC0_MUX2:0)
Analog Input Channel Pins (HADC0_VIN7:0)
End of Conversion/Serial Data Pin (HADC0_EOC_DOUT)
Clocked by SCLK0_0
HADC Event Interrupt to SEC/GIC
HADC Conversion Done Trigger to TRU Slaves
System MMR Write-Protection (WP63) from SPU
Enable Secure Peripheral (SECUREP63) from SPU

Figure 1-34: HADC System Diagram

System Accelerators (FFT/FIR/IIR/HAE/SINC)

The following sections provide information about the high-performance acceleration engines on the processor.

FFT Accelerator (FFTA)

The FFT Accelerator (FFTA) performs memory to memory FFT/IFFT operations without core software intervention. Additionally, the FFTA architecture allows execution of complex, pipelined, memory to memory algorithms including ping-ponged, windowed frequency domain filtering and very large FFTs. The FFTA may also be used in conjunction with minimal computation support from a core in applications such as the overlap-add operations required for large frequency domain based convolutions.

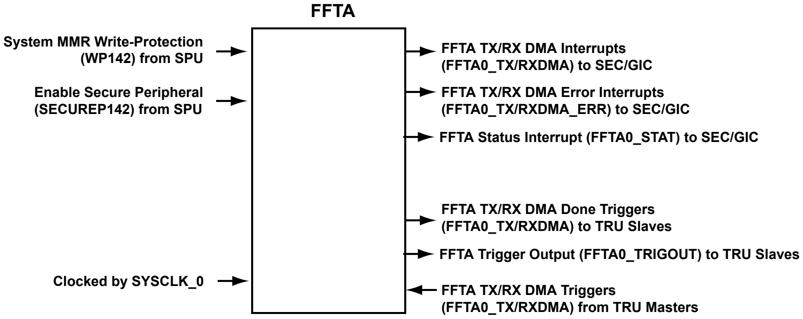

FFTA
System MMR Write-Protection (WP142) from SPU
Enable Secure Peripheral (SECUREP142) from SPU
Clocked by SYSCLK_0
FFTA TX/RX DMA Interrupts (FFTA0_TX/RXDMA) to SEC/GIC
FFTA TX/RX DMA Error Interrupts (FFTA0_TX/RXDMA_ERR) to SEC/GIC
FFTA Status Interrupt (FFTA0_STAT) to SEC/GIC
FFTA TX/RX DMA Done Triggers (FFTA0_TX/RXDMA) to TRU Slaves
FFTA Trigger Output (FFTA0_TRIGOUT) to TRU Slaves
FFTA TX/RX DMA Triggers (FFTA0_TX/RXDMA) from TRU Masters

Figure 1-35: FFTA System Diagram

FIR Accelerator (FIR)

FIR Accelerator (FIR) filters are frequently used in DSP applications. The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core. FIR filters are used in a wide array of applications including multi-rate processing with an interpolator or decimator.

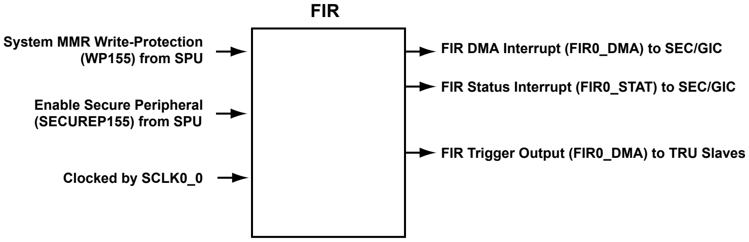

FIR
System MMR Write-Protection (WP155) from SPU
Enable Secure Peripheral (SECUREP155) from SPU
Clocked by SCLK0_0
FIR DMA Interrupt (FIR0_DMA) to SEC/GIC
FIR Status Interrupt (FIR0_STAT) to SEC/GIC
FIR Trigger Output (FIR0_DMA) to TRU Slaves

Figure 1-36: FIR System Diagram

IIR Accelerator (IIR)

The processor includes an IIR Accelerator (IIR) implemented in hardware that reduces the processing load on the core, freeing it up for other tasks.

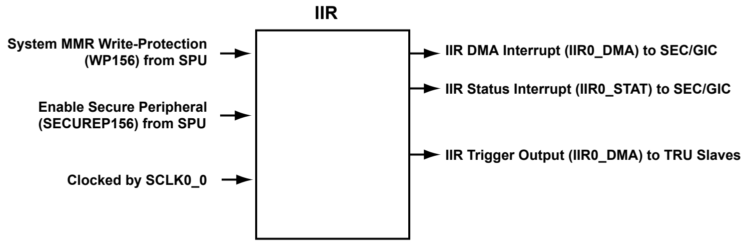

IIR
System MMR Write-Protection (WP156) from SPU
Enable Secure Peripheral (SECUREP156) from SPU
Clocked by SCLK0_0
IIR DMA Interrupt (IIR0_DMA) to SEC/GIC
IIR Status Interrupt (IIR0_STAT) to SEC/GIC
IIR Trigger Output (IIR0_DMA) to TRU Slaves

Figure 1-37: IIR System Diagram

Harmonic Analysis Engine (HAE)

The Harmonic Analysis Engine (HAE) analyzes harmonic frequencies present on the voltage and current input samples. The HAE receives input samples from two source channels whose frequencies are 45–65 Hz. The HAE then processes the input samples and produces output results. The output results consist of power quality measurements of the fundamental and up to 12 more harmonics.

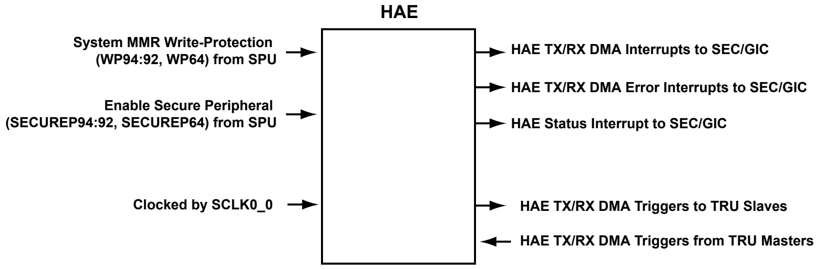

HAE
System MMR Write-Protection (WP94:92, WP64) from SPU
Enable Secure Peripheral (SECUREP94:92, SECUREP64) from SPU
Clocked by SCLK0_0
HAE TX/RX DMA Interrupts to SEC/GIC
HAE TX/RX DMA Error Interrupts to SEC/GIC
HAE Status Interrupt to SEC/GIC
HAE TX/RX DMA Triggers to TRU Slaves
HAE TX/RX DMA Triggers from TRU Masters

Figure 1-38: HAE System Diagram

Sinus Cardinalis (SINC) Filter

The Sinus Cardinalis (SINC) Filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator.

The filter consists of a set of integration and decimation stages implemented directly in logic for efficient execution. The SINC filter supports capture of current or voltage feedback signals from an isolating analog-to-digital converter (ADC). Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback; a secondary filter for overcurrent detection. The SINC module includes four filter channels and two modulator clock generators.

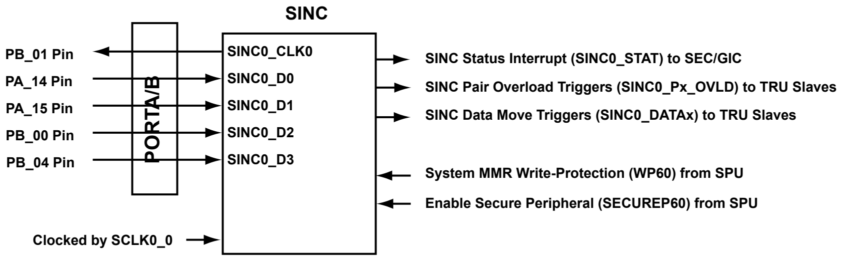

SINC
PB_01 Pin
PA_14 Pin
PA_15 Pin
PB_00 Pin
PB_04 Pin
PORTA/B
SINC0_CLK0
SINC0_D0
SINC0_D1
SINC0_D2
SINC0_D3
Clocked by SCLK0_0
SINC Status Interrupt (SINC0_STAT) to SEC/GIC
SINC Pair Overload Triggers (SINC0_Px_OVLD) to TRU Slaves
SINC Data Move Triggers (SINC0_DATAx) to TRU Slaves
System MMR Write-Protection (WP60) from SPU
Enable Secure Peripheral (SECUREP60) from SPU

Figure 1-39: SINC System Diagram

Security and Protection (SPU/PKTE/PKIC/PKA/TRNG)

The following modules provide system safety and security.

System Protection Unit (SPU)

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The System Protection Unit (SPU) restricts access to certain MMRs, similar to the functionality of a semaphore.

The SPU also protects peripherals based on security settings. It is part of the overall security infrastructure of the processor.

Security Packet Engine (PKTE)

The PKTE is a security packet engine designed to off-load the host processor to improve the speed of applications requiring cryptographic processing. The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation.

Public Key Accelerator (PKA)

The Public Key Accelerator (PKA) helps offload computationally-intensive operations commonly found in public key cryptography algorithms. The PKA also contains hardware logic to automatically zero out the PKA RAM buffer to clear out any information that is considered sensitive or secure.

Public Key Interrupt Controller (PKIC)

The Public Key Interrupt Controller (PKIC) is a common interrupt controller shared with the True Random Number Generator. The host processor configures the PKIC to generate interrupts when certain operations are complete or interrupts are caused by errors.

True Random Number Generator (TRNG)

The True Random Number Generator (TRNG) engine provides a true, non-deterministic, noise source for generating keys, Initialization Vectors (IVs), and other random number requirements. Other non-cryptographic purposes include statistical sampling, retry timers for communications protocols and noise generation.

Safety (WDOG/TMU/VMU)

Signal Watchdogs (WDOG)

The eight general-purpose Watchdog Timer (WDOG) timers feature modes to monitor off-chip signals. The Watchdog Period mode monitors whether external signals toggle with a period within an expected range. The Watchdog Width mode monitors whether the pulse widths of external signals are within an expected range. Both modes help to detect undesired toggling (or lack thereof) of system-level signals.

Thermal Monitor Unit (TMU)

The Thermal Monitoring Unit (TMU) unit senses the die temperature during runtime. In cases of weak temp violations an alert (interrupt) is given or in case for severe violations a HW reset is asserted to shut down the system.

Analog Subsystem (HADC)

The Housekeeping ADC (HADC) is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters like voltages, temperatures in the system or for any general-purpose use as well.

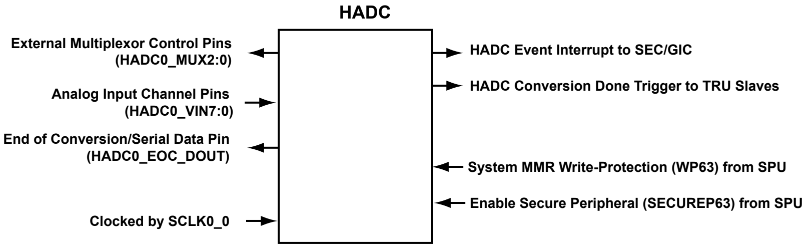

HADC
External Multiplexor Control Pins (HADC0_MUX2:0)
Analog Input Channel Pins (HADC0_VIN7:0)
End of Conversion/Serial Data Pin (HADC0_EOC_DOUT)
Clocked by SCLK0_0
HADC Event Interrupt to SEC/GIC
HADC Conversion Done Trigger to TRU Slaves
System MMR Write-Protection (WP63) from SPU
Enable Secure Peripheral (SECUREP63) from SPU

Figure 1-40: HADC System Diagram

System Debug (SCB/SWU/DBG/TAPC/CSPFT/STM)

The System Debug and Trace Unit (DBG) is based on ARM Core Sight technology. CoreSight™ is a set of architecture specifications defining debug and trace architecture. The processor uses CoreSight infrastructure to provide industry standard debug and trace capabilities through the following modules.

- Test Access Port Controller (TAPC)
- Trace Unit (CSPFT)
- System Trace Module (STM)

Additional debug resources are provided by the following modules.

- The System Crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable Quality of Service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.

- The System Watchpoint Unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

2 ARM Cortex-A5 Sub-System

The ADSP-SC589 processor includes an ARM® Cortex-A5® core. The ARM Cortex-A5 processor is the smallest, lowest cost and lowest power ARMv7 application processor. The A5 sub-system in the ADSP-SC589 processor includes a Floating-Point Unit, NEON Media Processing Engine, Generic Interrupt Controller and a Level 2 Cache Controller. The A5 also includes support for a L1-Cache sub-system and a full-fledged Memory Management Unit. The A5 implements the ARMv7 architecture and runs 32-bit ARM instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java byte-codes in Jazelle state.

This document describes the ARM Cortex-A5 core and memory architecture used on the ADSP-SC58x processor, but does not provide detailed programming information for the ARM processor. For more information about programming the ARM processor, visit the ARM Information Center:

- http://infocenter.arm.com.

The applicable documentation for programming the ARM Cortex-A5 processor include:

- Cortex-A5 Technical Reference Manual, Revision: r0p1
- Cortex-A Series Programmer's Guide, Revision: r0p1
- Cortex-A5 NEON Media Processing Engine Technical Reference Manual, Revision: r0p1
- Cortex-A5 Floating-Point Unit Technical Reference Manual, Revision: r0p1
- CoreLink Level 2 Cache Controller L2C-310 Technical Reference Manual, Revision: r3p3
- PrimeCell Generic Interrupt Controller (PL390) Technical Reference Manual, Revision: r0p0

Cortex A5 Features

The Cortex-A5 Sub-system has the following features.

- Thumb / ARM Instruction support
- L1- Instruction Cache and L1-Data Cache
- Floating Point Unit (FPU)
- NEON Media Processing Engine (NEON)

- Generic Interrupt Controller (GIC)
- Level 2 Cache Controller (L2CC)
- Memory Management Unit (MMU)

Functional Description

The following sections provide information on the function of the sub-system.

A5 Block Diagram

The following figure shows the primary blocks of the Cortex A5 sub-system. The performance increases with accesses to lower levels of memory as follows:

1. Level 1 – cache on-chip, separate data/code (highest)
2. Level 2 – cache on-chip, unified
3. Level 3 – memory external, (lowest)

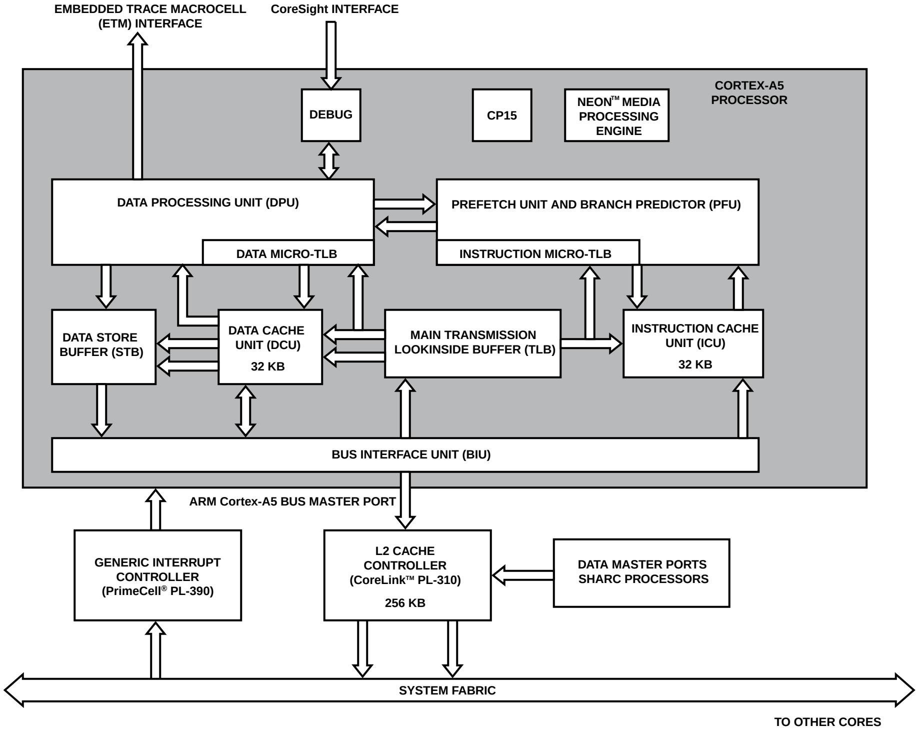

EMBEDDED TRACE MACROCELL (ETM) INTERFACE
CoreSight INTERFACE
CORTEX-A5 PROCESSOR
DEBUG
CP15
NEON™ MEDIA PROCESSING ENGINE
DATA PROCESSING UNIT (DPU)
PREFETCH UNIT AND BRANCH PREDICTOR (PFU)
DATA MICRO-TLB
INSTRUCTION MICRO-TLB
DATA STORE BUFFER (STB)
DATA CACHE UNIT (DCU) 32 KB
MAIN TRANSMISSION LOOKINSIDE BUFFER (TLB)
INSTRUCTION CACHE UNIT (ICU) 32 KB
BUS INTERFACE UNIT (BIU)
ARM Cortex-A5 BUS MASTER PORT
GENERIC INTERRUPT CONTROLLER (PrimeCell® PL-390)
L2 CACHE CONTROLLER (CoreLink™ PL-310) 256 KB
DATA MASTER PORTS SHARC PROCESSORS
SYSTEM FABRIC
TO OTHER CORES

Figure 2-1: A5 Sub-System Block Diagram

Control Co-Processor (CP15)

The system control co-processor, CP15, controls and provides status information for the functions implemented in the processor. The main functions of the system control co processor are:

- Overall system control and configuration
- MMU configuration and management
- Cache configuration and management
- System performance monitoring

All system architecture functions are controlled by reading or writing a general purpose processor register (Rt) from or to a set of registers (CRn) located within co-processor 15. The Op1, Op2, and CRm fields of the instruction can also be used to select registers or operations.

- MRC p15, Op1, Rt, CRn, CRm, Op2; read a CP15 register into an ARM register
- MCR p15, Op1, Rt, CRn, CRm, Op2; write a CP15 register from an ARM register

L1 Cache

The Cortex-A5 processor has separate instruction and data caches that run at ARM Core clock speed. The caches have the following features:

- L1-Data Cache size 32 KB
- L1-Instruction Cache size 32 KB
- Each cache can be disabled independently, using the system control coprocessor
- Cache replacement policy is pseudo random
- Data cache is 4-way set-associative
- Instruction cache is 2-way set-associative
- The cache line length is eight words.
- On a cache miss, critical word first filling of the cache is performed.

Prefetch Unit (PFU)

The PFU implements a two-level prediction mechanism, comprising the following:

- A 256 entry branch pattern history table
- A four-entry BTAC
- A four-entry return stack

Memory-Management Unit (MMU)

The ARM MMU is responsible for translating addresses of code and data from the virtual view of memory to the physical addresses in the real system. The translation is carried out by the MMU hardware and is transparent to the application. In addition, the MMU controls such things as memory access permissions, memory ordering and cache policies for each region of memory.

The MMU enables tasks or applications to be written in a way that requires them to have no knowledge of the physical memory map of the system, or about other programs that might be running simultaneously. This enables you to use the same virtual memory address space for each program. It also lets you work with a contiguous virtual memory map, even if the physical memory is fragmented. This virtual address space is separate from the actual physical map of memory in the system. Applications are written, compiled and linked to run in the virtual memory space. Virtual addresses are those used by you, and the compiler and linker, when placing code in memory. Physical addresses are those used by the actual hardware system.

The first level MMU uses a Harvard design with separate micro TLB structures in the PFU for instruction fetches and in the DPU for data read and write requests. A miss in the micro TLB results in a request to the main unified TLB shared between the data and instruction sides of the memory system. The TLB consists of a 128-entry two-way set-associative RAM based structure. The TLB page-walk mechanism supports page descriptors held in the L1 data cache. The caching of page descriptors is configured globally for each translation table base register, TTBRx, in the system coprocessor, CP15.

Page table entries support:

- 16 MB super sections
- 1 MB sections
- 64 KB large pages
- 4 KB small pages

NOTE: Virtual Memory translation tables are typically created by operating systems, and are often dynamically managed by the memory management layer. However, even a bare metal system can enable the MMU. For this, a flat mapping technique is used, where all virtual memory addresses shall be programmed exactly same as the physical memory addresses in the system.

NOTE: In order to utilize L1-Data Cache, application has to enable the MMU, via Co-Processor 15 in the ARM Core. After MMU and L1-Data Cache are enabled via CP15: SCTLR, application can disable / enable cache for individual pages / sections.

L2 Cache

The Level 2 Cache Controller in the ADSP-SC589 is a CoreLink Level 2 Cache Controller (L2C-310) from ARM and is clocked at SYSCLK speed. The addition of an on-chip secondary cache, also referred to as a Level 2 or L2 cache, is a recognized method of improving the performance of ARM-based systems when significant memory traffic is generated by the processor. By definition a secondary cache assumes the presence of a Level 1 or primary cache, closely coupled or internal to the processor. The cache controller is a unified, physically addressed, physically tagged cache. It includes the following features:

- 256 KB total size
- Lockdown by Line / Way / Master
- Fixed line length of 32 bytes, eight words or 256 bits

- Direct mapped to 8-way associativity (fixed)
- Prefetching capability
- Event monitoring
- Software option to enable exclusive cache configuration
- Additional Buffers:
  - Line Fill Buffers (LFBs)
  - Line Read Buffers (LRBs)
  - Eviction Buffers (EBs)
  - Store Buffers (STBs)
- TrustZone support, with the following features:
  - Non-Secure (NS) tag bit added in tag RAM and used for lookup in the same way as an address bit. The NS-tag bit is added in all buffers.
  - NS bit in Tag RAM used to determine security level of evictions to L3.
  - Restrictions for NS accesses for control, configuration, and maintenance registers to restrict access to secure data.
- Parity Support

NOTE: The L2CC Address Filtering registers should not be programmed by user. Not retaining the reset values can give unpredictable results.

Sharing L2 Cache with SHARC+ Cores

The L2CC (PL310) supports two master and two slave ports. The SHARC+ core can access the L2-Cache without bank conflict versus the Cortex A5 core by programming the L2CC registers. The cache access is restricted to the address range: from `CMMR_L2CC_START` [31:0] to `CMMR_L2CC_END` [31:0]. For more information, see the SHARC+ Core Programming Reference.

NOTE: There is no guarantee for the data coherency between A5 and SHARC+ cores.

NOTE: Programs should perform L2 cache write-back invalidation before changing the value of `CMMR_L2CC_START` and `CMMR_L2CC_END`.

Floating-Point Unit (FPU)

The Cortex-A5 FPU is a VFPv4-D16 implementation of the ARMv7 floating-point architecture. It provides low-cost high performance floating-point computation. The FPU supports all addressing modes and operations described in the *ARM Architecture Reference Manual*.

The features in the FPU are as follows.

- Support for single-precision and double-precision floating-point formats
- Support for conversion between half-precision and single-precision
- Support for Fused Multiply Accumulate (FMA) operations
- Normalized and de-normalized data are all handled in hardware
- Trap-less operation enabling fast execution

NeON

The Cortex-A5 NEON MPE extends the Cortex-A5 functionality to provide support for the ARM v7 Advanced SIMD v2 and Vector Floating-Point v4 (VFPv4) instruction sets. The Cortex-A5 NEON MPE supports all addressing modes and data-processing operations described in the *ARM Architecture Reference Manual*.

The Cortex-A5 NEON MPE features are:

- SIMD and scalar single-precision floating-point computation
- scalar double-precision floating-point computation
- SIMD and scalar half-precision floating-point conversion
- SIMD 8, 16, 32, and 64-bit signed and unsigned integer computation
- 8 or 16-bit polynomial computation for single-bit coefficients
- structured data load capabilities
- Large, shared register file, addressable as:
    - 32 32-bit S (single) registers
    - 32 64-bit D (double) registers
    - 16 128-bit Q (quad) registers
- The operations include:
    - Addition and subtraction
    - Multiplication with optional accumulation
    - Maximum or minimum value driven lane selection operations
    - Inverse square-root approximation
    - Comprehensive data-structure load instructions, including register-bank-resident table lookup

See the *ARM Architecture Reference Manual* for details of the extension register set.

Generic Interrupt Controller (GIC)

The GIC is an ARM Architecture compliant System-on-Chip (SoC) peripheral. It is a high-performance, area-optimized interrupt controller. The GIC implements the ARM Generic Interrupt Controller Architecture. The GIC takes interrupts asserted at the system level and signals them to each connected processor as appropriate. The GIC has the following features.

- Registers for managing interrupt sources, interrupt behavior, and interrupt routing to one or more processors
- Support for the ARM architecture Security Extensions
- Support for enabling, disabling, and generating processor interrupts from hardware (peripheral) interrupt sources
- Support for generating software interrupts
- Support for interrupt masking and prioritization

Refer to GIC Overview for more information on the ADSP-SC58x specific configuration of GIC.

Table 2-1: ADSP-SC58x *GIC Interrupt List*

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
GIC	Software Interrupt 0, core reset	N/A	0	GIC_SOFT00
GIC	Software Interrupt 1, undefined instruction	N/A	1	GIC_SOFT01
GIC	Software Interrupt 2, supervisor call	N/A	2	GIC_SOFT02
GIC	Software Interrupt 3, prefetch abort	N/A	3	GIC_SOFT03
GIC	Software Interrupt 4, data abort	N/A	4	GIC_SOFT04
GIC	Software Interrupt 5, reserved	N/A	5	GIC_SOFT05
GIC	Software Interrupt 6, IRQ interrupt	N/A	6	GIC_SOFT06
GIC	Software Interrupt 7, FIQ interrupt	N/A	7	GIC_SOFT07
CORE	Cortex A5 L2 Cache	22	54	C0_L2CC
CORE	Cortex A5 L1 Parity	23	55	C0_L1_PERR
CTI	Cortex A5 CTI 0	Reserved	284	ECT_C0_EVT
PMU	Cortex A5 Performance Monitoring	Reserved	285	C0_PMUIRQ

A5 Configurations

The following are the Cortex A5 and ADSP-SC58x configurations.

A5 Configurations

Table 2-2: A5 Core Configuration

Core Feature	Comment
JAZELLE Support	Implemented
NEON Engine	Implemented
FPU	Implemented
Instruction cache size	32 KB
Data cache size	32 KB

A5 Configuration Signals

Table 2-3: A5 Configuration Signals

Configuration	A5 TRM Signal Name	Comment
Default Exception Handler Endianness	CFGEND	Little Endian
CPU ID field	CLUSTERID[3:0]	4'b0000
Disable Write access to some CP15 registers	CP15SDISABLE	Not Enabled
Default exception handling state	TEINIT	ARM Mode
Exception vectors' location at reset	VINITHI	start exception vectors at address 0x00000000
Disable invalidate entire data cache, instruction cache and TLB at reset	L1RSTDISABLE	Disabled
Enable the RAM interface clamps	CPURAMCLAMP	clamps not active

A5 Power Modes

Table 2-4: ARM Core Power Modes

Mode	Comment
Run mode	Supported
Standby mode	Supported
Dormant mode	Not Supported
Shutdown mode	Not Supported

L2CC Configuration Signals

Table 2-5: L2CC Configuration Signals

Configuration	L2CC TRM Signal Name	Comment
Associativity	ASSOCIATIVITY	8-Way

Table 2-5: L2CC Configuration Signals (Continued)

Configuration	L2CC TRM Signal Name	Comment
Cache controller cache ID	CACHEID[5:0]	0
Address filtering Enable out of reset	CFGADDRFILTEN	Enabled
Address filtering End Address out of reset	CFGADDRFILTEND[11:0]	0xFFF
Address filtering Start Address out of reset	CFGADDRFILTSTART[11:0]	0x201
Endian mode for accessing configuration registers out of reset	CFGBIGEND	Little-endian
Base address for accessing configuration registers	REGFILEBASE[19:0]	0x10000
Size of ways	WAYSIZE[2:0]	32 KB

L2CC Power Down Modes

Table 2-6: L2CC Power Down Modes

Mode	Comment
Run mode	Supported
Dynamic Clock Gating	Supported
Standby mode	Not Supported
Dormant mode	Not Supported
Shutdown mode	Not Supported

L2CC Configuration

Table 2-7: L2CC Configuration

Feature	Comment
Cache way size	32 KB
Associativity	8 Ways
Default RAM latencies	2 cycles
DATA RAM banking	Disabled
Slave port 1 present	Enabled
Master port 1 present	Enabled
Parity logic	Enabled
Lock down by master	Enabled
Lock down by line	Enabled
Address filtering	Enabled
Speculative reading	Disabled

3 Clock Generation Unit (CGU)

The Clock Generation Unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a master clock that runs at a frequency that is a multiple of the CLKIN input clock frequency. The PCU divides down the master clock to generate various system clocks and synchronization signals.

CGU Features

The CGU module supports the following features:

- Provides smooth transitions from the current clock condition to a new condition with PLL logic and executes the changes to clocks due to register programming
- Provides PLL and clock domain status reporting for event management
- Supports the capability to bypass the PLL for power savings
- Manages power dynamically through software, allowing the dynamic control of the core clock frequency (f_{CCLK}) of the processor
- Controls clock gating of core and system clocks

NOTE: For more information about processor-specific CGU features, see the processor data sheet.

CGU Functional Description

The CGU generates all on-chip clocks and synchronization signals based on the programmed PLL multiplication factor and dividers. The CGU provides the following functionality.

Change the PLL clock frequency

The CGU allows programs to change the PLL clock frequency by writing new values to bits in the control register. Any time the PLL relocks, the CGU aligns all core and system clocks.

Change other clock frequencies

The CGU allows programs to change the CCLKn, SYSCLK, SCLKn, DCLK, and OCLK frequencies by writing values to the `CGU_DIV` register. Any time the clock frequency is changed, the OCLK, CCLKn, SYSCLK, DCLK and SCLKn clocks exit the frequency change sequence aligned.

Perform clock alignment

The CGU can align all clocks by writing to the `CGU_DIV` register. This function aligns all PLL-based clocks.

For more information on these functions, see the CGU Programming Model section.

ADSP-SC58x CGU Register List

The clock generation unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock, running at a frequency that is a multiple of the CLKIN input clock's frequency. The CGU also generates all on-chip clocks and synchronization signals. The PCU permits application software control of the PLL's operation. A set of registers govern CGU operations. For more information on CGU functionality, see the CGU register descriptions.

Table 3-1: ADSP-SC58x CGU Register List

Name	Description
`CGU_CCBF_DIS`	Core Clock Buffer Disable Register
`CGU_CCBF_STAT`	Core Clock Buffer Status Register
`CGU_CLKOUTSEL`	CLKOUT Select Register
`CGU_CTL`	Control Register
`CGU_DIV`	Clocks Divisor Register
`CGU_OSCWDCTL`	Oscillator Watchdog Register
`CGU_PLLCTL`	PLL Control Register
`CGU_REVID`	Revision ID Register
`CGU_SCBF_DIS`	System Clock Buffer Disable Register
`CGU_SCBF_STAT`	System Clock Buffer Status Register
`CGU_STAT`	Status Register
`CGU_TSCOUNT0`	Time Stamp Counter 32 LSB Register
`CGU_TSCOUNT1`	Time Stamp Counter 32 MSB Register
`CGU_TSCTL`	Time Stamp Control Register
`CGU_TSVALUE0`	Time Stamp Counter Initial 32 LSB Value Register
`CGU_TSVALUE1`	Time Stamp Counter Initial MSB Value Register

ADSP-SC58x CGU Interrupt List

Table 3-2: ADSP-SC58x CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
1	CGU0_EVT	CGU0 Event	Edge	
2	CGU1_EVT	CGU1 Event	Edge	

ADSP-SC58x CGU Trigger List

Table 3-3: ADSP-SC58x CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	`CGU0_EVT`	CGU0 Event	Edge
2	`CGU1_EVT`	CGU1 Event	Edge

Table 3-4: ADSP-SC58x CGU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CGU Definitions

DPM

The dynamic power management (DPM) works with the CGU to provide flexible power dissipation modes for the processor.

PCU

The PLL control unit (PCU) in the CGU controls PLL operations. All the MMR registers of the CGU are implemented in this unit.

PLL

The phase-locked loop (PLL) operates within the CGU.

RCU

The reset control unit (RCU) provides input to the CGU to manage clocks during processor reset.

CDU

The clock distribution unit distributes the clocks from the CGU to different clock domains

CGU

The clock generation unit (CGU) is comprised of the PLL and PCU. The CGU generates the clocks listed in the *Clock Descriptions* table.

Table 3-5: Clock Descriptions

Clock	Description
CCLK0_0	CCLK0 derived from CGU0
CCLK1_0	CCLK1 derived from CGU0
SYSCLK_0	SYSCLK derived from CGU0
SCLK0_0	SCLK0 derived from CGU0
SCLK1_0	SCLK1 derived from CGU0
DCLK_0	DCLK derived from CGU0
OCLK_0	OCLK derived from CGU0
CCLK0_1	CCLK0 derived from CGU1
CCLK1_1	CCLK1 derived from CGU1
SYSCLK_1	SYSCLK derived from CGU1
SCLK0_1	SCLK0 derived from CGU1
SCLK1_1	SCLK1 derived from CGU1
DCLK_1	DCLK derived from CGU1
OCLK_1	OCLK derived from CGU1

CGU PLL Block Diagram

The *CGU PLL Block Diagram* provides a top-level block diagram of the phase locked loop (PLL). The main blocks of the PLL are the phase/frequency detector (PFD), the charge pump, the loop filter, and the voltage controlled oscillator (VCO). The VCO multiplies the SYS_CLKINx input to a higher frequency.

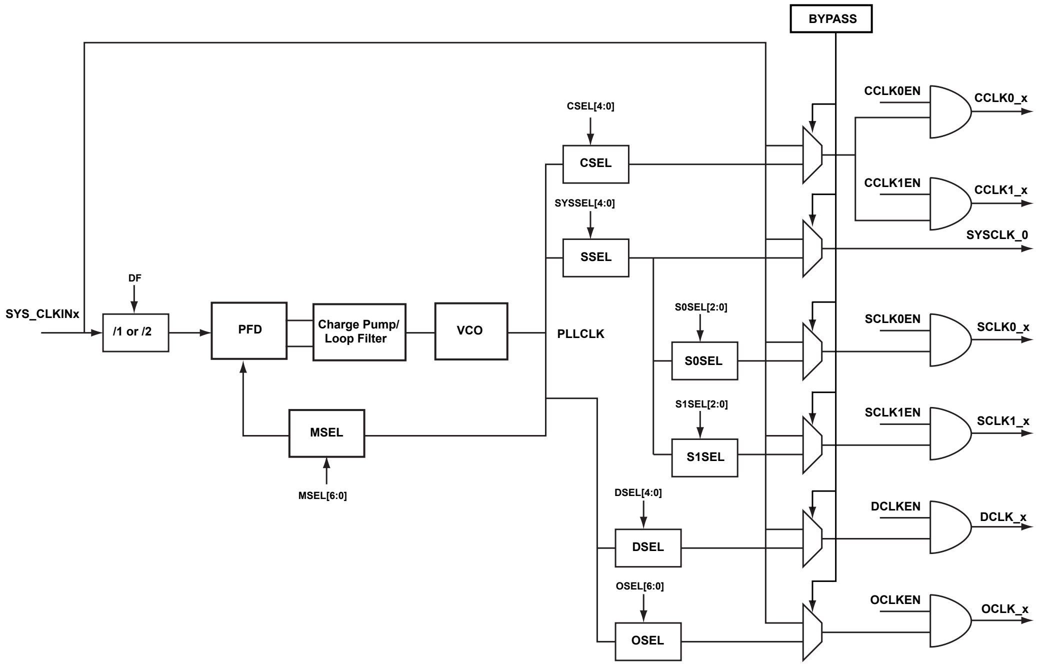

BYPASS
CSEL[4:0]
CSEL
SYSSEL[4:0]
SSEL
DF
SYS_CLKINx
/1 or /2
PFD
Charge Pump/ Loop Filter
VCO
PLLCLK
S0SEL[2:0]
S0SEL
S1SEL[2:0]
S1SEL
MSEL
MSEL[6:0]
DSEL[4:0]
DSEL
OSEL[6:0]
OSEL
CCLK0EN
CCLK0_x
CCLK1EN
CCLK1_x
SYSCLK_0
SCLK0EN
SCLK0_x
SCLK1EN
SCLK1_x
DCLKEN
DCLK_x
OCLKEN
OCLK_x

Figure 3-1: CGU PLL Block Diagram (x=0 CGU0, x=1 CGU1)

The ***SYS_CLKOUT Generation*** figure is a conceptual representation of the CLKOUT module. Different clocks that originate from the CGU blocks are available on the SYS_CLKOUT output pin. The selection of the clock output on the SYS_CLKOUT pin is controlled by the `CGU_CLKOUTSEL.CLKOUTSEL` bit field.

Additional configuration options are configured using the Clock Distribution Unit. See CDU Clock Configuration Options.

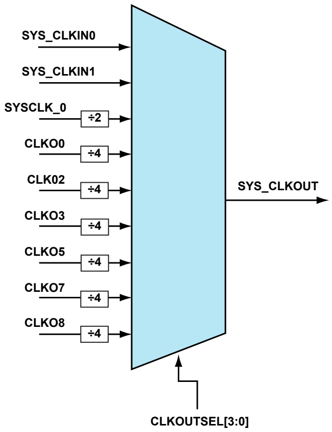

SYS_CLKIN0
SYS_CLKIN1
SYSCLK_0
÷2
CLKO0
÷4
CLK02
÷4
CLKO3
÷4
CLKO5
÷4
CLKO7
÷4
CLKO8
÷4
SYS_CLKOUT
CLKOUTSEL[3:0]

Figure 3-2: SYS_CLKOUT Generation

NOTE: The processor supports two PLLs (CGU1–0), see the ***CDU Block Diagram*** figure in the Clock Distribution (CDU) chapter. The configuration of CGU0 is mandatory because it provides the clock to the chip infrastructure (Fabric, L2 cache, L2 system, SEC and GIC) and to some high speed peripherals.

CGU Operating Modes

The CGU does not have configurable operating modes, but CGU operations affect the operating modes of other modules. Some CGU operation issues that affect operation of other modules include the following:

- The PLL of the CGU operates in either normal mode (CGU clock divisors applied) or bypass mode (CGU PLL is bypassed and clock divisors ignored).
- The SCB uses the CGU for clock synchronization across clock domains. For more information, see System Crossbars (SCB).
- The DPM uses the CGU for clock management as power state transitions occur. For more information, see the Dynamic Power Management (DPM) chapter.
- The CGU uses clock gating control to obtain flexible low-power modes.

CGU Power-up Sequence

See the product data sheet for exact power-up requirements. The processor is configured to come up in clock bypass mode. The programs is required to configure full speed clocks and safety monitors. CLKIN0 and all supplies should be stable before the `SYS_HWRST` signal is deasserted. CLKIN1 should be stable before operation of OCU safety unit.

CGU Event Control

The CGU generates an event or error for several different reasons.

CGU Events

After a frequency change, a CGU event indicates that the PLL has locked and clocks are synchronized. If a core was idled while changing frequencies, the CGU can use an event interrupt to break the core idle. While in active mode, a CGU event indicates that the PLL has locked.

CGU Errors

A CGU error occurs under following conditions:

- A write access to the `CGU_DIV` register triggers an alignment sequence while the PLL is locked and is still aligning the clocks.

  The `CGU_STAT.WDIVERR` bit state indicates this error. If this error occurs, clear the `CGU_STAT.WDIVERR` bit and rewrite the desired values to the `CGU_DIV` register.
- A change to the `CGU_DIV` register occurs while the PLL is locked and is still aligning the clocks

  The `CGU_STAT.WDIVERR` bit state indicates this error. If this error occurs, clear the `CGU_STAT.WDIVERR` bit and rewrite the desired values to the `CGU_DIV` register.

- A write access to the `CGU_CTL.DF` bit field occurs or a write access to the `CGU_CTL.MSEL` bit field occurs while the PLL is locking.

  The `CGU_STAT.WDFMSERR` bit state indicates this error. If this error occurs, wait until the PLL has finished locking, clear the error, and rewrite the desired value change.

- A clock divisor value error occurs when the *CCLK* divisor is greater than the *SYSCLK* divisor. For example, the `CGU_DIV.CSEL` is greater than `CGU_DIV.SYSSEL`.

  The `CGU_STAT.WDIVERR` bit state indicates this error. If this error occurs, clear it. The CGU writes new values to the `CGU_DIV.CSEL` bit field, so the field is less than or equal to the `CGU_DIV.SYSSEL` bit field value.

The CGU monitors changes to the following fields:

- CCLK Divisor - `CGU_DIV.CSEL`
- SYSCLK Divisor - `CGU_DIV.SYSSEL`

CGU Generated Bus Errors

The CGU generates a bus error when a read or write transaction is attempted to an unused address within the CGU address range. It also generates a bus error when a misaligned access is made to a CGU register. In addition to the bus error, the `CGU_STAT.ADDRERR` bit is set. If a write to a write-protected CGU register is attempted, the CGU generates a bus error. In addition, the `CGU_STAT.LWERR` bit is set.

Oscillator Watchdog

The oscillator watchdog detects the absence of input clock transitions and provides a fault warning through the `SYS_FAULT` pin. To detect harmonic or subharmonic crystal oscillator behavior, the watchdog (under programmable control) also detects and reports input oscillator frequencies above and below the specified limits. Use an internal asynchronous, local 1-MHz oscillator combined with a series of programmable counters for this detection. Set the `CGU_OSCWDCTL.MONDIS` bit and clear the `CGU_OSCWDCTL.FAULTEN` bit in the control register to optionally disable all the input clock monitor and fault detection functions.

Set the `CGU_OSCWDCTL.HODEN` bit to enable harmonic oscillation detection. The CGU uses the `CGU_OSCWDCTL.HODF` bit field to indicate the desired lower fail limit for the harmonic oscillation detection in MHz. The upper limit is always twice the lower limit. The ***HODF Settings for Different Input Clock Frequencies*** table shows an example of the `CGU_OSCWDCTL.HODF` bit settings for different input clock frequencies.

Table 3-6: HODF Settings for Different Input Clock Frequencies

CGU_OSCWDCTL.HODF[5:0]	Subharmonic Frequency (MHz)	Nom. Lower Fail Limit (MHz)	Input Clock Frequency (MHz)	Nom. Upper Fail Limit (MHz)	Second Harmonic Frequency (MHz)
14	10	14	20	28	40
21	15	21	30	42	60

The CGU uses the `CGU_OSCWDCTL.BOUF` asynchronous control bit field to indicate the desired upper fail limit for the bad oscillation detection. Set the `CGU_OSCWDCTL.BOUEN` bit to enable upper-limit bad oscillation detection. A bad oscillation detection condition signals a fault before any processor operations occur. This detection occurs (even in bypass mode) whenever a clock frequency exceeds its specifications.

The `CGU_OSCWDCTL.BOUF` =0 operation starts with a target of 32 MHz and each additional LSB increases the frequency test limit by 2 MHz. For example:

Target Upper Frequency Limit = `CGU_OSCWDCTL.BOUF` × 2 MHz + 32 MHz

The `CGU_STAT.OSCWDSTATFC` status bits indicate the nature of the fault. The ***Fault Map*** table shows the fault values.

Table 3-7: Fault Map

`CGU_STAT.OSCWDSTATFC` Bitfield Values	Fault Type
0	No Fault
1	No Input Clock
2	Subharmonic CLKIN
3	Harmonic CLKIN
4	No AUX_CLK
5	CLKIN > Upper Freq Limit (BOUF)
6	Reserved
7	Multiple Limit Faults

There is a priority to the faults given in the case of multiple fault errors. The highest priority is given to No Input clock followed by No AUX_CLK. The other three fault cases share the lowest priority. Multiple limit faults are asserted if more than one type of subharmonic CLKIN, harmonic CLKIN, or BOUF faults are observed.

NOTE: All the `CGU_STAT.OSCWDSTATFC` faults other than the absence of AUX_CLK (for example, `CGU_STAT.OSCWDSTATFC` =4) are not reliable and used for debug only.

NOTE: Program and enable the OSCWDOG to match the actual crystal, before bringing the PLL out of bypass.

CGU Programming Model

The programming model for the CGU involves the various mode configuration techniques.

Configuring CGU Modes

Use the following procedures to configure the clocks and PLL.

NOTE: The program needs to perform the following sequence only once, after coming out of reset, inside the application, before changing clocks. This sequence clears the `CGU_STAT.CLKSALGN` bit:

```
*pREG_CGU0_PLLCTL |= BITM_CGU_PLLCTL_PLLBPCL; // come out of bypass and enter
Full ON
while( (pADI_CGU0 ->STAT & 0xF) != 0x5 ) { } // poll
// now clocks are running with hardware default divisors.
// now program can change frequencies If desired the program can put the PLL
again into bypass.
```

Changing Clock Frequencies

Applications change clock frequencies in two ways. The first way is modifying the PLL multiplication value by writing to the `CGU_CTL` register and the second is modifying the clock dividers by writing to the `CGU_DIV` register. Both actions have different implications even if the frequencies of the final clock are the same. Write accesses to change the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit fields while the PLL is locking set the `CGU_STAT.WDFMSERR` error bit. The `CGU_STAT.WDIVERR` error bit is set when one of following accesses is attempted while the PLL is locked, but still aligning the clocks:

- A write access to the `CGU_DIV` register to trigger an alignment sequence
- A write access to the `CGU_DIV` register to change the `CGU_DIV.CSEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.S1SEL`, or `CGU_DIV.DSEL` bits

Read-after-write accesses to these registers return the new value, even if the frequency of the clock change is still in-progress.

Modifying the PLL multiplier requires the PLL to relock. Once the PLL locks, the CGU synchronizes the clocks. Changes to the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit field result in bypassing the PLL. By setting the `CGU_CTL.WFI` bit, programs force the PLL to wait for all the cores to return to their idle or reset states before the frequency changes. If necessary, clear the `CGU_DIV.UPDT` bit to avoid multiple clock alignment sequences. If the `CGU_DIV` register is not updated, the CGU uses the current values to determine the frequencies of the clock. It is the programs responsibility to guarantee that the new `CGU_CTL.DF` or `CGU_CTL.MSEL` and `CGU_DIV` combinations are legal.

Changing the PLL Clock Frequency

To change the phase-locked loop clock (*PLLCLK*) frequency, write new values to the `CGU_CTL.MSEL` field or `CGU_CTL.DF` field. Any time the PLL relocks, all core and system clocks are aligned.

1. Read `CGU_STAT` register and verify that:
    a. The `CGU_STAT.PLLEN` bit =1 (PLL enabled)
    b. The `CGU_STAT.PLOCK` bit =1 (PLL is not locking)
    c. The `CGU_STAT.CLKSALGN` bit =0 (clocks aligned)
2. Write the desired values to the clock divisor select fields of the `CGU_DIV` register with the `CGU_DIV.UPDT` bit =0.
3. Write the desired values to the `CGU_CTL.DF` and `CGU_CTL.MSEL` fields.

a. To change the PLL frequency while the cores are idle, write to the `CGU_CTL` register with the `CGU_CTL.WFI` bit =1.

b. To change the PLL frequency while the cores are active, write to the `CGU_CTL` register with the `CGU_CTL.WFI` bit =0.

This sequence performs these actions:

1. Updates the corresponding CGU registers
2. Bypasses the PLL
3. Makes the PLL lock to the new values in the `CGU_CTL.MSEL` or `CGU_CTL.DF` fields
4. Changes the clock frequencies
5. Exits the PLL bypass with all clocks aligned

When exiting the PLL bypass state, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.PLLEN` bit =1, the `CGU_STAT.PLOCK` bit =1, the `CGU_STAT.PLLBP` bit =0, and the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.PLOCK` bit, `CGU_STAT.PLLBP` bit, and `CGU_STAT.CLKSALGN` bit to discover when the PLL is locked and the clocks are aligned.

Changing the frequency of the PLL is allowed while the PLL is bypassed. But, the new PLLCLK frequency is not used until the PLL is no longer bypassed.

Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency

To change the clock frequencies, write new values to `CGU_DIV.CSEL` or `CGU_DIV.SYSSEL` bits. The frequency change occurs only when the PLL is not bypassed. Any time the *CCLKn* or *SYSCLK* clock frequencies are changed, they exit the frequency change sequence aligned.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired `CGU_DIV.CSEL`, `CGU_DIV.SYSSEL`, and `CGU_DIV.OSEL` bit field values with the `CGU_DIV.UPDT` bit = 1.

   ADDITIONAL INFORMATION: This write updates the `CGU_DIV` register, changes the *SCLKn* and *SYSCLK* frequencies, and aligns the clocks. When the clocks are aligned, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write attempt to change the `CGU_DIV.S0SEL` or `CGU_DIV.S1SEL` bit fields while `CGU_STAT.CLKSALGN` bit =1 (clocks alignment in progress) triggers an MMR access bus error and the `CGU_DIV` register is not modified.

Programming the *SYSCLK* frequency to a higher value than *CCLKn* also triggers an MMR access bus error and the `CGU_DIV` register is not modified.

Writing to the `CGU_DIV` register is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Accessing the DDR memory while changing the *SYSCLK* frequency is not supported and can have unpredictable results.

Changing the OCLK Frequency

To change the *OCLK* clock frequency, write a new `CGU_DIV.OSEL` bit value. Any time the *OCLK* clock frequency is changed, the *OCLK*, *CCLKn*, *SYSCLK*, and *SCLKn* clocks exit the frequency change sequence aligned.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired `CGU_DIV.OSEL` value with the `CGU_DIV.UPDT` bit =1.

   ADDITIONAL INFORMATION: This write updates the `CGU_DIV` register, changes the *OCLK* frequency, and aligns all clocks except *OCLK*.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write attempt to change the `CGU_DIV.DSEL` field while the `CGU_STAT.CLKSALGN` bit =1 (clock alignment in progress) triggers an MMR access bus error and the `CGU_DIV` register is not modified. When the clocks are aligned, a CGU event occurs.

Writing to the `CGU_DIV.OSEL` bit field is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Aligning All Clocks

To align the clocks, write 1 to the `CGU_DIV.ALGN` bit. The frequency can be changed, if necessary. The clocks aligned include:

- *CCLKn*
- *SYSCLK*
- *SCLKn*
- *DCLK*
- *OCLK*

1. Read the `CGU_STAT` register to verify that `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write 1 to the `CGU_DIV.ALGN` bit. All other fields can change.

   ADDITIONAL INFORMATION: This write does not alter the `CGU_DIV` register unless one of the clock-select fields is modified. When the clocks are aligned, a CGU event occurs.

The `CGU_STAT` register exits this sequence with the `CGU_STAT.CLKSALGN` bit =0. Poll the `CGU_STAT.CLKSALGN` bit to discover when the clocks are aligned. Any write to the `CGU_DIV` register intended to align clocks or to change a clock select field while the `CGU_STAT.CLKSALGN` bit =1 (clocks alignment in progress) triggers an MMR access bus error. And, the `CGU_DIV` register is not modified.

Writing 1 to the `CGU_DIV.ALGN` bit has no effect while the processor is in active (PLL bypassed) mode.

The CGU does not support accessing the DDR memory while changing the *SYSCLK* or *DCLK* frequencies. This type of access can have unpredictable results.

Shutting Off CCLKn From Another Master

CCLKn can be shut off to save power when it is not in use.

1. Disable interrupts to core n.
2. Set the `RCU_SIDIS.SI[n]` bit to disable the interfaces of core n in order to:
    a. Stop DMA accesses to its L1.
    b. Stop accesses to memory to core 0.
    c. Stop accesses to MMRs.
3. Test the `RCU_SISTAT.SI[n]` bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
4. Set the `CGU_CCBF_DIS.CCBF0` bit to disable the *CCLKn* buffer.
5. Check the `CGU_CCBF_STAT.CCBF0` bit.

If the `CGU_CCBF_STAT.CCBF0` bit is set, continue.

Reenable CCLKn From Another Master

1. Clear the `CGU_CCBF_DIS.CCBF0` to enable *CCLKn*.
2. Check the `CGU_CCBF_STAT.CCBF0` bit.
    a. If the `CGU_CCBF_DIS.CCBF0` bit is cleared, continue.
3. Clear the `RCU_SIDIS.SI[n]` bit. The core deasserts its acknowledge signal in response to the `RCU_SYSRST0`signal. This operation clears the `RCU_SISTAT.SI[n]` bit.

Valid Clock Multiplier Settings

Processor operations depend on valid settings in the `CGU_CTL` and `CGU_DIV` registers. These registers control the clock multiplier and divisor values. Set these registers such that the minimum and maximum clocks specified in the data sheet are not violated. All other clock specifications in the data sheet must also be adhered to for correct operation of the processor.

NOTE: The frequency of any processor core clock to the *SYSCLK* is either 1:1 or 2:1 only.

PLL Bypass and PLL Disable

Writing 1 to the `CGU_PLLCTL.PLLBPST` bit tells the PLL to apply OSC_CLKIN clock to *CCLK*, *SYSCLK*, *SCLK0*, *SCLK1*, *DCLK* (PLL Bypass), and *OCLK* outputs. Writing 1 to the `CGU_PLLCTL.PLLBPCL` bit tells the PLL to exit its PLL Bypass state and make all output clocks align and transition to their programmed frequencies.

The PLL can be disabled by clearing `CGU_PLLCTL.PLLEN` bit while in the bypass state. If necessary, clock buffers can be disabled. *CCLK0* and *CCLK1* clocks can be disabled or enabled by writing 1 or 0 to the corresponding bit in the `CGU_CCBF_DIS` register. To know which Core clock buffers are enabled or disabled, software reads the `CGU_CCBF_STAT` register. To know which Core clock buffers were disabled since the last read, software reads the `CGU_CCBF_STAT` register. The *SCLK0*, *SCLK1*, *DCLK* and *OCLK* clocks can be disabled or enabled by writing 1 or 0 to the corresponding bit in the `CGU_SCBF_DIS` register. Software cannot disable *SYSCLK*.

ADSP-SC5xx Specific Information

The processor has two system crystal oscillators and two system CGU units to provide the clocks to the system. Both of the CGUs come up in bypass mode out of reset.

CGU0 is the main CGU which provides the *SYSCLK (SYSCLK_0)*, *SCLK0 (SCLK0_0)*, and *SCLK1 (SCKL1_0)* to the system buses, infrastructure, and most of the peripherals. The rest of the clock outputs from the two CGUs can be routed to a specific peripheral and the cores in the system. For more details, refer to the Clock Distribution Unit (CDU) chapter.

NOTE: Frequency ratios of Core clock to *SYSCLK* are either 2:1 or 1:1.

The processor supports no other frequency ratios. Program the divider/CDU values for the *CLKO0*, *CLKO1*, and *CLKO2* carefully in relation to the divider programming for the *SYSCLK*.

ADSP-SC58x CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 3-8: ADSP-SC58x CGU Register List

Name	Description
`CGU_CCBF_DIS`	Core Clock Buffer Disable Register
`CGU_CCBF_STAT`	Core Clock Buffer Status Register
`CGU_CLKOUTSEL`	CLKOUT Select Register
`CGU_CTL`	Control Register
`CGU_DIV`	Clocks Divisor Register
`CGU_OSCWDCTL`	Oscillator Watchdog Register
`CGU_PLLCTL`	PLL Control Register

Table 3-8: ADSP-SC58x CGU Register List (Continued)

Name	Description
CGU_REVID	Revision ID Register
CGU_SCBF_DIS	System Clock Buffer Disable Register
CGU_SCBF_STAT	System Clock Buffer Status Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Time Stamp Control Register
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

Core Clock Buffer Disable Register

The `CGU_CCBF_DIS` register controls each core's clock buffer to determine if the CCLK is enabled.

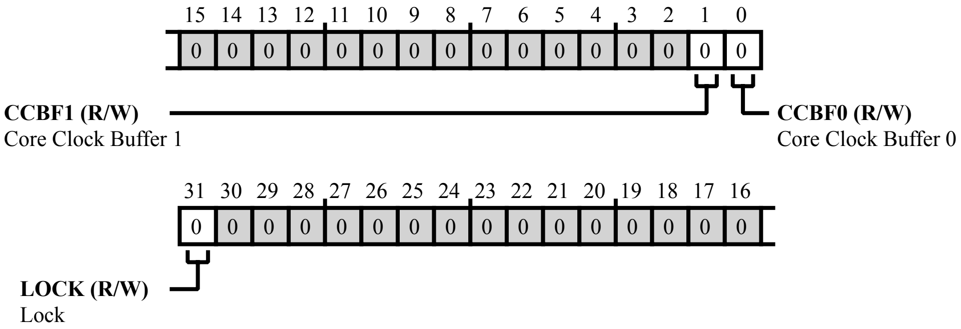

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CCBF1 (R/W)
Core Clock Buffer 1
CCBF0 (R/W)
Core Clock Buffer 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 3-3: CGU_CCBF_DIS Register Diagram

Table 3-9: CGU_CCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If set (=1) the `CGU_CCBF_DIS.LOCK` bit locks the `CGU_CCBF_DIS` register.	
		0	Unlock register
		1	Lock register
1 (R/W)	CCBF1	Core Clock Buffer 1. The `CGU_CCBF_DIS.CCBF1` bit enables (=0) or disables (=1) CCLK1s buffer.	
0 (R/W)	CCBF0	Core Clock Buffer 0. The `CGU_CCBF_DIS.CCBF0` bit enables (=0) or disables (=1) CCLK0s buffer.	
		0	Enable buffer
		1	Disable buffer

Core Clock Buffer Status Register

The `CGU_CCBF_STAT` register shows which core clock buffer(s) are disabled. For example clearing the `CGU_CCBF_DIS.CCBF0` bit clears the `CGU_CCBF_STAT.CCBF0` bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

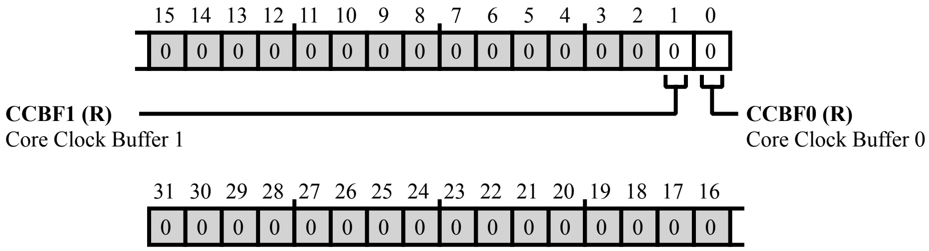

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CCBF1 (R)
Core Clock Buffer 1
CCBF0 (R)
Core Clock Buffer 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 3-4: CGU_CCBF_STAT Register Diagram

Table 3-10: CGU_CCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	CCBF1	Core Clock Buffer 1. The `CGU_CCBF_STAT.CCBF1` bit reports the status of the `CGU_CCBF_DIS.CCBF1` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled
0 (R/NW)	CCBF0	Core Clock Buffer 0. The `CGU_CCBF_STAT.CCBF0` bit reports the status of the `CGU_CCBF_DIS.CCBF0` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

CLKOUT Select Register

The `CGU_CLKOUTSEL` selects the signal that the CGU drives through the CLKOUT multiplexer. Also, this register selects the divisor for the USBCLK output.

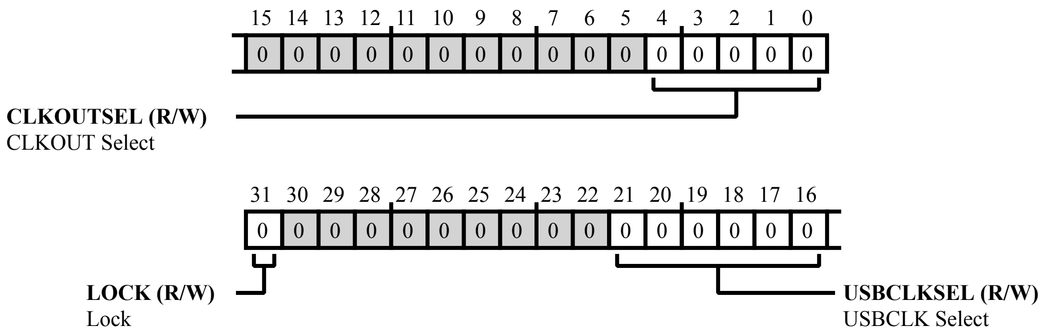

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKOUTSEL (R/W)
CLKOUT Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
USBCLKSEL (R/W)
USBCLK Select

Figure 3-5: CGU_CLKOUTSEL Register Diagram

Table 3-11: CGU_CLKOUTSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `CGU_CLKOUTSEL.LOCK` bit is set, the `CGU_CLKOUTSEL` register is read only (locked).	
		0	Unlock
		1	Lock
21:16 (R/W)	USBCLKSEL	USBCLK Select. The `CGU_CLKOUTSEL.USBCLKSEL` selects the divisor in the USBCLK equation: USBCLK frequency = (USB PLL frequency) / (`CGU_CLKOUTSEL.USBCLKSEL` +1) Where the value of `CGU_CLKOUTSEL.USBCLKSEL` is between 0 and 63.	
		0	USBCLKSEL = 0
		63	USBCLKSEL = 63
4:0 (R/W)	CLKOUTSEL	CLKOUT Select. The `CGU_CLKOUTSEL.CLKOUTSEL` selects the signal that the CGU drives through the CLKOUT pin multiplexer.	
		0	CLKIN0
		1	CLKIN1
		2	CGU_0.SYSCLK
		3	CLKO0

Table 3-11: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	CLKO2
		5	CLKO3
		6	CLKO5
		7	CLKO7
		8	CLKO8
		9	Reserved
		10	Reserved
		11	Reserved
		12	Reserved
		13	Reserved
		14	Reserved
		15	Reserved
		16	Reserved
		17	Reserved
		18	Reserved
		19-31	Reserved

Control Register

The CGU_CTL controls the clock generation divisors for SYS_CLKIN and the PLL. Read after write accesses to the CGU_CTL register returns the new value even if the clock's frequency change is still in progress.

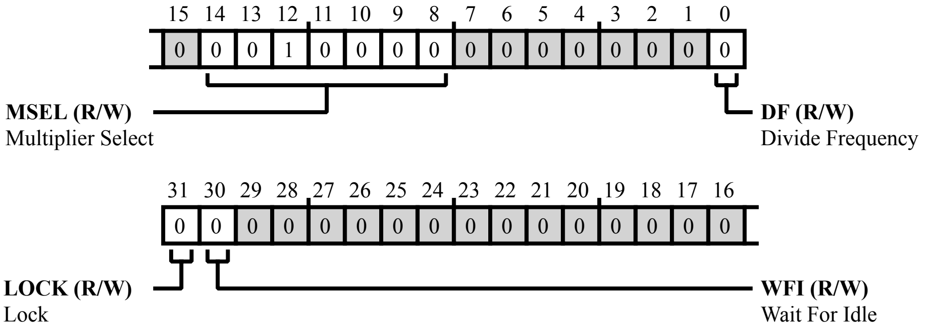

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
MSEL (R/W)
Multiplier Select
DF (R/W)
Divide Frequency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
WFI (R/W)
Wait For Idle

Figure 3-6: CGU_CTL Register Diagram

Table 3-12: CGU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `CGU_CTL.LOCK` bit is set, the `CGU_CTL` register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	WFI	Wait For Idle. Modifying the PLL multiplier requires the PLL to re-lock and once the PLL locks, clocks have to be synchronized. Changes to the `CGU_CTL.MSEL` and the `CGU_CTL.DF` bit values results in bypassing the PLL. The `CGU_CTL.WFI` bit forces the PLL to wait for all processor cores to be in an idle or reset state before changing frequencies as a result of changes to the `CGU_CTL.MSEL` or `CGU_CTL.DF` bits. Write accesses to the `CGU_CTL` to change the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit values while the PLL is locking sets the `CGU_STAT.WDFMSERR` bit.	
		0	Update Immediately
		1	Wait for Idle

Table 3-12: CGU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:8 (R/W)	MSEL	Multiplier Select. The `CGU_CTL.MSEL` bit field selects the multiplier in the PLLCLK equation: PLLCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL Where the value of MSEL is between 1 and 127.	
		0	MSEL = 128
		1-127	MSEL = 1 to 127
0 (R/W)	DF	Divide Frequency. The `CGU_CTL.DF` bit selects whether or not the CLKIN input is divided by two before being passed to the PLL.	
		0	Pass OSC_CLKIN to PLL
		1	Pass OSC_CLKIN/2 to PLL

Clocks Divisor Register

The CGU_DIV register controls clock divisors for core clocks, system clocks, external (off core) memory clocks, and output clock. Read after write accesses to the CGU_DIV register returns the new value even if the clock's frequency change is still in progress.

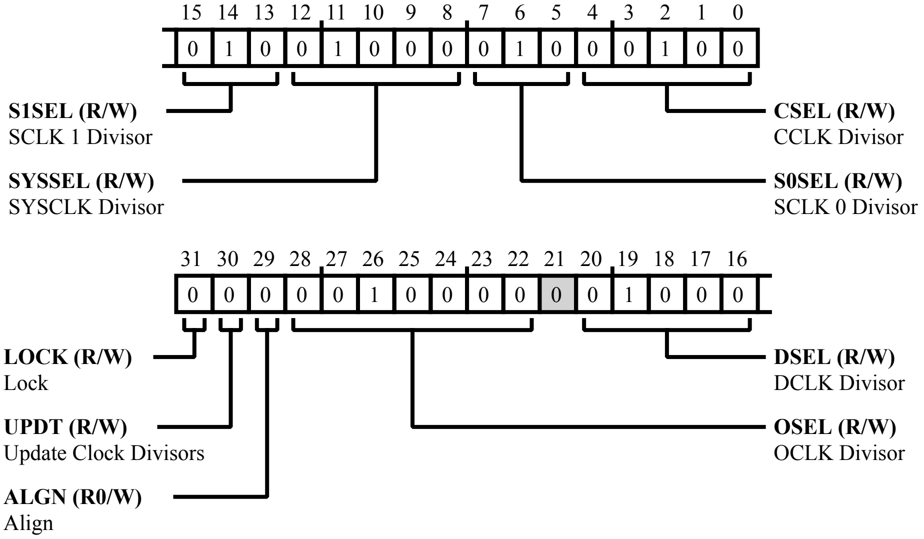

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0
S1SEL (R/W)
SCLK 1 Divisor
SYSSEL (R/W)
SYSCLK Divisor
CSEL (R/W)
CCLK Divisor
S0SEL (R/W)
SCLK 0 Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
LOCK (R/W)
Lock
UPDT (R/W)
Update Clock Divisors
ALGN (R0/W)
Align
DSEL (R/W)
DCLK Divisor
OSEL (R/W)
OCLK Divisor

Figure 3-7: CGU_DIV Register Diagram

Table 3-13: CGU_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_DIV.LOCK bit is set, the CGU_DIV register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	UPDT	Update Clock Divisors. The CGU_DIV.UPDT controls whether the CGU drives new CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and CGU_DIV.OSEL values to PLL after CGU_DIV register update.	
		0	No PLL Update
		1	Drive Updated SEL Values to PLL

Table 3-13: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R0/W)	ALGN	Align. The `CGU_DIV.ALGN` directs the CGU to align the PLL-based clocks. The divisor selections (`CGU_DIV.CSEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL`, and/or `CGU_DIV.OSEL`) do not have to change.	
		0	No Action
		1	Align PLL Clocks
28:22 (R/W)	OSEL	OCLK Divisor. The `CGU_DIV.OSEL` selects the divisor in the OCLK equation: OCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / `CGU_DIV.OSEL` Where the value of `CGU_DIV.OSEL` is between 1 and 127.	
		0	OSEL = 128
		1-127	OSEL = 1 to 127
20:16 (R/W)	DSEL	DCLK Divisor. The `CGU_DIV.DSEL` selects the divisor in the DCLK equation: DCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/`CGU_DIV.DSEL` Where the value of `CGU_DIV.DSEL` is between 1 and 31.	
		0	DSEL = 32
		1-31	DSEL = 1 to 31
15:13 (R/W)	S1SEL	SCLK 1 Divisor. The `CGU_DIV.S1SEL` selects the divisor in the SCLK1 equation: SCLK1 frequency = (SYSCLK frequency) / `CGU_DIV.S1SEL` Where the value of `CGU_DIV.S1SEL` is between 1 and 7.	
		0	S1SEL = 8
		1-7	S1SEL = 1 to 7
12:8 (R/W)	SYSSEL	SYSCLK Divisor. The `CGU_DIV.SYSSEL` selects the divisor in the SYSCLK equation: SYSCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/`CGU_DIV.SYSSEL` Where the value of `CGU_DIV.SYSSEL` is between 1 and 31.	
		0	SYSSEL = 32
		1-31	SYSSEL = 1 to 31

Table 3-13: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:5 (R/W)	S0SEL	SCLK 0 Divisor. The `CGU_DIV.S0SEL` selects the divisor in the SCLK0 equation: SCLK0 frequency = (SYSCLK frequency) / `CGU_DIV.S0SEL` Where the value of `CGU_DIV.S0SEL` is between 1 and 7.	
		0	S0SEL = 8
		1-7	S0SEL = 1 to 7
4:0 (R/W)	CSEL	CCLK Divisor. The `CGU_DIV.CSEL` selects the divisor in the CCLK equation: CCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / `CGU_DIV.CSEL` Where the value of `CGU_DIV.CSEL` is between 1 and 31.	
		0	CSEL = 32
		1-31	CSEL= 1 to 31

Oscillator Watchdog Register

The CGU_OSCWDCTL register configures the CGU to allow the detection of the absence of input clock transitions and provides a fault warning via the SYS_FAULT pin. The CGU_OSCWDCTL register also detects and reports input oscillator frequencies above and below specified limits, in order to specifically detect harmonic or sub-harmonic crystal oscillator behavior. This detection is achieved by using an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters.

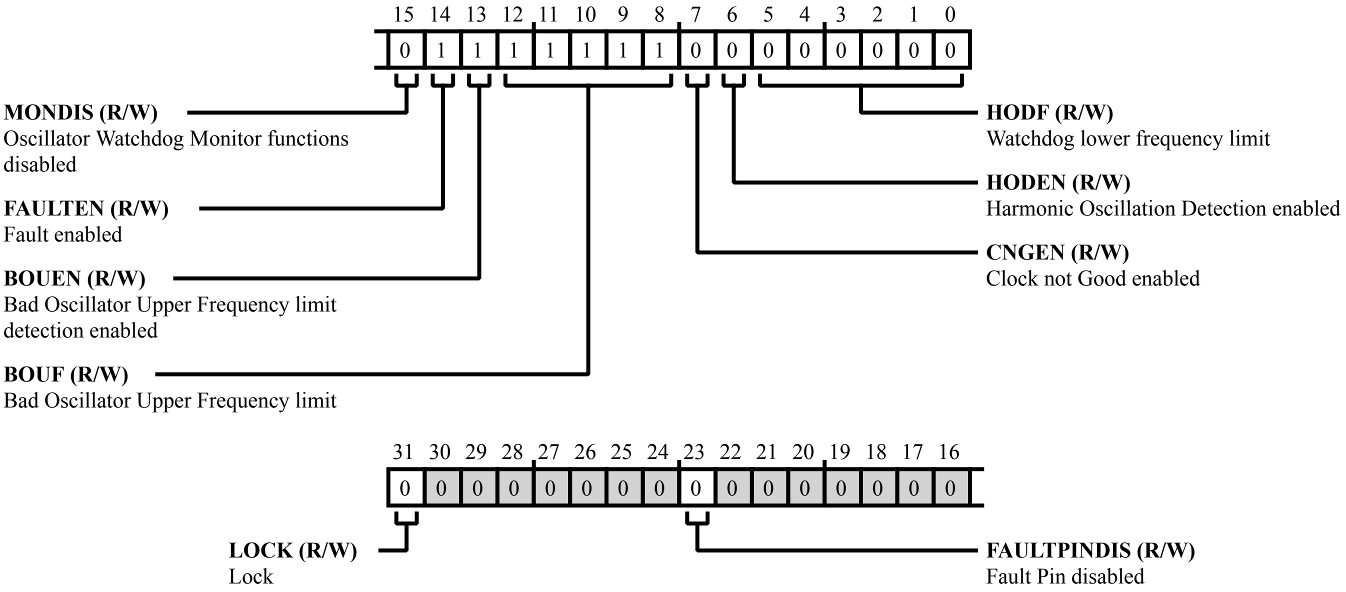

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
MONDIS (R/W)
Oscillator Watchdog Monitor functions disabled
FAULTEN (R/W)
Fault enabled
BOUEN (R/W)
Bad Oscillator Upper Frequency limit detection enabled
BOUF (R/W)
Bad Oscillator Upper Frequency limit
HODF (R/W)
Watchdog lower frequency limit
HODEN (R/W)
Harmonic Oscillation Detection enabled
CNGEN (R/W)
Clock not Good enabled
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
FAULTPINDIS (R/W)
Fault Pin disabled

Figure 3-8: CGU_OSCWDCTL Register Diagram

Table 3-14: CGU_OSCWDCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock.
23 (R/W)	FAULTPINDIS	Fault Pin disabled. The CGU_OSCWDCTL.FAULTPINDIS bit disables pin fault detection.
15 (R/W)	MONDIS	Oscillator Watchdog Monitor functions disabled. The CGU_OSCWDCTL.MONDIS bit disables all the input clock monitor and fault detection functions.
14 (R/W)	FAULTEN	Fault enabled. The CGU_OSCWDCTL.FAULTEN bit enables fault detection.
13 (R/W)	BOUEN	Bad Oscillator Upper Frequency limit detection enabled. The CGU_OSCWDCTL.BOUEN bit enables upper limit bad oscillation detection.

Table 3-14: CGU_OSCWDCTL Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">12:8
(R/W)</td><td rowspan="3">BOUF</td><td colspan="2">Bad Oscillator Upper Frequency limit.
The `CGU_OSCWDCTL.BOUF` bits indicate the desired upper fail limit for the bad oscillation detection.</td></tr>
<tr><td>0</td><td>Enable buffer</td></tr>
<tr><td>1</td><td>Disable buffer</td></tr>
<tr><td>7
(R/W)</td><td>CNGEN</td><td colspan="2">Clock not Good enabled.
The `CGU_OSCWDCTL.CNGEN` bit enables the detection of an oscillator watchdog clock fault.</td></tr>
<tr><td>6
(R/W)</td><td>HODEN</td><td colspan="2">Harmonic Oscillation Detection enabled.
The `CGU_OSCWDCTL.HODEN` bit enables harmonic oscillation detection.</td></tr>
<tr><td>5:0
(R/W)</td><td>HODF</td><td colspan="2">Watchdog lower frequency limit.
The `CGU_OSCWDCTL.HODF` bit field is used to indicate the desired lower fail limit for the harmonic oscillation detection in MHz.</td></tr>
</table>

PLL Control Register

The CGU_PLLCTL register contains bits that enable and disable the PLL as well as control its function.

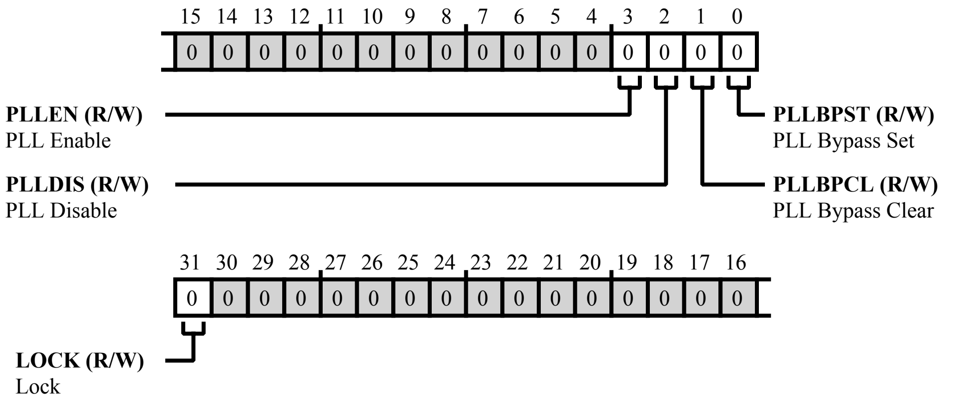

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PLLEN (R/W)
PLL Enable
PLLDIS (R/W)
PLL Disable
PLLBPST (R/W)
PLL Bypass Set
PLLBPCL (R/W)
PLL Bypass Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 3-9: CGU_PLLCTL Register Diagram

Table 3-15: CGU_PLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. Setting (=1) the CGU_PLLCTL.LOCK bit locks access to the CGU_PLLCTL register.	
		0	Unlock register
		1	Lock register
3 (R/W)	PLLEN	PLL Enable. Setting (=1) the CGU_PLLCTL.PLLEN bit enables the PLL.	
		0	No action
		1	Enable PLL
2 (R/W)	PLLDIS	PLL Disable. Setting (=1) the CGU_PLLCTL.PLLDIS bit disables the PLL.	
		0	No action
		1	Disable PLL
1 (R/W)	PLLBPCL	PLL Bypass Clear. Setting (=1) the CGU_PLLCTL.PLLBPCL bit takes the PLL out of bypass mode.	
		0	No action
		1	Exit bypass mode

Table 3-15: CGU_PLLCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PLLBPST	PLL Bypass Set. Setting (=1) the `CGU_PLLCTL.PLLBPST` bit bypasses the PLL and all the clocks run on CLKIN.	
		0	Use PLL
		1	Bypass PLL

Revision ID Register

The `CGU_REVID` register reports the version of the CGU.

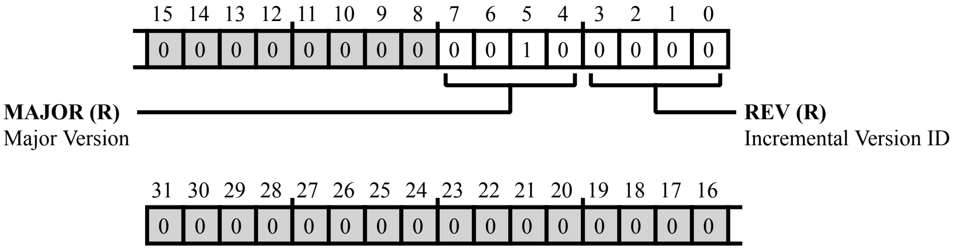

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
MAJOR (R)
Major Version
REV (R)
Incremental Version ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 3-10: CGU_REVID Register Diagram

Table 3-16: CGU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version.
3:0 (R/NW)	REV	Incremental Version ID.

System Clock Buffer Disable Register

The `CGU_SCBF_DIS` register controls each system's clock buffer to determine if the SCLKn buffer is enabled.

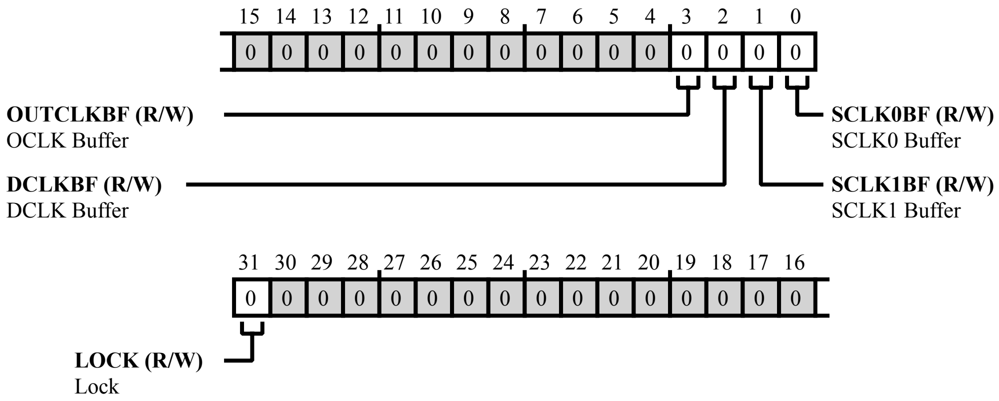

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OUTCLKBF (R/W)
OCLK Buffer
DCLKBF (R/W)
DCLK Buffer
SCLK0BF (R/W)
SCLK0 Buffer
SCLK1BF (R/W)
SCLK1 Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 3-11: CGU_SCBF_DIS Register Diagram

Table 3-17: CGU_SCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. The `CGU_SCBF_DIS.LOCK` bit allows writes to the `CGU_SCBF_DIS` register when cleared (=0) or blocks writes if set (=1) and the `SPU_CTL.GLCK` bit is set.	
		0	Unlock register
		1	Lock register
3 (R/W)	OUTCLKBF	OCLK Buffer. The `CGU_SCBF_DIS.OUTCLKBF` bit enables (=0, default) or disables (=1) OCLKs buffer.	
		0	Enable buffer
		1	Disable buffer
2 (R/W)	DCLKBF	DCLK Buffer. The `CGU_SCBF_DIS.DCLKBF` bit enables (=0, default) or disables (=1) DCLKs buffer.	
		0	Enable buffer
		1	Disable buffer

Table 3-17: CGU_SCBF_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	SCLK1BF	SCLK1 Buffer. The `CGU_SCBF_DIS.SCLK1BF` bit enables (=0, default) or disables (=1) SCLK1s buffer.	
		0	Enable buffer
		1	Disable buffer
0 (R/W)	SCLK0BF	SCLK0 Buffer. The `CGU_SCBF_DIS.SCLK0BF` bit enables (=0, default) or disables (=1) SCLK0s buffer.	
		0	Enable buffer
		1	Disable buffer

System Clock Buffer Status Register

The `CGU_SCBF_STAT` register shows which system clock buffer(s) are disabled. For example clearing the `CGU_CCBF_DIS.CCBF0` bit clears the `CGU_SCBF_STAT.SCLK0BF` bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

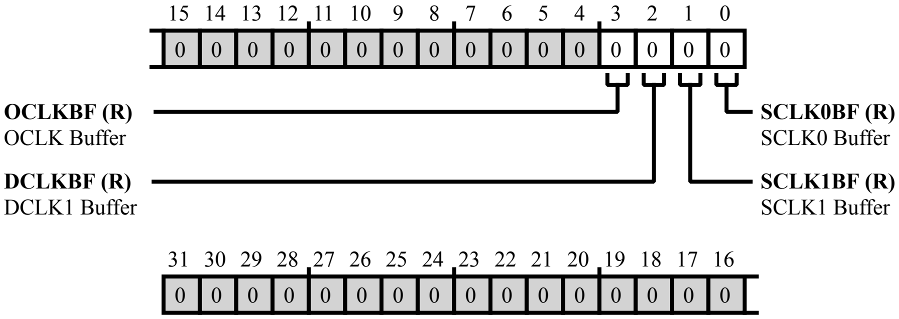

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OCLKBF (R)
OCLK Buffer
DCLKBF (R)
DCLK1 Buffer
SCLK0BF (R)
SCLK0 Buffer
SCLK1BF (R)
SCLK1 Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 3-12: CGU_SCBF_STAT Register Diagram

Table 3-18: CGU_SCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	OCLKBF	OCLK Buffer. The `CGU_SCBF_STAT.OCLKBF` bit reports the status of the `CGU_SCBF_DIS.OUTCLKBF` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled
2 (R/NW)	DCLKBF	DCLK1 Buffer. The `CGU_SCBF_STAT.DCLKBF` bit reports the status of the `CGU_SCBF_DIS.DCLKBF` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled
1 (R/NW)	SCLK1BF	SCLK1 Buffer. The `CGU_SCBF_STAT.SCLK1BF` bit reports the status of the `CGU_SCBF_DIS.SCLK1BF` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

Table 3-18: CGU_SCBF_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SCLK0BF	SCLK0 Buffer. The `CGU_SCBF_STAT.SCLK0BF` bit reports the status of the `CGU_SCBF_DIS.SCLK0BF` bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

Status Register

The `CGU_STAT` register reflects the PLL status and errors detected during the PLL configuration. This register may be cleared asynchronously by a reset signal from the RCU module. All bits---except those defined as W1C (write-1-to-clear)---are read only.

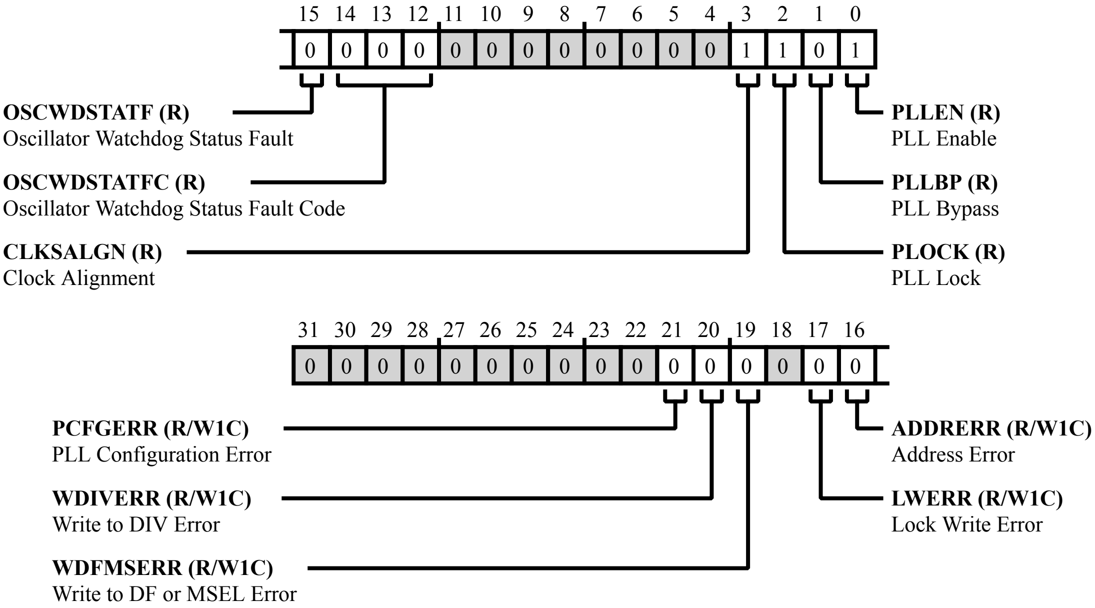

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1
OSCWDSTATF (R)
Oscillator Watchdog Status Fault
OSCWDSTATFC (R)
Oscillator Watchdog Status Fault Code
CLKSALGN (R)
Clock Alignment
PLLEN (R)
PLL Enable
PLLBP (R)
PLL Bypass
PLOCK (R)
PLL Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PCFGERR (R/W1C)
PLL Configuration Error
WDIVERR (R/W1C)
Write to DIV Error
WDFMSERR (R/W1C)
Write to DF or MSEL Error
ADDRERR (R/W1C)
Address Error
LWERR (R/W1C)
Lock Write Error

Figure 3-13: CGU_STAT Register Diagram

Table 3-19: CGU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PCFGERR	PLL Configuration Error. If the `CGU_PLLCTL.PLLBPST` and the `CGU_PLLCTL.PLLBPCL` bits are set (=1) simultaneously or the `CGU_PLLCTL.PLLDIS` bit was set (=1) in full-on mode or while trying to enter full-on mode (`CGU_PLLCTL.PLLBPCL` =1), the `CGU_STAT.PCFGERR` bit triggers the bus error.	
		0	No Error
		1	Configuration Error

Table 3-19: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	WDIVERR	Write to DIV Error. The `CGU_STAT.WDIVERR` bit indicates a write access to the `CGU_DIV` register (to trigger an alignment sequence or to change the `CGU_DIV.CSEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.S1SEL`, or `CGU_DIV.DSEL` bit values) while the PLL is locked, but still aligning the clocks. Read after write accesses to the `CGU_STAT` and `CGU_DIV` registers return the new value even if the clock frequency change is still in progress.	
		0	No Error
		1	Write DIV Error
19 (R/W1C)	WDFMSERR	Write to DF or MSEL Error. The `CGU_STAT.WDFMSERR` bit indicates a write access to the `CGU_CTL` register to change the `CGU_CTL.DF` or `CGU_CTL.MSEL` bit values while the PLL is locking.	
		0	No Error
		1	Write DF/MSEL Error
17 (R/W1C)	LWERR	Lock Write Error. The `CGU_STAT.LWERR` bit indicates an attempt to write to write-protected (locked) CGU registers. The CGU issues a bus error for this condition.	
		0	No Error
		1	Lock Write Error
16 (R/W1C)	ADDRERR	Address Error. The `CGU_STAT.ADDRERR` bit indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The CGU issues a bus error for this condition.	
		0	No Error
		1	Address Error
15 (R/NW)	OSCWDSTATF	Oscillator Watchdog Status Fault. The `CGU_STAT.OSCWDSTATF` bit indicates a fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.	
		0	No Fault
		1	Fault
14:12 (R/NW)	OSCWDSTATFC	Oscillator Watchdog Status Fault Code. The `CGU_STAT.OSCWDSTATFC` bit field indicates the nature of the fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.	
		0	No Fault
		1	No Input Clock

Table 3-19: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Subharmonic CLKIN
		3	Harmonic CLKIN
		4	No AUX_CLK
		5	CLKIN > Upper Frequency Limit (BOUF)
		6	Reserved
		7	Multiple Limit Faults
3 (R/NW)	CLKSALGN	Clock Alignment. The `CGU_STAT.CLKSALGN` bit indicates whether a clock alignment sequence is in progress. This bit is set when clocks alignment is required by changes to `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL`, or `CGU_DIV.OSEL`. The `CGU_STAT.CLKSALGN` bit is cleared when clocks are aligned. Note that (after a PLL frequency change in active state) the `CGU_STAT.CLKSALGN` bit may indicate that clocks are not aligned even though the clocks are aligned (all clocks are aligned and running at CLKIN frequency).	
		0	Clocks are Aligned
		1	Clocks not Aligned (alignment in progress)
2 (R/NW)	PLOCK	PLL Lock. The `CGU_STAT.PLOCK` bit indicates whether the PLL is locked. This bit is set when the PLL locks (PLL lock counter end-of-count). The `CGU_STAT.PLOCK` bit is cleared when requested PLL frequency change (for PLL reset, PLL disable-to-enable transition, or a change to the `CGU_CTL.MSEL` or `CGU_CTL.DF` values) is in progress.	
		0	PLL not Locked (PLL frequency change in progress)
		1	PLL Locked
1 (R/NW)	PLLBP	PLL Bypass. The `CGU_STAT.PLLBP` bit indicates whether the PLL is bypassed. The default value for the `CGU_STAT.PLLBP` bit is determined by the bypass strap pin.	
		0	PLL not Bypassed
		1	PLL Bypassed
0 (R/NW)	PLLEN	PLL Enable. The `CGU_STAT.PLLEN` bit indicates whether the PLL is enabled.	
		0	Disabled
		1	Enabled

Time Stamp Counter 32 LSB Register

The `CGU_TSCOUNT0` register address is used to read the CoreSight time stamp counter LSB 32-bit (bits [31:0]) value.

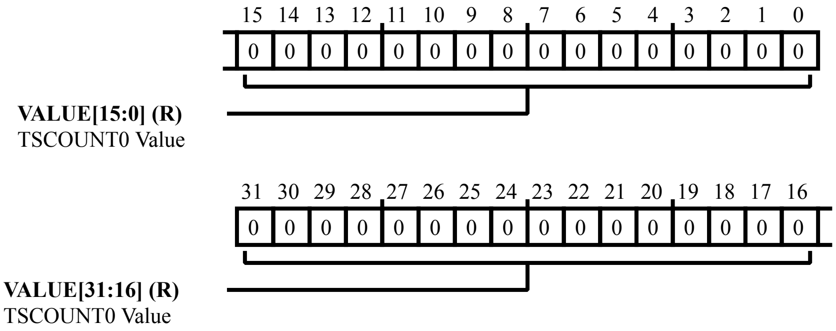

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
TSCOUNT0 Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
TSCOUNT0 Value

Figure 3-14: CGU_TSCOUNT0 Register Diagram

Table 3-20: CGU_TSCOUNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	TSCOUNT0 Value. The `CGU_TSCOUNT0.VALUE` bit field holds the time stamp counter 32 LSBs.

Time Stamp Counter 32 MSB Register

The `CGU_TSCOUNT1` register address is used to read the CoreSight time stamp counter MSB 32-bit (bits [63:32]) value.

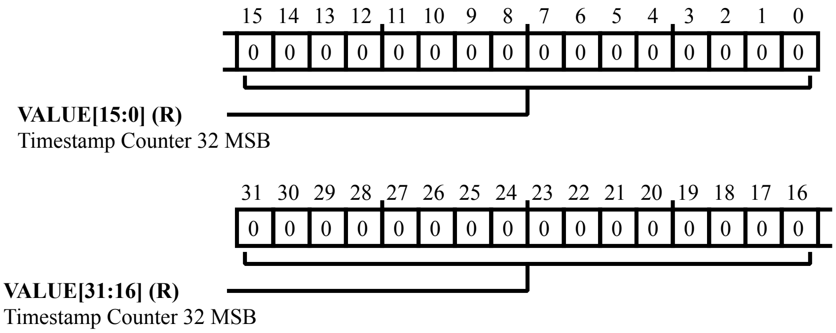

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Timestamp Counter 32 MSB
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Timestamp Counter 32 MSB

Figure 3-15: CGU_TSCOUNT1 Register Diagram

Table 3-21: CGU_TSCOUNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Timestamp Counter 32 MSB. The `CGU_TSCOUNT1.VALUE` bit field holds the time stamp counter 32 MSBs.

Time Stamp Control Register

The `CGU_TSCTL` register controls the operation of the CoreSight time stamp counter.

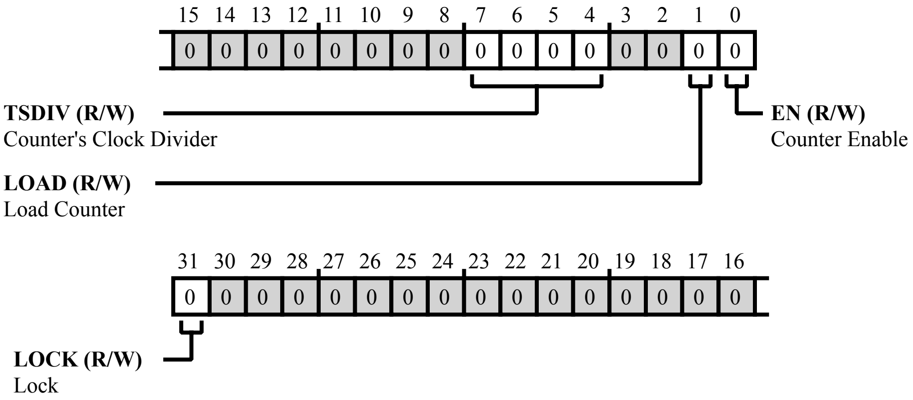

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSDIV (R/W)
Counter's Clock Divider
LOAD (R/W)
Load Counter
EN (R/W)
Counter Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 3-16: CGU_TSCTL Register Diagram

Table 3-22: CGU_TSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. Setting the `CGU_TSCTL.LOCK` bit locks this register.	
		0	Unlock
		1	Lock
7:4 (R/W)	TSDIV	Counter's Clock Divider. The `CGU_TSCTL.TSDIV` bit field divides SYSCLK by 2^{TSDIV}.	
		0-15	Divides SYSCLK by 2^{TSDIV}
1 (R/W)	LOAD	Load Counter. Writing one to the `CGU_TSCTL.LOAD` bit causes CoreSight time stamp counter to be loaded from the `CGU_TSVALUE0` and `CGU_TSVALUE1` registers.	
		0	Always read as "0"
0 (R/W)	EN	Counter Enable. The `CGU_TSCTL.EN` bit enables or disables the CoreSight time stamp counter.	
		0	Counter Disabled
		1	Counter Enabled

Time Stamp Counter Initial 32 LSB Value Register

The `CGU_TSVALUE0` register holds the least significant bits (bits [31:0]) value that is initially loaded to the CoreSight time stamp counter.

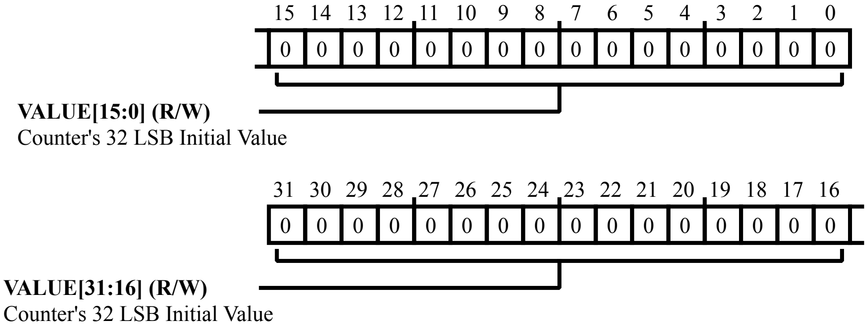

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Counter's 32 LSB Initial Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Counter's 32 LSB Initial Value

Figure 3-17: CGU_TSVALUE0 Register Diagram

Table 3-23: CGU_TSVALUE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's 32 LSB Initial Value. The `CGU_TSVALUE0.VALUE` bit field holds the LSBs value that is initially loaded to the CoreSight time stamp counter.

Time Stamp Counter Initial MSB Value Register

The `CGU_TSVALUE1` register holds the most significant bits (bits [63:32]) value that is initially loaded to the CoreSight time stamp counter.

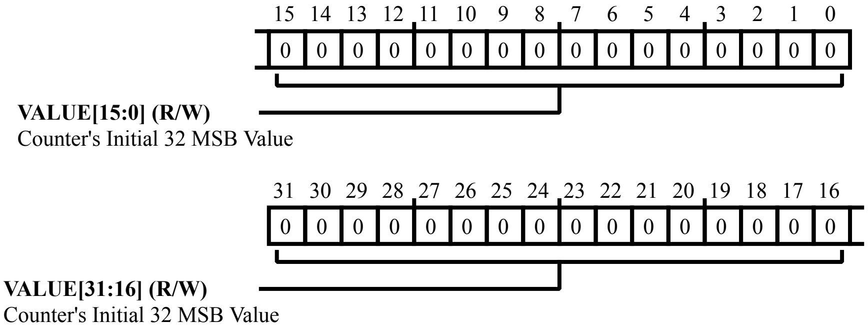

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Counter's Initial 32 MSB Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Counter's Initial 32 MSB Value

Figure 3-18: CGU_TSVALUE1 Register Diagram

Table 3-24: CGU_TSVALUE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's Initial 32 MSB Value. The `CGU_TSVALUE1.VALUE` bit field holds the MSBs value that is initially loaded to the CoreSight time stamp counter.

4 Clock Distribution Unit (CDU)

The Clock Distribution Unit (CDU) consists of an array of multiplexors that select clocks originated from up to four different clock sources. These sources are different clocks that are generated from the CGUs. The multiplexors are configured by software. All clocks multiplexors have up to four different sources. Unused input clocks are grounded internally and never selected. The output clock signal for each multiplexor is assigned to one or more destinations within the processor as shown in CDU Clock Configuration Options.

CDU Features

The CDU modules supports the following features:

- Generation of up to 10 output clocks
- Output clock buffers that can be disabled by software
- Each multiplexor has four input clocks
- Multiplexors that are configured by writing to configuration registers (`CDU_CFG[n]`)
- Clocks originated in CGU0 that are selected by default
- A CDU status register (`CDU_STAT`) that indicates a configuration change is in-progress
- The processor supports 2 independent CGUs: CGU0 is used mandatory, CGU1 is used optional

CDU Functional Description

The CDU functions as a set of software-configurable multiplexors that select clocks from different sources.

CDU Block Diagram

The ***CDU Block Diagram*** figure shows the functional blocks within the CDU. As shown in the figure, the CDU takes different clocks generated by the CGU blocks and provides flexibility to route any clock from possible options as shown in CDU Clock Configuration Options to the output CDU clocks. The output clocks from the CDU are connected to specific targets such as the core, DDR, or CAN module. These targets are clocked though the CDU output clocks rather than being directly clocked by CGU clocks.

This configuration provides the flexibility to meet the specific clock requirements of the different modules in the system (core, DDR, or CAN module) without compromising the clocking of other modules. Such a flexibility is not possible with a single CGU in the system. With two CGUs:

- Each CGU has its own set of clock multipliers and dividers, providing a greater number of orthogonal clocks than possible with a single CGU.
- Each CGU can be clocked from a different CLKIN source, providing additional flexibility.
- CGU0 is always clocked by CLKIN0, whereas CGU1 can be clocked either by CLKIN0 or CLKIN1.

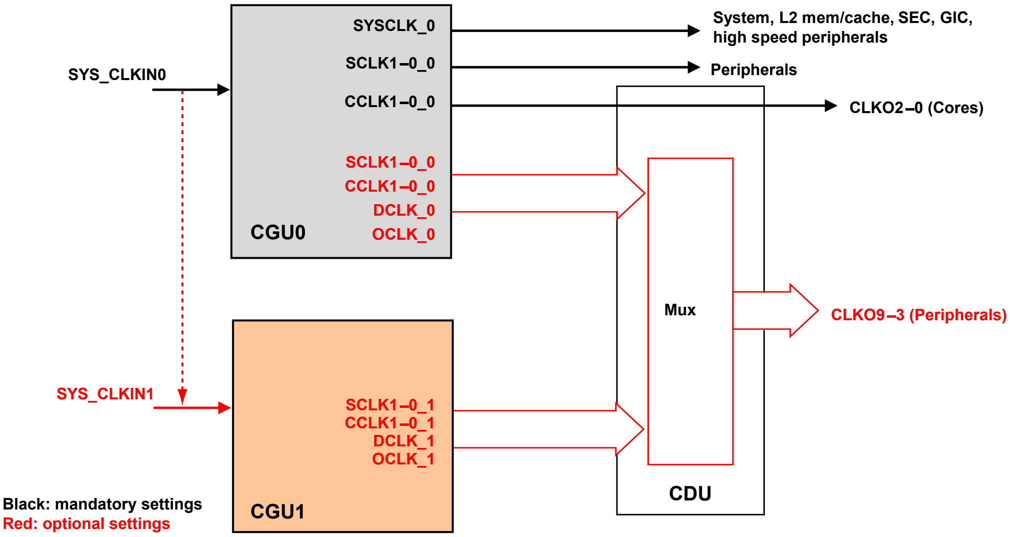

SYSCLK_0
SCLK1–0_0
CCLK1–0_0
SCLK1–0_0
CCLK1–0_0
DCLK_0
OCLK_0
CGU0
SYS_CLKIN0
System, L2 mem/cache, SEC, GIC, high speed peripherals
Peripherals
CLKO2–0 (Cores)
Mux
CLKO9–3 (Peripherals)
SYS_CLKIN1
SCLK1–0_1
CCLK1–0_1
DCLK_1
OCLK_1
CGU1
CDU
Black: mandatory settings
Red: optional settings

Figure 4-1: CGU/CDU Block Diagram

CDU Definitions

The *Clock Descriptions* table provides a brief description of the clocks supported by the processor.

Table 4-1: Clock Descriptions

Clock	Description
CCLK0_0	CCLK0 derived from CGU0
CCLK1_0	CCLK1 derived from CGU0
SYSCLK_0	SYSCLK derived from CGU0
SCLK0_0	SCLK0 derived from CGU0
SCLK1_0	SCLK1 derived from CGU0
DCLK_0	DCLK derived from CGU0
OCLK_0	OCLK derived from CGU0
CCLK0_1	CCLK0 derived from CGU1
CCLK1_1	CCLK1 derived from CGU1

Table 4-1: Clock Descriptions (Continued)

Clock	Description
SCLK0_1	SCLK0 derived from CGU1
SCLK1_1	SCLK1 derived from CGU1
DCLK_1	DCLK derived from CGU1
OCLK_1	OCLK derived from CGU1
CDU_CLKOn	Clocks that come out from the CDU that go to different blocks.

CDU Clock Configuration Options

The ***CDU Targets*** table provides information on clock source and destination options.

Table 4-2: CDU Targets

CDU0 Input				CDU0 Output	Target
CCLK0_0	SYSCLK_0	N/A	N/A	CDU_CLKO0	Core1 (SHARC1)
CCLK0_0	SYSCLK_0	N/A	N/A	CDU_CLKO1	Core2 (SHARC2)
CCLK1_0	SYSCLK_0	N/A	N/A	CDU_CLKO2	Core0 (ARM A5)
DCLK_0	DCLK_1	N/A	N/A	CDU_CLKO3	DDR1/2/3
OCLK_0	OCLK_1	DCLK_1	N/A	CDU_CLKO4	CAN
OCLK_0	OCLK_1	DCLK_1	DCLK_0	CDU_CLKO5	S/PDIF-RX Ref
OCLK_0	CCLK0_1	N/A	N/A	CDU_CLKO6	Reserved
SCLK1_0	SCLK1_1	CCLK0_1	OCLK_0	CDU_CLKO7	GigE/RGMII
SCLK0_0	SCLK0_1	CCLK1_1	DCLK_1	CDU_CLKO8	Link port
OCLK_0/2	CCLK1_1/2	CCLK1_1	DCLK_1	CDU_CLKO9	SDIO

The ***Peripheral Clock Domains*** table shows the clock source assigned to each processor peripheral.

Table 4-3: Peripheral Clock Domains

Peripheral	Clock Source
TIMER, CRC0, CRC1, TWI, UART, PORT, PINT, SMC, WDT, EPWM, CNT, EMAC0, SINC, TMU, HADC, HAE, ACM, MDMA, EMDMA, RTC, CRYPTO ACCELERATOR (SPE), DAI, SPORT, SRC, PCG, S/PDIF Rx, S/PDIF Tx, OTPC	SCLK0_0
QSPI, SPI, EPPI	SCLK1_0
L2_CTL, SEC, TRU, RCU, SPU, SMPU, CDU, DPM, EBMDMA, MAX BW MDMA, FFT, MLB, CRYPTO ACCELERATOR (PKA), ROM, CGU	SYSCLK_0

To further illustrate clocking options, the ***CDU Core Clock Options*** figure shows the conceptual routing of different clock sources to each target. Note that all three cores can be clocked by the clocks originating from CGU0. Select SYS_CLKIN accordingly to get the best match for the core clock frequency.

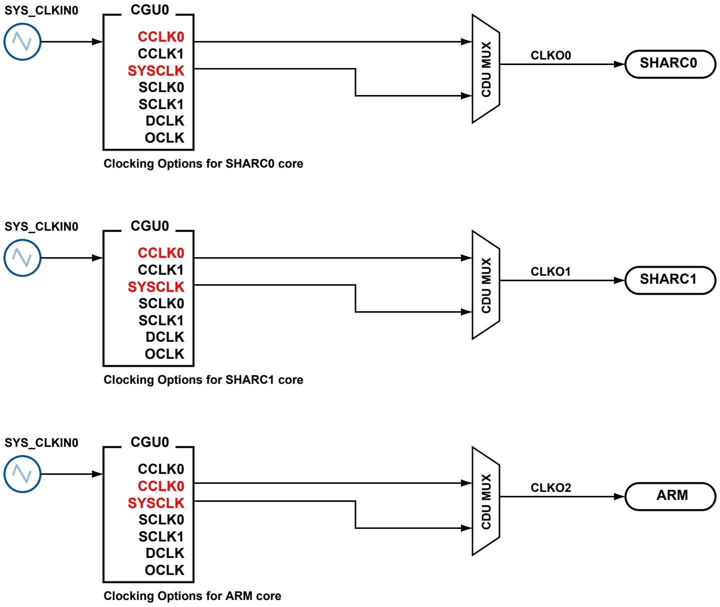

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
CLKO0
SHARC0
Clocking Options for SHARC0 core
SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
CLKO1
SHARC1
Clocking Options for SHARC1 core
SYS_CLKIN0
CGU0
CCLK0
CCLK0
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
CLKO2
ARM
Clocking Options for ARM core

Figure 4-2: CDU Core Clock Options

In the ***CDU Clock Options - DDR*** figure, DDR is clocked from the DCLK from CGU0 or CGU1. This configuration provides the flexibility to program the DDR with frequencies orthogonal to the core clock frequencies.

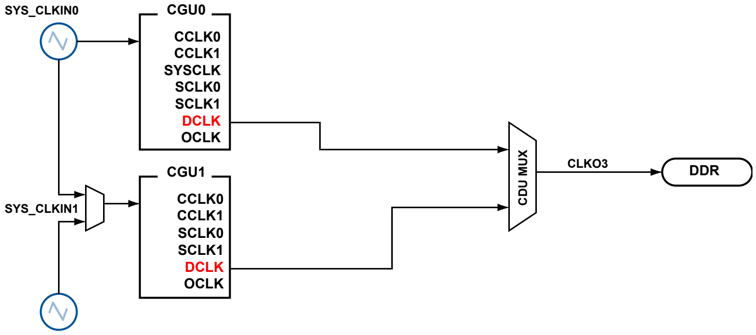

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
CLKO3
DDR

Figure 4-3: CDU Clock Options - DDR

The CAN clock should be a multiple of 500 KHz. Based on this setting, the clock for the CAN module can be derived either from CGU0 or CGU1. It depends on which CLKIN can provide a frequency in multiples of 500 KHz.

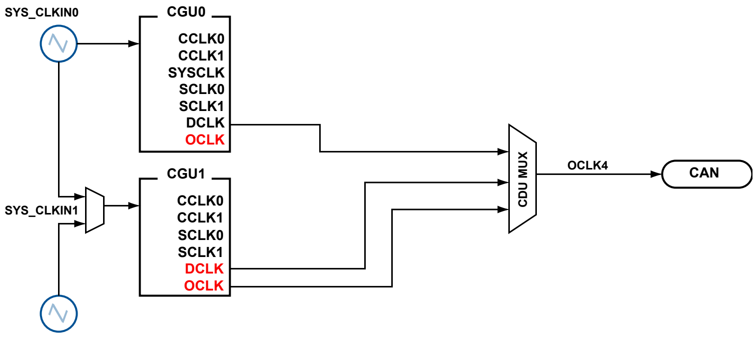

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
OCLK4
CAN

Figure 4-4: CDU Clock Options - CAN

The S/PDIF clock is ideally programmed to operate at 200 MHz (out of the four options).

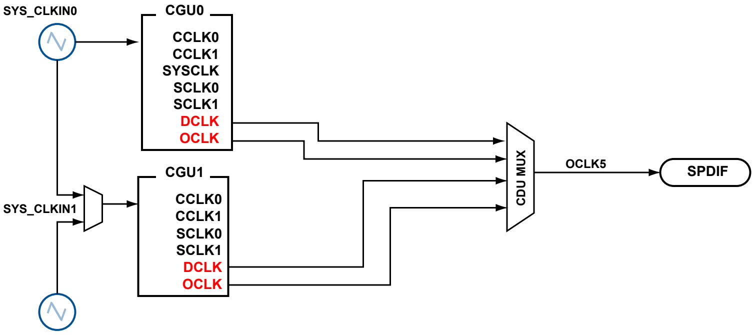

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
OCLK5
SPDIF

Figure 4-5: CDU Clock Options - S/PDIF

The Gigabit Ethernet clock should always be 125 MHz if GigE/RGMII is used. Program the CDU depending on which CGU can provide 125 MHz.

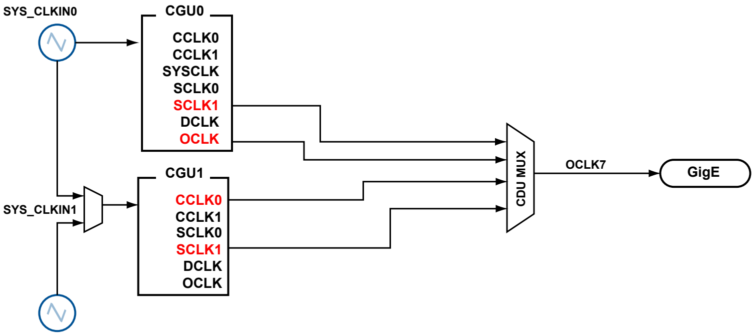

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
OCLK7
GigE

Figure 4-6: CDU Clock Options Gigabit Ethernet

The link port can operate up to a maximum speed of 150 MHz. If the link port needs to operate close to 150 MHz, then the link port clock can be selected from CGU1 to get maximum operation.

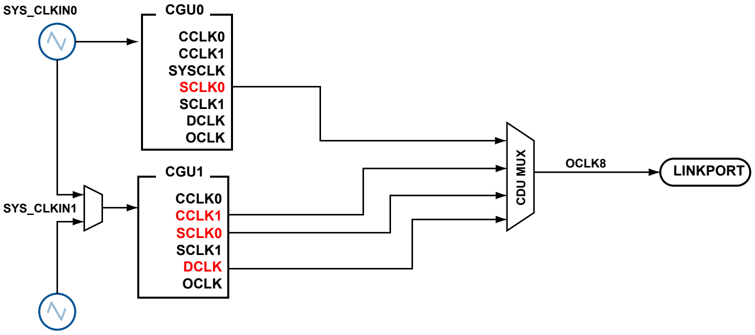

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
SCLK0
SCLK1
DCLK
OCLK
CDU MUX
OCLK8
LINKPORT

Figure 4-7: CDU Clock Options Link Port

The MSI clock can operate at up to 52 MHz for MMC/eMMC operation. The MSI clock can be selected from CGU1 when the OCLK from CGU0 cannot be used.

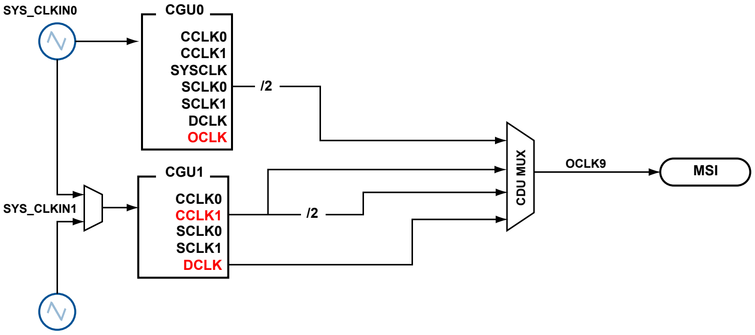

SYS_CLKIN0
CGU0
CCLK0
CCLK1
SYSCLK
SCLK0
/2
SCLK1
DCLK
OCLK
SYS_CLKIN1
CGU1
CCLK0
CCLK1
/2
SCLK0
SCLK1
DCLK
CDU MUX
OCLK9
MSI

Figure 4-8: CDU Clock Options MSI

CDU Programming Model

The `CDU_CFG[n]` registers are a system configuration resource. These registers are accessed by a system configuration routine that also handles the configuration of other system modules. Writes to a `CDU_CFG[n]` register must occur when there is not a CDU_CLKOn configuration change in progress. If a CGUn inputs are selected, that CGUn configuration must be completed first.

Changing the PLL and Clock Frequency

1. Read the `CGU_STAT` register. Verify that:
   - `CGU_STAT.PLLEN` =1 (enabled)
   - `CGU_STAT.PLOCK` =1 (PLL is not locking)
   - `CGU_STAT.CLKSALGN` =0 (clocks aligned)
2. Write the desired values into the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bits with the `CGU_DIV.UPDT` bit cleared (= 0).
3. Write the desired values to the `CGU_CTL.DF` and `CGU_CTL.MSEL` bits.

- To change the PLL frequency while all cores are idle, set the `CGU_CTL.WFI` bit (=1).
- To change the PLL frequency while the cores are active, clear the `CGU_CTL.WFI` bit (=0).

4. Read the `CGU_STAT` register. Verify that:
   - `CGU_STAT.PLLEN` = 1 (enabled)
   - `CGU_STAT.PLOCK` = 1 (not locking)
   - `CGU_STAT.CLKSALGN` = 0 (clocks aligned)
5. If clocks switch from CGUm clocks to CGUn input clocks, read the `CGU_SCBF_STAT` and `CGU_CCBF_STAT` registers. The `CGU_CCBF_STAT`[1:0] bit field corresponds to the CCLK1 and CCLK0 clock buffers, respectively. The `CGU_SCBF_STAT`[3:0] bit field corresponds to the OUTCLK, DCLK, SCLK1, and SCLK0 clock buffers, respectively.
6. Read the `CDU_STAT` register. Verify that the `CDU_STAT.CLKO0` through `CDU_STAT.CLKO9` bits =0 (no CLKOn configuration change in progress).
7. Write to the `CDU_CLKINSEL` register to select the CGU's CLKIN input clock.
8. Write to the `CDU_CFG[n].SEL` bit to select the clock source (the `CDU_CFG[n].EN` bit should =1.)
9. Read the `CDU_CFG[n]` register. Verify that the `CDU_CLKINSEL` has the programmed value.
10. Verify that the `CDU_STAT.CLKO0` through `CDU_STAT.CLKO9` bits =0.

CLKOn is reconfigured.

Changing the Clock Frequency

Use the following procedure to change a clock frequency.

1. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` bit =0 (clocks aligned).
2. Write the desired values into the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bits with the `CGU_DIV.UPDT` bit =1.
3. Read the `CGU_DIV` registers to verify that the `CGU_DIV.CSEL`, `CGU_DIV.S0SEL`, `CGU_DIV.SYSSEL`, `CGU_DIV.S1SEL`, `CGU_DIV.DSEL` and `CGU_DIV.OSEL` bit values are correct.
4. Read the `CGU_STAT` register to verify that the `CGU_STAT.CLKSALGN` =0 (clocks aligned).
5. If clocks switch from CGUm clocks to CGUn input clocks, read the `CGU_SCBF_STAT` and `CGU_CCBF_STAT` registers. The `CGU_CCBF_STAT`[1:0] bit field corresponds to the CCLK1 and CCLK0 clock buffers, respectively. The `CGU_SCBF_STAT`[3:0] bit field corresponds to the OUTCLK, DCLK, SCLK1, and SCLK0 clock buffers, respectively.
6. Read the `CDU_STAT` register. Verify that the `CDU_STAT.CLKO5` through `CDU_STAT.CLKO9` bits =0 (No CLKOn configuration change in progress).

7. Write to the `CDU_CFG[n].SEL` bit to select the clock source (the `CDU_CFG[n].EN` bit should =1).
8. Read the `CDU_CFG[n]` register. Verify that the `CDU_CLKINSEL` has the programmed value.
9. Verify that the `CDU_STAT.CLKO0` through `CDU_STAT.CLKO9` bits =0.

ADSP-SC58x CDU Register Descriptions

Clock Distribution Unit (CDU) contains the following registers.

Table 4-4: ADSP-SC58x CDU Register List

Name	Description
`CDU_CFG[n]`	CDU Configuration
`CDU_CLKINSEL`	CLKIN Select
`CDU_REVID`	CDU Revision ID
`CDU_STAT`	CDU Status

CDU Configuration

The `CDU_CFG[n]` registers control the configuration of the clock multiplexors. CDU0_CFG[n] corresponds to output clock CDU_CLKO[n].

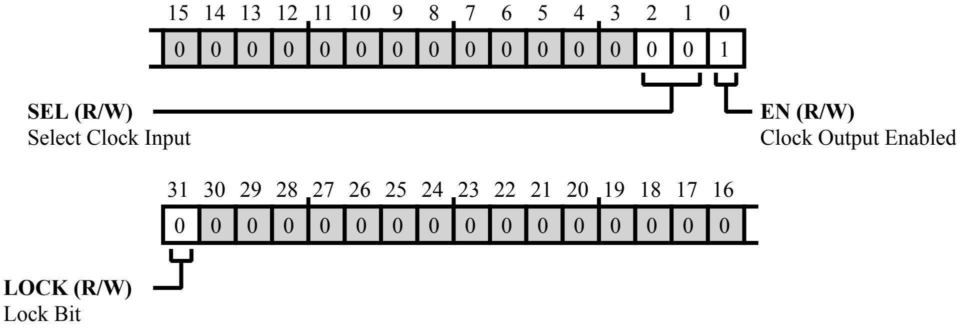

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
SEL (R/W)
Select Clock Input
EN (R/W)
Clock Output Enabled
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit

Figure 4-9: CDU_CFG[n] Register Diagram

Table 4-5: CDU_CFG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
2:1 (R/W)	SEL	Select Clock Input.	
		0	IN0_CLKOn Selected
		1	IN1_CLKOn Selected
		2	IN2_CLKOn Selected
		3	IN3_CLKOn Selected
0 (R/W)	EN	Clock Output Enabled. The `CDU_CFG[n].EN` bit enables clock output.	

CLKIN Select

The `CDU_CLKINSEL` register controls the configuration of the CLKIN multiplexors. One bit is assigned to each CGU in the system. This bit selects either CLKIN0 or CLKINn CGUn inputs.

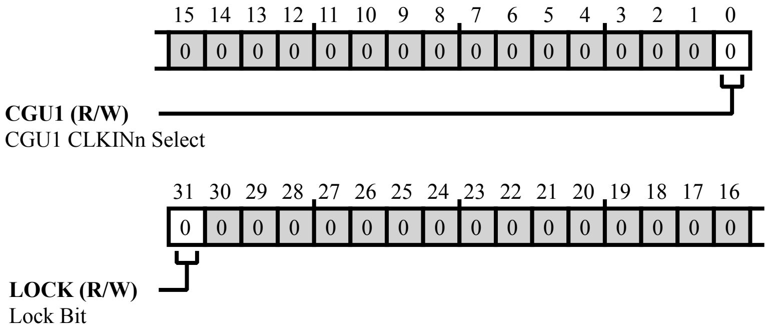

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CGU1 (R/W)
CGU1 CLKINn Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit

Figure 4-10: CDU_CLKINSEL Register Diagram

Table 4-6: CDU_CLKINSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
0 (R/W)	CGU1	CGU1 CLKINn Select. The `CDU_CLKINSEL.CGU1` bit drives CDU_CLKIN_SEL[0] to CGU1.	
		0	Selects CLKIN0
		1	Selects CLKIN1

CDU Revision ID

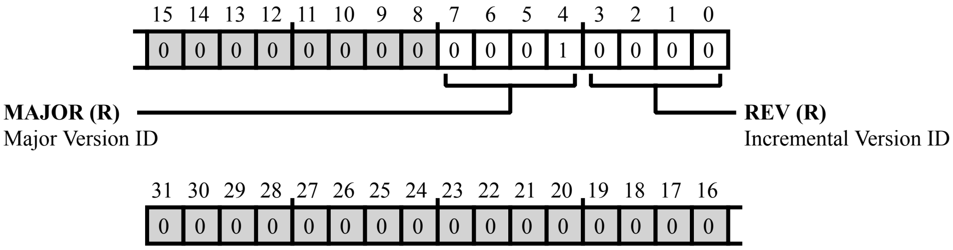

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
MAJOR (R)
Major Version ID
REV (R)
Incremental Version ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 4-11: CDU_REVID Register Diagram

Table 4-7: CDU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

CDU Status

The `CDU_STAT` register reflects the status a change in the configuration of the clock muxes.

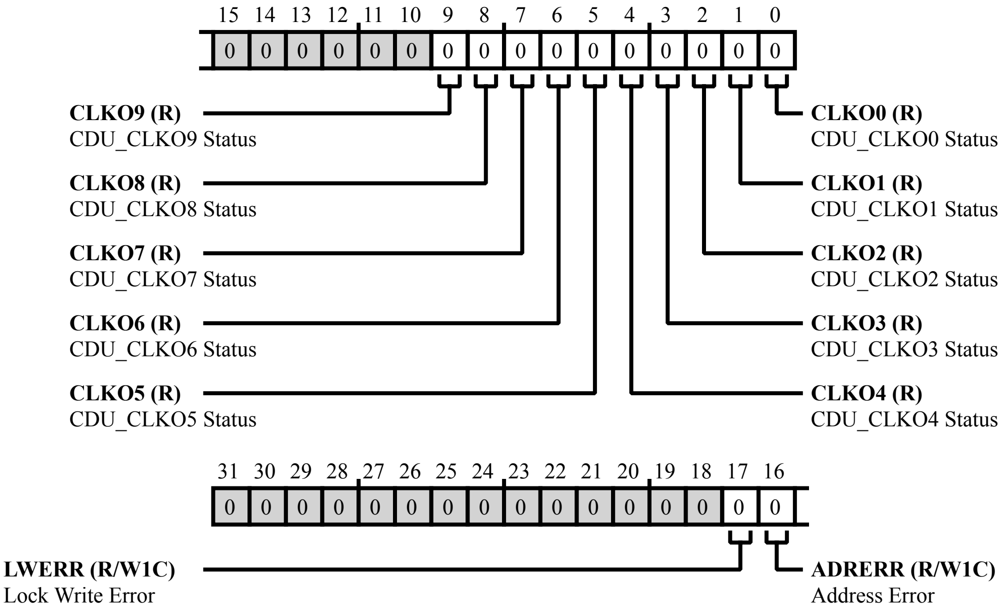

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKO9 (R)
CDU_CLKO9 Status
CLKO8 (R)
CDU_CLKO8 Status
CLKO7 (R)
CDU_CLKO7 Status
CLKO6 (R)
CDU_CLKO6 Status
CLKO5 (R)
CDU_CLKO5 Status
CLKO0 (R)
CDU_CLKO0 Status
CLKO1 (R)
CDU_CLKO1 Status
CLKO2 (R)
CDU_CLKO2 Status
CLKO3 (R)
CDU_CLKO3 Status
CLKO4 (R)
CDU_CLKO4 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LWERR (R/W1C)
Lock Write Error
ADRERR (R/W1C)
Address Error

Figure 4-12: CDU_STAT Register Diagram

Table 4-8: CDU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	LWERR	Lock Write Error. The `CDU_STAT.LWERR` bit indicates a lock error where write transactions try to access write protected registers.	
		0	No Lock Write Error
		1	Lock Write Error
16 (R/W1C)	ADRERR	Address Error. The `CDU_STAT.ADRERR` bit indicates an address error where read or write transactions try to access unimplemented addresses or accesses are non-aligned.	
		0	No Address Error
		1	Address Error
9 (R/NW)	CLKO9	CDU_CLKO9 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress

Table 4-8: CDU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	CLKO8	CDU_CLKO8 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
7 (R/NW)	CLKO7	CDU_CLKO7 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
6 (R/NW)	CLKO6	CDU_CLKO6 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
5 (R/NW)	CLKO5	CDU_CLKO5 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
4 (R/NW)	CLKO4	CDU_CLKO4 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
3 (R/NW)	CLKO3	CDU_CLKO3 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
2 (R/NW)	CLKO2	CDU_CLKO2 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
1 (R/NW)	CLKO1	CDU_CLKO1 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress
0 (R/NW)	CLKO0	CDU_CLKO0 Status.	
		0	No Configuration Change in Progress
		1	Configuration Change in Progress

5 Dynamic Power Management (DPM)

The dynamic power management (DPM) unit of the processor controls transitions between different power-saving modes.

DPM Features

The DPM allows programs to control the power mode of the processor as follows.

- Permits operation of multiple, external wake-up sources

DPM Functional Description

The DPM can be programmed to transition between power modes.

ADSP-SC58x DPM Register List

A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 5-1: ADSP-SC58x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_PER_DIS0	Peripherals Disable Register 0
DPM_PER_DIS1	Peripherals Disable Register 1
DPM_REVID	Revision ID
DPM_STAT	Status Register

DPM Definitions

To make the best use of the DPM, it is useful to understand the following terms.

CGU

Acronym for the clock generation unit (CGU), which is comprised of the PLL and PCU

DPM

Acronym for the dynamic power management (DPM) controller.

Full-on mode

The normal operating mode in which all clock domains are derived from the PLL.

PCU

Acronym for the PLL control unit (PCU).

PLL

Acronym for the phase-locked loop (PLL).

RCU

Acronym for the reset control unit (RCU).

DPM Operating Modes

The DPM includes several operating modes. The modes are:

- Reset
- Full-on

Reset State

Reset is the initial state of the processor and is the result of a hardware or software triggered event. The DPM itself does not trigger entering reset. The external `SYS_HWRST` pin or the RCU triggers entering reset. The DPM responds to reset by transitioning to its default state.

From Reset, the DPM always transitions to PLL Bypassed state.

Full-on Mode

Full-on mode is the default state of the DPM after Reset.

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest.

DPM Event Control

The DPM event is triggered when an enabled wake-up is asserted. The DPM generates bus errors when a misaligned access to a register occurs. It also generates errors when an attempt is made to access unused DPM address space or a write-protected register.

DPM Events

The DPM event interrupt is triggered when any bit in the `DPM_STAT` register is set, indicating that an enabled wake-up was asserted. The DPM event interrupt stays active until the user clears any bits that are set in the `DPM_STAT` register.

DPM Errors

The DPM generates a bus error when a read or write transaction is attempted to an unused address within the DPM address range. It also generates a bus error when a misaligned access is made to a DPM register. In addition to the bus error, the DPM sets the `DPM_STAT.ADDRERR` bit.

If a write to a write-protected DPM register is attempted, the DPM generates a bus error. In addition, the DPM sets the `DPM_STAT.LWERR` bit.

DPM Programming Model

The ***DPM_PER_DIS0 Register Mapping*** table shows the module clocks and the corresponding peripheral. The `DPM_PER_DIS0` register is used to shut off the clock to each peripheral if it is not required by the application.

NOTE: In the table, SMPU-L2CTL-CL2_x and SMPU-L2CTL-DL2_x correspond to the SMPU modules associated with the core and the DMA ports of L2 respectively. The six instances of SMPU blocks are shown in the Figure 9-1 ADSP-SC58x Complete L2 System Block Diagram – two SMPUs for each L2CLTL, one for the port 0-core and another for the port 1-DMA.

Table 5-2: DPM_PER_DIS0 Register Mapping

Peripheral Name	Gated Module clocks	DPM_PER_DIS0 bit
FIR0	SCLK0	0
IIR0	SCLK0	1
FFT0	SYSCLK	2
RTC0	SCLK0	4
DAI0	SCLK0 CLK05 SYSCLK	5

Table 5-2: DPM_PER_DIS0 Register Mapping (Continued)

Peripheral Name	Gated Module clocks	DPM_PER_DIS0 bit
DAI1	SCLK0 CLK05 SYSCLK	6
HAE0	SCLK0	7
SINC0	SCLK0	8
MLB0	SYSCLK	9
EMAC0 (GigE)	CLKO7 SCLK0	10
EMAC1(10/100)	SCLK0	11
MSI0	CLKO9 SCLK0	12
EMDMA0/CH0	SCLK0	13
EMDMA0/CH1	SCLK0	14
PCIe	CLKO6 SYSCLK	15
CRYPTO ACCELERATOR-0(EIP-150/PKP)	SCLK0	16
CRYPTO ACCELERATOR-1(EIP-93/SPE)	SCLK0	17
SMPU-SMC	SCLK0	18
SMPU-SPI2	SCLK1	19
SMPU-L2CTL-CL2-0	SYSCLK	20
SMPU-L2CTL-DL2-0	SYSCLK	21
SMPU-L2CTL-CL2-1	SYSCLK	22
SMPU-L2CTL-DL2-1	SYSCLK	23
SMPU-L2CTL-CL2-2	SYSCLK	24
SMPU-L2CTL-DL2-2	SYSCLK	25
SMPU-PCIE-Slave	SYSCLK	26
SMPU-DMC0	SYSCLK	27
SMPU-DMC1	SYSCLK	28
CAN0	CLKO4	29
CAN1	CLKO4	30

Table 5-3: DPM_PER_DIS1 Register Mapping

Peripheral Name	Gated Module Clocks	DPM_PER_DIS1 bit
TWI0	SCLK0	0
TWI1	SCLK0	1
TWI2	SCLK0	2
USB0	SCLK0	3
USB1	SCLK0	4

ADSP-SC58x DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 5-4: ADSP-SC58x DPM Register List

Name	Description
`DPM_CTL`	Control Register
`DPM_PER_DIS0`	Peripherals Disable Register 0
`DPM_PER_DIS1`	Peripherals Disable Register 1
`DPM_REVID`	Revision ID
`DPM_STAT`	Status Register

Control Register

The `DPM_CTL` register controls sleep modes selections and PLL operations of the DPM. A write protect feature permits locking out changes to this register.

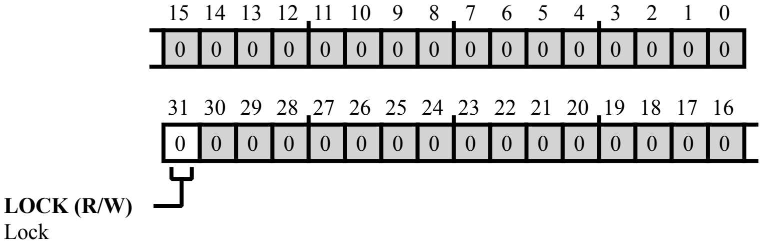

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 5-1: DPM_CTL Register Diagram

Table 5-5: DPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `DPM_CTL.LOCK` bit is set, the `DPM_CTL` register is read only (locked).	
		0	Unlock
		1	Lock

Peripherals Disable Register 0

The `DPM_PER_DIS0` register is used to shut off the clocks to peripherals that are not used in an application to save clock switching power.

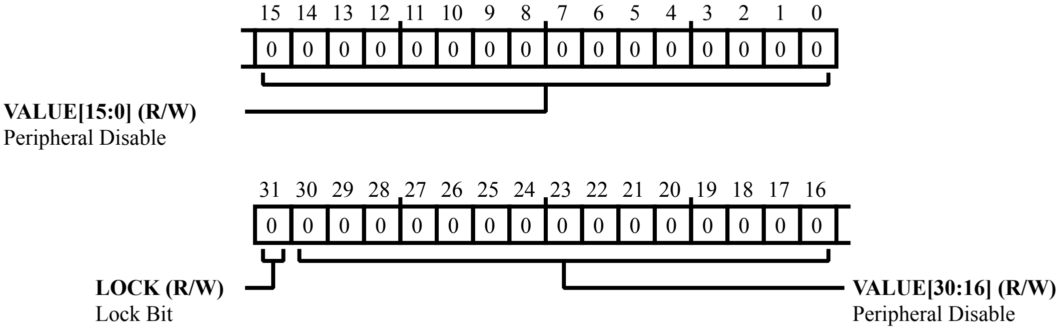

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Peripheral Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit
VALUE[30:16] (R/W)
Peripheral Disable

Figure 5-2: DPM_PER_DIS0 Register Diagram

Table 5-6: DPM_PER_DIS0 Register Fields

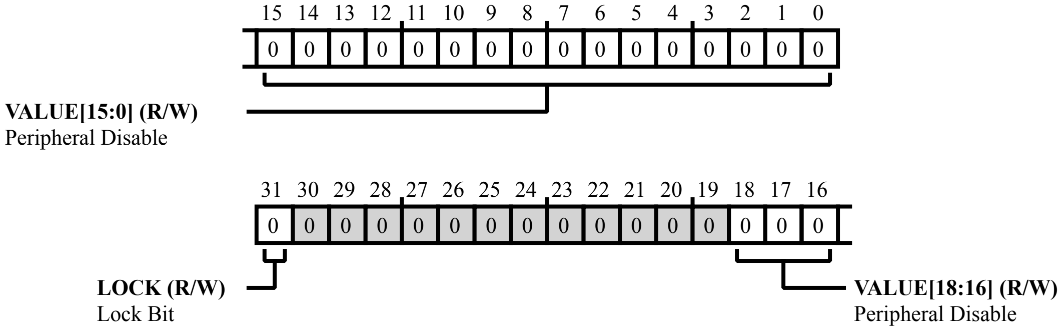
Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
30:0 (R/W)	VALUE	Peripheral Disable. The `DPM_PER_DIS0.VALUE` bits are used to shut of clocks in individual peripherals.	
		0	FIR0
		1	IIR0
		2	FFT0
		3	Reserved
		4	RTC0
		5	DAI0
		6	DAI1
		7	HAE0
		8	SINC0
		9	MLB0
		10	EMAC0 (GigE)
		11	EMAC1(10/100)
		12	MSI0
		13	EMDMA0/CH1

Table 5-6: DPM_PER_DIS0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		14	EMDMA0/CH0
		15	PCIe-Slave
		16	CRYPTO ACCELERATOR-0 (EIP-150/PKP)
		17	CRYPTO ACCELERATOR-1 (EIP-150/PKP)
		18	SMPU-SMC
		19	Reserved
		20	SMPU-L2CTL-CL2-0
		21	SMPU-L2CTL-DL2-0
		22	SMPU-L2CTL-CL2-1
		23	SMPU-L2CTL-DL2-1
		24	SMPU-L2CTL-CL2-2
		25	SMPU-L2CTL-DL2-2
		26	SMPU-PCIE-SLV
		27	SMPU-DMC0
		28	SMPU-DMC1
		29	CAN0
		30	CAN1

Peripherals Disable Register 1

The `DPM_PER_DIS1` register is used to shut off the clocks to peripherals that are not used in an application to save clock switching power.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Peripheral Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit
VALUE[18:16] (R/W)
Peripheral Disable

Figure 5-3: DPM_PER_DIS1 Register Diagram

Table 5-7: DPM_PER_DIS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
18:0 (R/W)	VALUE	Peripheral Disable. The `DPM_PER_DIS1.VALUE` bits are used to shut of clocks in individual peripherals.	
		0	TWI0
		1	TWI1
		2	TWI2
		3	USB0
		4	USB1
		5	Reserved
		6	Reserved
		7	Reserved
		8	Reserved
		9	Reserved
		10	Reserved
		11	Reserved
		12	Reserved
		13	Reserved

Table 5-7: DPM_PER_DIS1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		14	Reserved
		15	Reserved
		16	Reserved
		17	Reserved
		18	Reserved

Revision ID

The `DPM_REVID` register provides the revision of the DPM module.

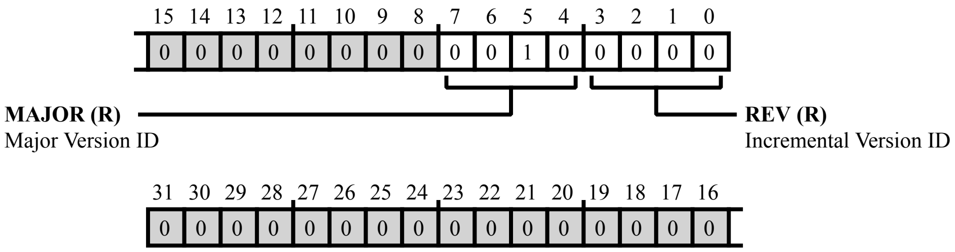

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
MAJOR (R)
Major Version ID
REV (R)
Incremental Version ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 5-4: DPM_REVID Register Diagram

Table 5-8: DPM_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Status Register

The DPM_STAT register contains bits that report the state of the module and various errors.

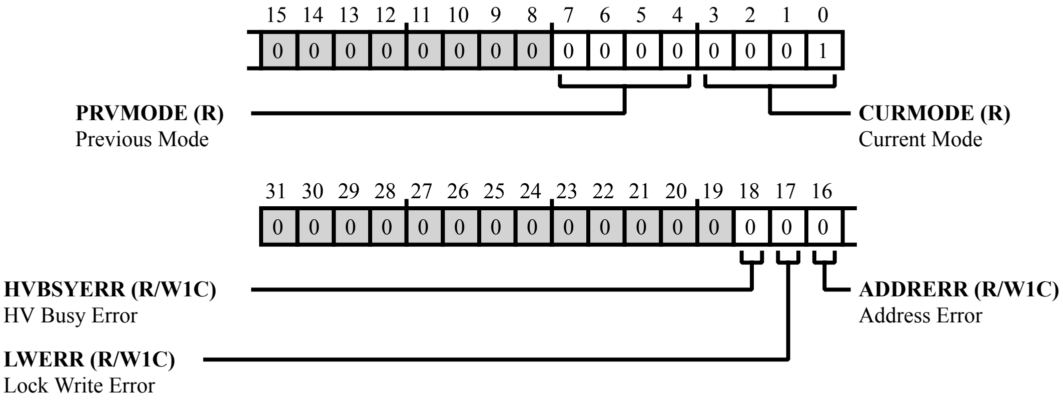

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
PRVMODE (R)
Previous Mode
CURMODE (R)
Current Mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HVBSYERR (R/W1C)
HV Busy Error
ADDRERR (R/W1C)
Address Error
LWERR (R/W1C)
Lock Write Error

Figure 5-5: DPM_STAT Register Diagram

Table 5-9: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	HVBSYERR	HV Busy Error. The `DPM_STAT.HVBSYERR` bit indicates that a Read access to the Wake Status or Restoren registers (during engineering transfers only) while the DPMLV is transferring registers from the DPMHV. Triggers the DPMLV_PSLVERR interrupt.	
		0	Inactive
		1	Active
17 (R/W1C)	LWERR	Lock Write Error. The `DPM_STAT.LWERR` bit indicates that a write transaction attempted an access to a write protected register. Triggers the DPMLV_PSLVERR interrupt.	
		0	Inactive
		1	Active
16 (R/W1C)	ADDRERR	Address Error. The `DPM_STAT.ADDRERR` bit indicates that a read or write transaction attempted an access to an unimplemented address or a write transaction attempted an access to a read only register or accesses are non aligned. Triggers the DPMLV_PSLVERR interrupt.	
		0	Inactive
		1	Active
7:4 (R/NW)	PRVMODE	Previous Mode. The `DPM_STAT.PRVMODE` bit field indicates the previous mode of the the module.	

Table 5-9: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Reset
		1	Full-On
		2	Reserved
		3	Reserved
		4	Reserved
		5-15	Reserved
3:0 (R/NW)	CURMODE	Current Mode. The `DPM_STAT.CURMODE` bit field indicates the current mode of the the module.	
		0	Reserved
		1	Full-On
		2	Reserved
		3	Reserved
		4-15	Reserved

6 Reset Control Unit (RCU)

Reset is the initial state of the whole processor at power-on or the run-time state of any core, as controlled by another core in the device via the RCU or as a a result of a hardware or software triggered event. In this state, all control registers are set to their default values and functional units are idle. Exiting a full system hardware reset results with only the host core being released from reset and ready to boot (Core 0 on ADSP-SC58x devices, Core 1 on ADSP-215xx devices). The following table shows the cores that are released from reset.

Additional information on reset and booting can be found in Boot ROM and Booting the Processor.

Product	Core Released from Reset by RCU
ADSP-SC58x	Core ID = 0
ADSP-2158x	Core ID = 1

The Reset Control Unit (RCU) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. This functionality is important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset). While core resets and software system resets are controlled directly in the RCU, hardware resets can come from the TRU, the SEC, or the CGU Oscillator Watchdog.

RCU Features

The RCU module supports the following features:

- Hardware reset through the $\overline{\texttt{SYS\_HWRST}}$ pin
- Software system reset through the RCU control (`RCU_CTL`) register
- Hardware system reset through:
    - TRU module
    - SEC module (System Fault Unit)
- A clock not good reset state (safe state of chip under reset) from the Oscillator Watchdog.
- Core reset through RCU Core Reset Output (`RCU_CRCTL`) register

RCU Functional Description

This section provides information on the function of RCU module.

Hardware reset using SYS_HWRST pin

Asserting the `SYS_HWRST` pin resets all functional units, except the real time clock (RTC) (if present).

Hardware reset through RCU

The RCU can perform a full system reset which can be initiated through hardware blocks like the SEC, the TRU, and the oscillator watchdog.

Software reset using RCU registers

Setting the `RCU_STAT.SWRST` bit issues a software reset for all system units except the RTC module and `RCU_BCODE`, `RCU_CRCTL` and `RCU_STAT` registers.

Core reset RCU registers

A core can be individually reset by software, or by setting the `RCU_CRCTL.CR[n]` bit.

ADSP-SC58x RCU Register List

The Reset Control Unit (RCU) controls how all the functional units in the processor enter and exit Reset. Differences in functional requirements and clocking constraints exist (units in different clock domains have to enter reset asynchronously, but units exit reset in a deterministic way), and these differences define how reset signals are generated. Reset signals propagate through all functional units asynchronously. For more information on RCU functionality, see the RCU register descriptions.

Table 6-1: ADSP-SC58x RCU Register List

Name	Description
`RCU_BCODE`	Boot Code Register
`RCU_CRCTL`	Core Reset Outputs Control Register
`RCU_CRSTAT`	Core Reset Outputs Status Register
`RCU_CTL`	Control Register
`RCU_MSG`	Message Register
`RCU_MSG_CLR`	Message Clear Bits Register
`RCU_MSG_SET`	Message Set Bits Register
`RCU_SIDIS`	System Interface Disable Register
`RCU_SISTAT`	System Interface Status Register

Table 6-1: ADSP-SC58x RCU Register List (Continued)

Name	Description
`RCU_STAT`	Status Register
`RCU_SVECT0`	Software Vector Register 0
`RCU_SVECT1`	Software Vector Register 1
`RCU_SVECT2`	Software Vector Register 2
`RCU_SVECT_LCK`	SVECT Lock Register

ADSP-SC58x RCU Trigger List

Table 6-2: ADSP-SC58x RCU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 6-3: ADSP-SC58x RCU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
50	`RCU0_SYSRST0`	RCU0 System Reset 0	Pulse
51	`RCU0_SYSRST1`	RCU0 System Reset 1	Pulse

RCU Definitions

To make the best use of the RCU, it is useful to understand the terms in this section.

The target or source defines the following are types of resets.

Hardware Reset (by target)

All functional units except a small subsection of debug interfaces are set to their default states. State is lost in all non-volatile storage.

System Reset (by target)

All functional units except the RCU, flash interface, and debug are set to their default states.

Core n Only Reset (by target)

Affects Core n only. The system software must guarantee that a bus master cannot access the core in reset state.

Hardware Reset (by source)

The $\overline{\texttt{SYS\_HWRST}}$ input signal is asserted active (pulled low).

System Reset (by source)

Software can trigger the reset by writing to the `RCU_CTL` register or by another functional unit such as the TRU or any of the generic reset inputs.

RCU Architectural Concepts

To understand the architecture of the RCU, it is important to consider the reset sources and how differing resets affect the functional units of the processor.

The RCU provides the hardware that controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. For example, units in different clock domains must enter reset asynchronously but exit reset in a deterministic way.

The program must guarantee that none of the reset functions put the system in an undefined state or cause resources to stall. This functionality is important when only one of the cores is reset. The program must guarantee that there is no pending system activity involving Core n before it is reset. For example, there must be no pending transactions to core 0 when the core 0 is reset and vice-versa.

The ***RCU Reset Sources*** table defines how reset sources affect the different functional units.

Table 6-4: RCU Reset Sources

Reset Source	Reset Type	Affected Functional Units
$\overline{\texttt{SYS\_HWRST}}$ pin assertion	Hardware Reset	All functional units, except RTC (if present)
SYSCLK clock domain system reset by fault unit (FMU) or TRU master	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
`RCU_CTL.SYSRST` bit set (software triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
`RCU_CRCTL.CR[n]` bit set (software triggered reset)	Core Only Reset	Core n only, for $(2 \geq n \geq 0)$

RCU Status and Error Signals

The `RCU_STAT` register reflects status and error information. There are three kinds of errors that can occur in the RCU. The reset out error is triggered when `RSTOUT` is both asserted and deasserted at the same time. The lock write error occurs if an attempt is made to write a lock RCU register. The address error occurs if a read-only register is written to or if an attempt is made to a reserved address within the RCU MMR address range.

Resetting the ARM Core through Another Core or System Master

The RCU allows reset of a given core n using another core or system master. Core 0 can be individually reset by software, either setting any of CR0 bit in the `RCU_CRCTL` register. Cores that reset themselves cannot guarantee that all the system transactions to or from it have completed. Although a core n reset can be triggered by core n itself, it is recommended that another core or system master trigger it. Core n can be reset to restore its functionality when it cannot execute software.

The following steps show the suggested programming sequence to reset core n only.

1. Clear the `RCU_CRSTAT.CR[n]` bit.
2. Disable interrupts to core n
3. Set the `RCU_SIDIS.SI[n]` bit to disable the interfaces for core n, to stop DMA accesses to its L1, to stop accesses to memory for core n, and stop accesses to MMRs.
4. Test the `RCU_SISTAT.SI[n]` bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
5. Set the `RCU_CRCTL.CR[n]` bit to reset core n.
6. Poll the `RCU_CRSTAT.CR[n]` bit until core n is in reset.
7. Once the core is in reset, clear the `RCU_SIDIS.SI[n]` bit to reenable the core interfaces.
8. Clear the `RCU_CRCTL.CR[n]` bit to take core n out of reset.
9. Poll the `RCU_CRSTAT.CR[n]` bit until core n is out of reset.

Resetting a SHARC+ Core Through Another Core

Resetting a SHARC+ core involves a software handshake between the Master core which issues a reset to the SHARC+ core. The handshake is done using a core interrupt along with message passing through a variable in shared L2 memory RAM. Each SHARC+ core needs two bits for handshaking.

- Core Reset Request bit (CRR). This bit is set by the master core to indicate to the SHARC+ core that a core reset needs to be done. If the CRR bit is set the SHARC+ core should disable all interrupts, stop all system and memory accesses and enter the IDLE state.
- IDLE acknowledgement (IDLE). Once the SHARC+ core is ready for reset, it sets this bit before entering the IDLE state to inform the master core that it is ready for reset.

Two bits are needed for SHARC0 core and two bits are needed for SHARC1 core. These bits are:

- CCR0 – SHARC0 core reset request bit in shared variable
- IDLE0 – SHARC0 IDLE acknowledgement bit in shared variable
- CCR1 – SHARC1 core reset request bit in shared variable
- IDLE1 – SHARC1 IDLE acknowledgement bit in shared variable

Use the following programming to reset the SHARC+ core.

1. The master core checks the IDLE status bits in shared variable for the corresponding SHARC+ core (`RCU_MSG.C1IDLE` and `RCU_MSG.C0IDLE`).
2. If one of the core idle bits is set then the program jumps to step 10. Otherwise the process continues to step 3.
3. The master core sets the CCRx bit in message register and raises the SOFT0 software interrupt through the SEC to the SHARC+ core (see Programming Examples for more information).
4. The SHARC+ core goes to into the ISR, and checks the “CCRx” status bit in RCU_MSG register to ensure that software interrupt is for core reset.

   ADDITIONAL INFORMATION: The SOFT0 software interrupt handler can be used for other non-reset/ general purposes as well.

   ADDITIONAL INFORMATION: The ISR should be in L1 memory.
5. The core should disable all interrupts before entering IDLE for reset.
6. The core sets the appropriate idle status bit (`RCU_MSG.C1IDLE` and `RCU_MSG.C0IDLE`).
7. Core enters IDLE.
8. Master core polls for the IDLEx status bit
9. The master core keeps a timeout option where if within N number of cycles the SHARC core doesn’t respond then a system reset is initiated.

   ADDITIONAL INFORMATION: N can be decided by the user depending on timing criticality of the application.
10. Once the IDLEx status bit is found set, the master core initiates a core reset though the RCU.
11. Clear the `RCU_CRSTAT.CR[n]` bit.
12. Set the `RCU_SIDIS.SI[n]` bit to disable the interfaces for core n, to stop DMA accesses to its L1, to stop accesses to memory for core n, and stop accesses to MMRs.
13. Test the `RCU_SISTAT.SI[n]` bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
14. Set the `RCU_CRCTL.CR[n]` bit to reset core n.

15. Poll the `RCU_CRSTAT.CR[n]` bit until core n is in reset.
16. Once the core is in reset, clear the `RCU_SIDIS.SI[n]` bit to re-enable the core interfaces.
17. Clear the `RCU_CRCTL.CR[n]` bit to take core n out of reset.
18. Poll the `RCU_CRSTAT.CR[n]` bit until core n is out of reset.

If a core is servicing a higher priority interrupt and gets stuck then it may not respond to the SEC.

ADSP-SC58x RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 6-5: ADSP-SC58x RCU Register List

Name	Description
`RCU_BCODE`	Boot Code Register
`RCU_CRCTL`	Core Reset Outputs Control Register
`RCU_CRSTAT`	Core Reset Outputs Status Register
`RCU_CTL`	Control Register
`RCU_MSG`	Message Register
`RCU_MSG_CLR`	Message Clear Bits Register
`RCU_MSG_SET`	Message Set Bits Register
`RCU_SIDIS`	System Interface Disable Register
`RCU_SISTAT`	System Interface Status Register
`RCU_STAT`	Status Register
`RCU_SVECT0`	Software Vector Register 0
`RCU_SVECT1`	Software Vector Register 1
`RCU_SVECT2`	Software Vector Register 2
`RCU_SVECT_LCK`	SVECT Lock Register

Boot Code Register

The `RCU_BCODE` register can be used to determine if and how core boots. This register is set to its default values by RESET.

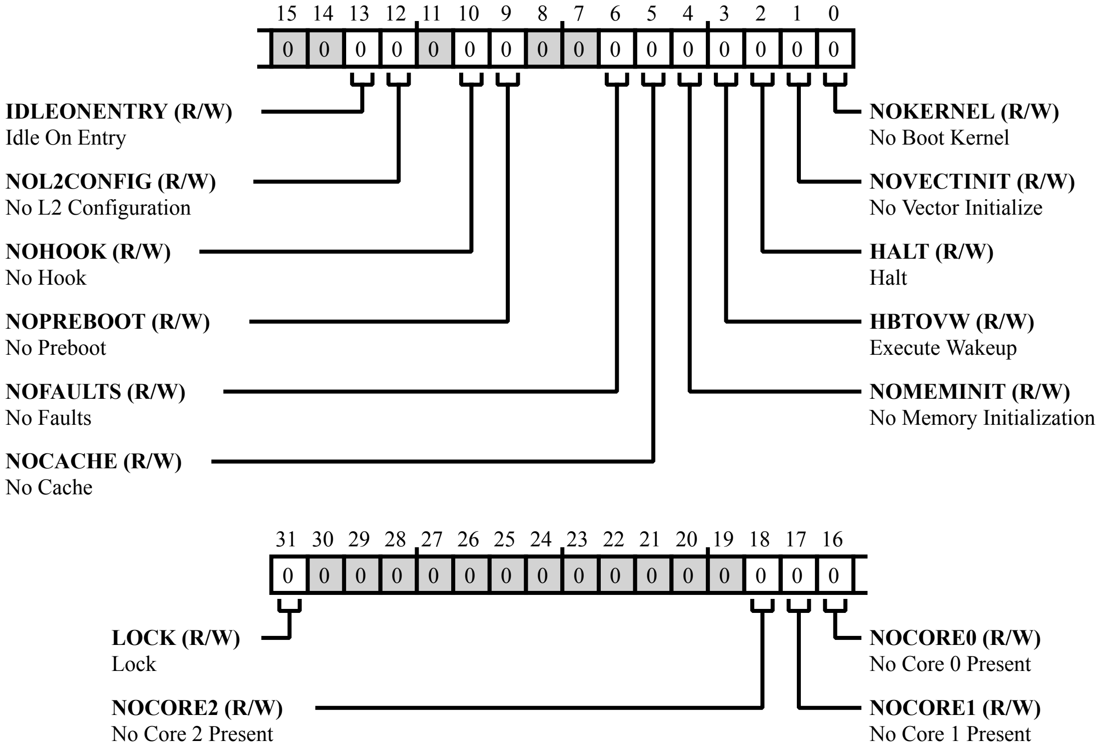

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IDLEONENTRY (R/W)
Idle On Entry
NOL2CONFIG (R/W)
No L2 Configuration
NOHOOK (R/W)
No Hook
NOPREBOOT (R/W)
No Preboot
NOFAULTS (R/W)
No Faults
NOCACHE (R/W)
No Cache
NOKERNEL (R/W)
No Boot Kernel
NOVECTINIT (R/W)
No Vector Initialize
HALT (R/W)
Halt
HBTOVW (R/W)
Execute Wakeup
NOMEMINIT (R/W)
No Memory Initialization
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
NOCORE2 (R/W)
No Core 2 Present
NOCORE0 (R/W)
No Core 0 Present
NOCORE1 (R/W)
No Core 1 Present

Figure 6-1: RCU_BCODE Register Diagram

Table 6-6: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_BCODE.LOCK` bit is set, the `RCU_BCODE` register is read only (locked).	
		0	Unlock
		1	Lock
18 (R/W)	NOCORE2	No Core 2 Present. The `RCU_BCODE.NOCORE2` bit indicates the presence of core 2.	
		0	Core does not exist
		1	Core exists

Table 6-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	NOCORE1	No Core 1 Present. The `RCU_BCODE.NOCORE1` bit indicates the presence of core 1.	
		0	Core does not exist
		1	Core exists
16 (R/W)	NOCORE0	No Core 0 Present. The `RCU_BCODE.NOCORE0` bit indicates the presence of core 0.	
		0	Core does not exist
		1	Core exists
13 (R/W)	IDLEONENTRY	Idle On Entry. The `RCU_BCODE.IDLEONENTRY` bit configures the RCU to enter the idle state at startup.	
		0	Do not enter idle state
		1	Enter idle state
12 (R/W)	NOL2CONFIG	No L2 Configuration. The `RCU_BCODE.NOL2CONFIG` bit configures the RCU to not perform the L2 memory configuration.	
		0	Configure L2 memory
		1	Do not configure L2 memory
10 (R/W)	NOHOOK	No Hook. The `RCU_BCODE.NOHOOK` bit configures the RCU to not perform the hook routine.	
		0	Perform hook routine
		1	Do not perform hook routine
9 (R/W)	NOPREBOOT	No Preboot. The `RCU_BCODE.NOPREBOOT` bit configures the RCU to not perform the customer preboot routine.	
		0	Perform preboot
		1	Do not perform preboot
6 (R/W)	NOFAULTS	No Faults. The `RCU_BCODE.NOFAULTS` bit configures the RCU to not perform fault initialization.	
		0	Perform fault initialization
		1	Do not perform fault initialization

Table 6-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	NOCACHE	No Cache. The `RCU_BCODE.NOCACHE` bit configures the RCU to not perform a cache initialization and to not enable the cache.	
		0	Enable and initialize cache
		1	Do not initialize or enable cache
4 (R/W)	NOMEMINIT	No Memory Initialization. The `RCU_BCODE.NOMEMINIT` bit configures the RCU to not perform a memory initialization.	
		0	Perform memory initialization
		1	Do not perform memory initialization
3 (R/W)	HBTOVW	Execute Wakeup. The `RCU_BCODE.HBTOVW` bit configures the RCU to execute a wakeup.	
		0	Do not wakeup
		1	Execute wakeup
2 (R/W)	HALT	Halt. The `RCU_BCODE.HALT` bit configures the RCU to execute the no boot routine.	
		0	Do not execute routine
		1	Execute routine
1 (R/W)	NOVECTINIT	No Vector Initialize. The `RCU_BCODE.NOVECTINIT` bit configures the RCU to not vector to the application.	
		0	Vector
		1	Do not vector
0 (R/W)	NOKERNEL	No Boot Kernel. The `RCU_BCODE.NOKERNEL` bit configures the RCU to not execute the boot kernel.	
		0	Execute boot kernel
		1	Do not execute boot kernel

Core Reset Outputs Control Register

The RCU core reset control n registers (`RCU_CRCTL`) include a lock bit (`RCU_CRCTL.LOCK`) and a core reset bit (`RCU_CRCTL.CR[n]`) for each core reset signal on the product.

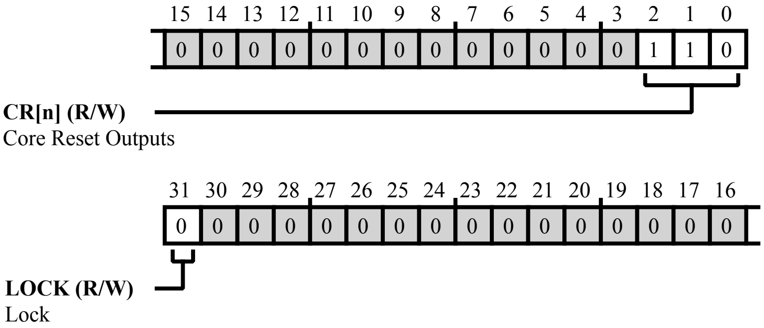

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
CR[n] (R/W)
Core Reset Outputs
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 6-2: RCU_CRCTL Register Diagram

Table 6-7: RCU_CRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_CRCTL.LOCK` bit is set, the `RCU_CRCTL` register is read only (locked).	
		0	Unlock
		1	Lock
2:0 (R/W)	CR[n]	Core Reset Outputs. The `RCU_CRCTL.CR[n]` bits control CRES[1:0] core reset signals. The RCU_CRES[n] signals can be individually controlled. They are reset to their default value by a hard reset or a system reset. For each RCU_CRES[n], the selected RCU0_CRMSKi[n] bit is cleared.	
		0	RCU_CRES[2:0] Deasserted
		7	RCU_CRES[2:0] Asserted

Core Reset Outputs Status Register

The RCU core reset status register (RCU_CRSTAT) contains status bits, indicating which core reset signals have been asserted.

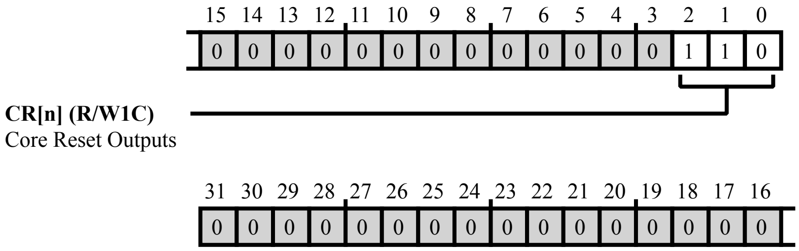

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
CR[n] (R/W1C)
Core Reset Outputs
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 6-3: RCU_CRSTAT Register Diagram

Table 6-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W1C)	CR[n]	Core Reset Outputs. The `RCU_CRSTAT.CR[n]` bits indicate which cores have been reset since the last time the bit was cleared. Bits masked by CORE_DISABLE_MASK[15:0] are permanently disabled and the corresponding CR bits set. CR bits are sticky, they need to be cleared by software.	
		0	RCU_CRES[1:0] deasserted. CR[n] corresponds to RCU_CRES[n].
		7	RCU_CRES[2:0] were asserted since the last time bits were cleared. CR[n] corresponds to RCU_CRES[n].

Control Register

The RCU control register (`RCU_CTL`) provides a register lock, enables for the core and system reset requests inputs and control for the Reset Output pin.

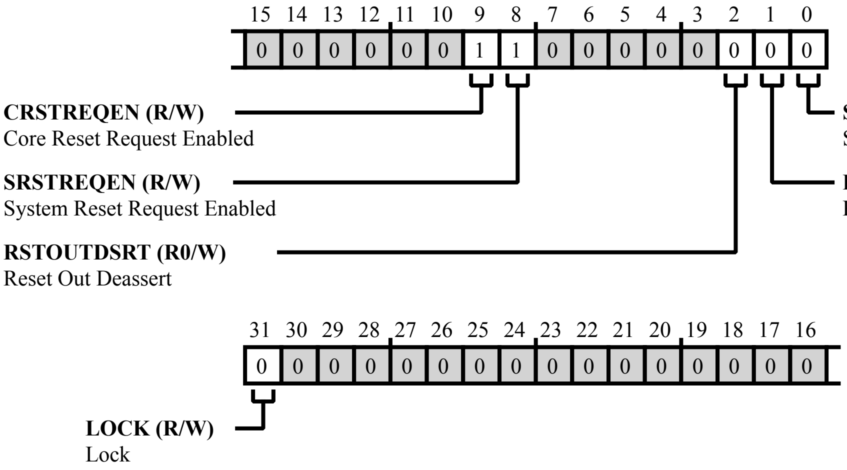

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
CRSTREQEN (R/W)
Core Reset Request Enabled
SRSTREQEN (R/W)
System Reset Request Enabled
RSTOUTDSRT (R0/W)
Reset Out Deassert
SYSRST (R0/W)
System Reset
RSTOUTASRT (R0/W)
Reset Out Assert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 6-4: RCU_CTL Register Diagram

Table 6-9: RCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_CTL.LOCK` bit is set, the `RCU_CTL` register is read only (locked). This bit is cleared by a hard reset or any system reset event.	
		0	Unlock
		1	Lock
9 (R/W)	CRSTREQEN	Core Reset Request Enabled. The `RCU_CTL.CRSTREQEN` bit controls whether the SYSCLK domain source(s) of reset is/are enabled to reset the core(s) when asserted. This bit is cleared by hard reset or any system reset event.	
		0	Disabled
		1	Enabled
8 (R/W)	SRSTREQEN	System Reset Request Enabled. The `RCU_CTL.SRSTREQEN` bit controls whether the SYSCLK domain sources of reset are enabled to do a system reset when asserted. This bit is cleared by a hard reset.	
		0	Disabled
		1	Enabled

Table 6-9: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R0/W)	RSTOUTDSRT	Reset Out Deassert. The `RCU_CTL.RSTOUTDSRT` bit controls the deassertion of the system reset pin.	
		0	No Action
		1	Deassert RSTOUT
1 (R0/W)	RSTOUTASRT	Reset Out Assert. The `RCU_CTL.RSTOUTASRT` bit controls assertion of the system reset pin.	
		0	No Action
		1	Assert RSTOUT
0 (R0/W)	SYSRST	System Reset. The `RCU_CTL.SYSRST` bit provides reset for all system units.	
		0	No Action
		1	System Reset

Message Register

The `RCU_MSG` is a general-purpose register. It is intended to provide flexibility for Boot ROM code and to pass predefined variables to the debugger. Please see the Booting chapter for product-specific details.

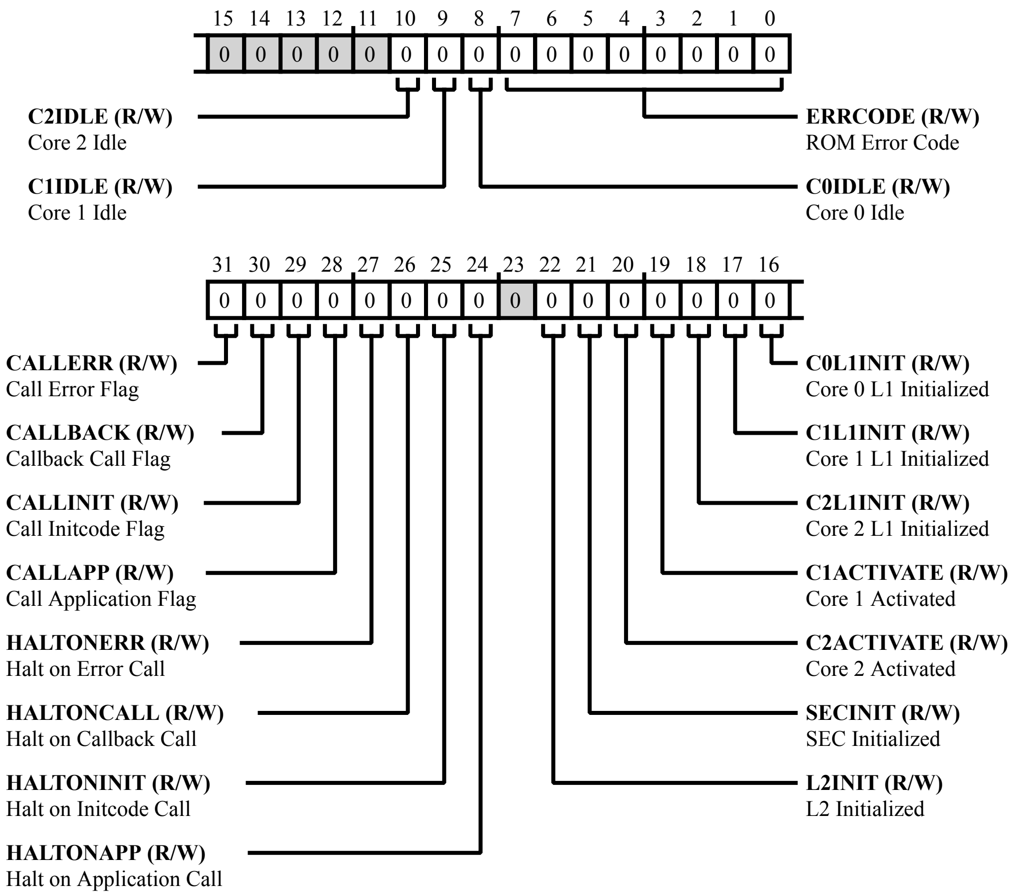

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
C2IDLE (R/W)
Core 2 Idle
C1IDLE (R/W)
Core 1 Idle
ERRCODE (R/W)
ROM Error Code
C0IDLE (R/W)
Core 0 Idle
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CALLERR (R/W)
Call Error Flag
CALLBACK (R/W)
Callback Call Flag
CALLINIT (R/W)
Call Initcode Flag
CALLAPP (R/W)
Call Application Flag
HALTONERR (R/W)
Halt on Error Call
HALTONCALL (R/W)
Halt on Callback Call
HALTONINIT (R/W)
Halt on Initcode Call
HALTONAPP (R/W)
Halt on Application Call
C0L1INIT (R/W)
Core 0 L1 Initialized
C1L1INIT (R/W)
Core 1 L1 Initialized
C2L1INIT (R/W)
Core 2 L1 Initialized
C1ACTIVATE (R/W)
Core 1 Activated
C2ACTIVATE (R/W)
Core 2 Activated
SECINIT (R/W)
SEC Initialized
L2INIT (R/W)
L2 Initialized

Figure 6-5: RCU_MSG Register Diagram

Table 6-10: RCU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CALLERR	Call Error Flag. The `RCU_MSG.CALLERR` bit indicates that a flag has been set by the boot code prior to an error call.	
		0	Flag not set
		1	Flag set

Table 6-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	CALLBACK	Callback Call Flag. The `RCU_MSG.CALLBACK` bit indicates that a flag has been set by the boot code prior to a callback call.	
		0	Flag not set
		1	Flag set
29 (R/W)	CALLINIT	Call Initcode Flag. The `RCU_MSG.CALLINIT` bit indicates that a flag has been set by the boot code prior to an initcode call.	
		0	Flag not set
		1	Flag set
28 (R/W)	CALLAPP	Call Application Flag. The `RCU_MSG.CALLAPP` bit indicates that a flag has been set by the boot code prior to an application call.	
		0	Flag not set
		1	Flag set
27 (R/W)	HALTONERR	Halt on Error Call. The `RCU_MSG.HALTONERR` bit generates an emulation exception prior to an error call.	
		0	Do not generate exception
		1	Generate exception
26 (R/W)	HALTONCALL	Halt on Callback Call. The `RCU_MSG.HALTONCALL` bit generates an emulation exception prior to a callback call.	
		0	Do not generate exception
		1	Generate exception
25 (R/W)	HALTONINIT	Halt on Initcode Call. The `RCU_MSG.HALTONINIT` bit generates an emulation exception prior to an initcode call.	
		0	Do not generate exception
		1	Generate exception

Table 6-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	HALTONAPP	Halt on Application Call. The `RCU_MSG.HALTONAPP` bit generates an emulation exception prior to an application call.	
		0	Do not generate exception
		1	Generate exception
22 (R/W)	L2INIT	L2 Initialized. The `RCU_MSG.L2INIT` bit indicates that the L2 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
21 (R/W)	SECINIT	SEC Initialized. The `RCU_MSG.SECINIT` bit is used by tools for initialization of the SEC.	
20 (R/W)	C2ACTIVATE	Core 2 Activated. The `RCU_MSG.C2ACTIVATE` bit is used by tools for activation of Core 2.	
19 (R/W)	C1ACTIVATE	Core 1 Activated. The `RCU_MSG.C1ACTIVATE` bit is used by tools for activation of Core 1.	
18 (R/W)	C2L1INIT	Core 2 L1 Initialized. The `RCU_MSG.C2L1INIT` bit indicates that the core 2 L1 resource is initialized.	
17 (R/W)	C1L1INIT	Core 1 L1 Initialized. The `RCU_MSG.C1L1INIT` bit indicates that the core 1 L1 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
16 (R/W)	C0L1INIT	Core 0 L1 Initialized. The `RCU_MSG.C0L1INIT` bit indicates that the core 0 L1 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
10 (R/W)	C2IDLE	Core 2 Idle. The `RCU_MSG.C2IDLE` bit indicates that core 2 is in a safe idle state in ROM.	
9 (R/W)	C1IDLE	Core 1 Idle. The `RCU_MSG.C1IDLE` bit indicates that core 1 is in a safe idle state in ROM.	
8 (R/W)	C0IDLE	Core 0 Idle. The `RCU_MSG.C0IDLE` bit indicates that core 0 is in a safe idle state in ROM.	

Table 6-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	ERRCODE	ROM Error Code. The `RCU_MSG.ERRCODE` bit indicates the error code of the ROM. It is valid only when in the error handler.

Message Clear Bits Register

The `RCU_MSG_CLR` register is used to clear bits in `RCU_MSG` register. Reading this register returns 0x00000000.

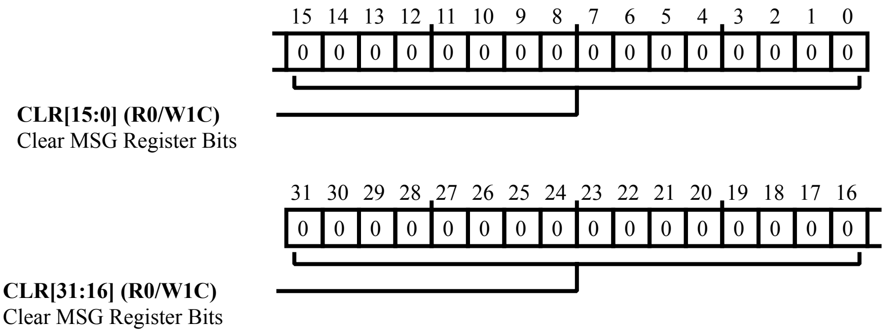

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLR[15:0] (R0/W1C)
Clear MSG Register Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLR[31:16] (R0/W1C)
Clear MSG Register Bits

Figure 6-6: RCU_MSG_CLR Register Diagram

Table 6-11: RCU_MSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1C)	CLR	Clear MSG Register Bits. The `RCU_MSG_CLR.CLR` bit resets MSG bit n.

Message Set Bits Register

The `RCU_MSG_SET` register is used to set bits in `RCU_MSG` register. Reading this register returns 0x00000000.

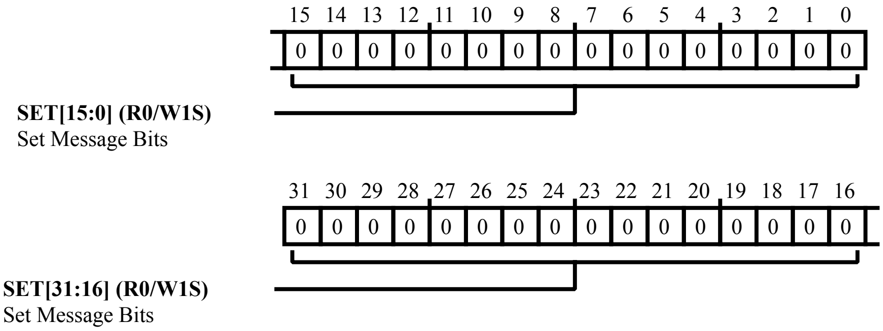

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SET[15:0] (R0/W1S)
Set Message Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SET[31:16] (R0/W1S)
Set Message Bits

Figure 6-7: RCU_MSG_SET Register Diagram

Table 6-12: RCU_MSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1S)	SET	Set Message Bits. The `RCU_MSG_SET.SET` bit sets MSG bit n.

System Interface Disable Register

The RCU system interface disable register (`RCU_SIDIS`) lets the RCU assert a system interface disable request to functional units in the processor. This register is set to its default values by a hard reset or any system reset event. For information on mapping between `RCU_SIDIS` bits and functional units, see the RCU functional description.

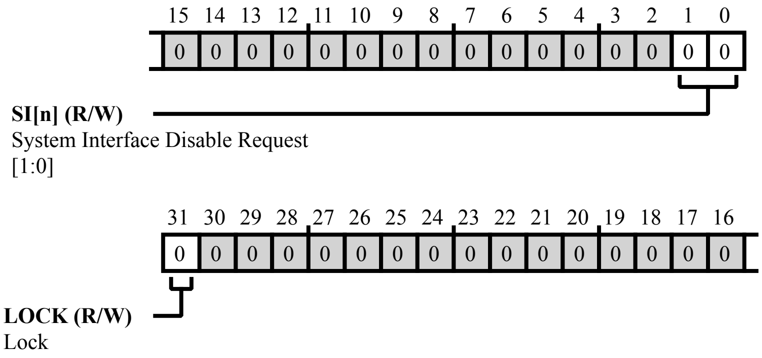

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SI[n] (R/W)
System Interface Disable Request [1:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 6-8: RCU_SIDIS Register Diagram

Table 6-13: RCU_SIDIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_SIDIS.LOCK` bit is set, the `RCU_SIDIS` register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	SI[n]	System Interface Disable Request [1:0]. Each `RCU_SIDIS.SI[n]` bit corresponds to a functional unit in the processor that supports the system interface disable request-acknowledge protocol.	
		0	RCU_SI_DISABLE_REQ[1:0] deasserted
		3	RCU_SI_DISABLE_REQ[1:0] asserted

System Interface Status Register

The RCU system interface status register (`RCU_SISTAT`) indicates whether a functional unit has or has not acknowledged an RCU unit disable request.

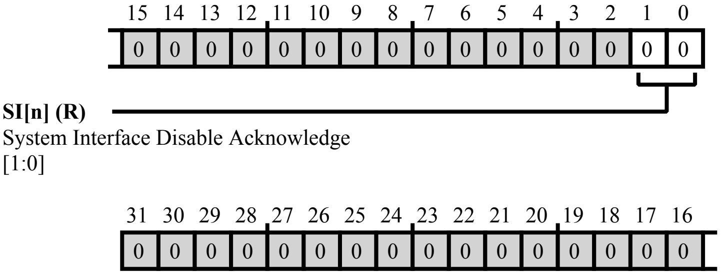

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SI[n] (R)
System Interface Disable Acknowledge
[1:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 6-9: RCU_SISTAT Register Diagram

Table 6-14: RCU_SISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	SI[n]	System Interface Disable Acknowledge [1:0]. The `RCU_SISTAT.SI[n]` bit indicates whether a functional unit has or has not acknowledged an RCU unit disable request.	
		0	No Acknowledge
		3	SI_DISABLE_ACK[1:0] asserted

Status Register

The RCU status register (`RCU_STAT`) contains status bits for all RCU reset sources, reset status, and boot mode inputs. Status bits for reset sources are sticky and can cleared by software. Error status bits are cleared by any reset event.

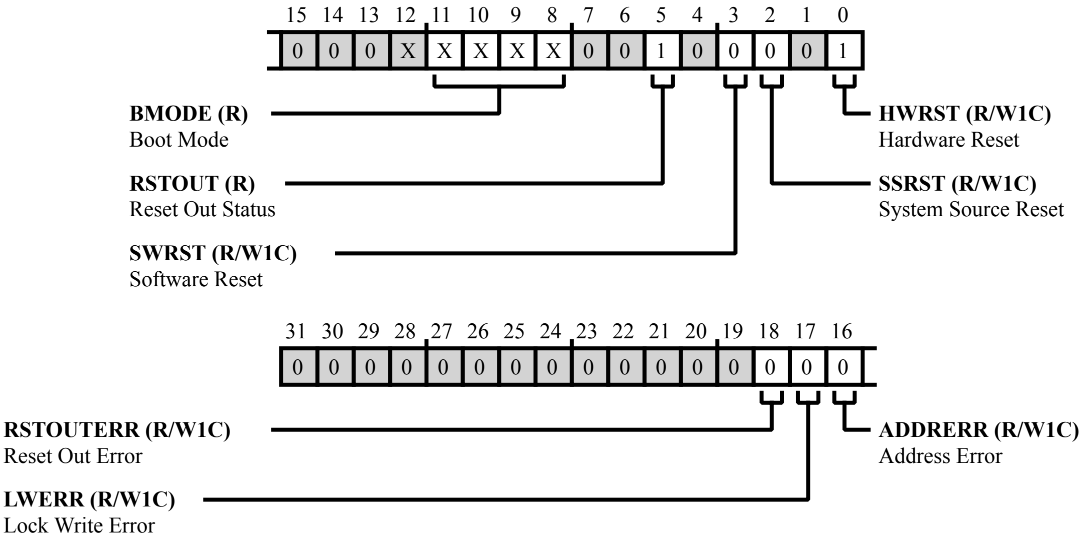

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 X X X X X 0 0 1 0 0 0 0 1
BMODE (R)
Boot Mode
RSTOUT (R)
Reset Out Status
SWRST (R/W1C)
Software Reset
HWRST (R/W1C)
Hardware Reset
SSRST (R/W1C)
System Source Reset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RSTOUTERR (R/W1C)
Reset Out Error
LWERR (R/W1C)
Lock Write Error
ADDRERR (R/W1C)
Address Error

Figure 6-10: RCU_STAT Register Diagram

Table 6-15: RCU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	RSTOUTERR	Reset Out Error. The `RCU_STAT.RSTOUTERR` bit indicates (if set) that a write attempted to set the `RCU_CTL.RSTOUTASRT` and `RCU_CTL.RSTOUTDSRT` simultaneously. This condition triggers a bus error.	
		0	No Error
		1	Error Occurred
17 (R/W1C)	LWERR	Lock Write Error. The `RCU_STAT.LWERR` bit indicates (when set) there was an attempted write to an RCU register while the `RCU_CTL.LOCK` bit was set and the global lock bit is enabled (`SPU_CTL.GLCK` bit =1). This status bit is sticky; write-1-to-clear	
		0	No Error
		1	Error Occurred

Table 6-15: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ADDRERR	Address Error. The `RCU_STAT.ADDRERR` bit indicates that the RCU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
11:8 (R/NW)	BMODE	Boot Mode. The `RCU_STAT.BMODE` bits indicate the input on the boot mode pins.	
5 (R/NW)	RSTOUT	Reset Out Status. The `RCU_STAT.RSTOUT` bit indicates the assertion status of the system reset pin.	
		0	RSTOUT Deasserted
		1	RSTOUT Asserted
3 (R/W1C)	SWRST	Software Reset. The `RCU_STAT.SWRST` bit indicates that a system reset (which was triggered by software) has occurred since the last time a hardware reset occurred or since the `RCU_STAT.SWRST` bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
2 (R/W1C)	SSRST	System Source Reset. The `RCU_STAT.SSRST` bit indicates that a system reset triggered by hardware in the system clock domain, clock A domain, or clock B domain has occurred since the last time a hardware reset occurred or since the `RCU_STAT.SSRST` bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
0 (R/W1C)	HWRST	Hardware Reset. The `RCU_STAT.HWRST` bit indicates that a hardware reset has occurred.	
		0	Inactive
		1	Reset Occurred

Software Vector Register 0

The `RCU_SVECT0` register contains the default location of the first instruction to execute after a reset.

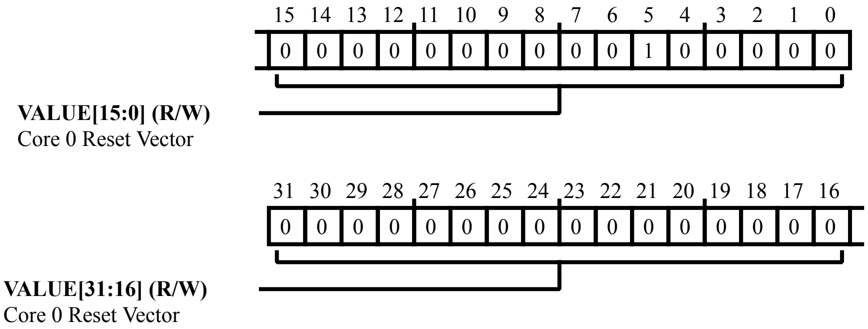

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
VALUE[15:0] (R/W)
Core 0 Reset Vector
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Core 0 Reset Vector

Figure 6-11: RCU_SVECT0 Register Diagram

Table 6-16: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Core 0 Reset Vector. The `RCU_SVECT0.VALUE` bit field contains the default location of the first instruction to execute after a reset.

Software Vector Register 1

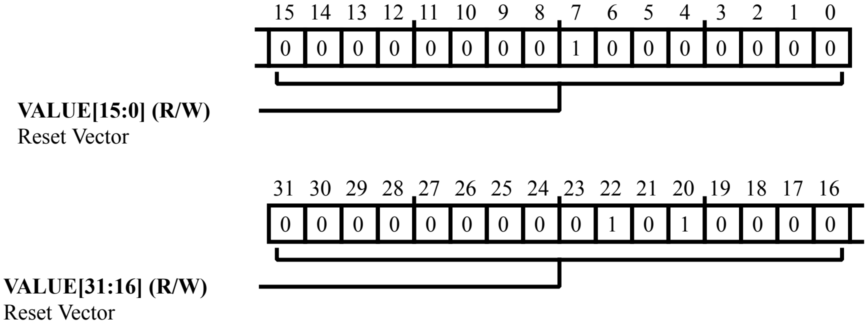

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Reset Vector
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
VALUE[31:16] (R/W)
Reset Vector

Figure 6-12: RCU_SVECT1 Register Diagram

Table 6-17: RCU_SVECT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

Software Vector Register 2

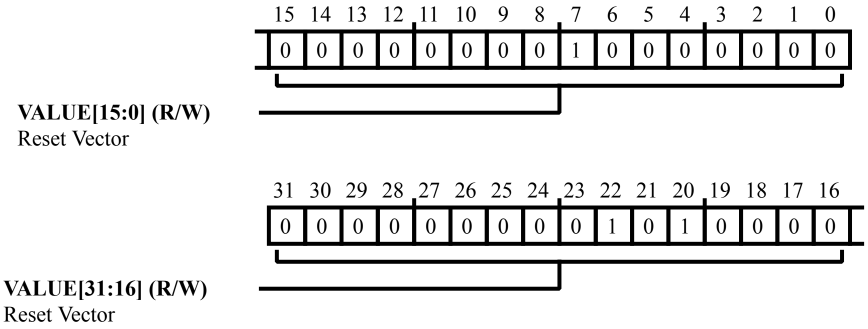

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Reset Vector
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
VALUE[31:16] (R/W)
Reset Vector

Figure 6-13: RCU_SVECT2 Register Diagram

Table 6-18: RCU_SVECT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

SVECT Lock Register

The RCU software vector lock register (RCU_SVECT_LCK) provides a register lock and software vector n enable bits for each processor core on the product. This register is set to its default values by a hard reset or any system reset event.

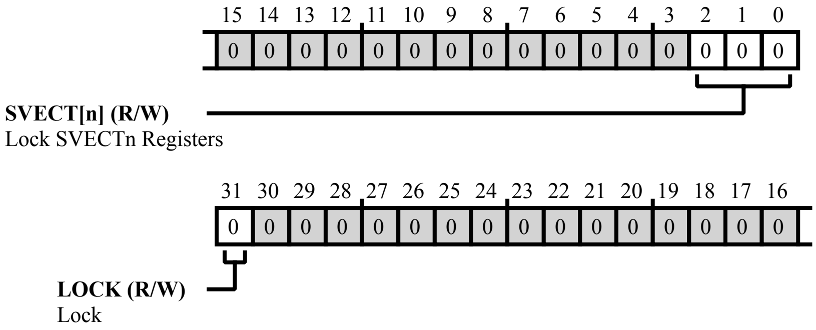

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SVECT[n] (R/W)
Lock SVECTn Registers
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 6-14: RCU_SVECT_LCK Register Diagram

Table 6-19: RCU_SVECT_LCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_SVECT_LCK.LOCK` bit is set, the `RCU_SVECT_LCK` register is read only (locked).	
		0	Unlock
		1	Lock
2:0 (R/W)	SVECT[n]	Lock SVECTn Registers. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `RCU_SVECT_LCK.SVECT[n]` bit is set, the `RCU_SVECT0` and `RCU_SVECT1` registers are read only (locked).	

7 System Event Controller (SEC) and Generic Interrupt Controller (GIC)

There are two interrupt controllers—a generic interrupt controller (GIC) for the ARM core and the system event controller (SEC) for the SHARC cores.

System event management is the responsibility of the system event controller (SEC). The SEC manages the configuration of all system event sources. The SEC also manages the propagation of system events to all connected cores and the system fault interface.

All of the peripheral interrupts are routed using a single SEC interrupt to the desired core. The SEC allows programmability of the peripheral interrupt's priority, supporting up to 256 priority levels that are arbitrated within the SEC itself. The SEC also allows these interrupts to be grouped and masked by priority level and provides the flexibility to choose which core(s) the interrupt is routed to.

The SEC also supports self-nesting of interrupts, which is required when sharing a single interrupt request to an individual core, as this allows for a higher-priority peripheral interrupt to be passed to the core while it is currently servicing a lower-priority peripheral interrupt. For more details please refer to "Self-Nesting Mode for System Event Controller Interrupt (SECI)" in the *SHARC+ Core Programming Reference*.

For more information about the ARM GIC, visit the ARM Information Center.

SEC Features

The following list describes the system event controller features.

- Comprehensive system event source management including interrupt enable, fault enable, priority, core mapping, and source grouping.
- Fault management including fault action configuration, timeout, external indication, and system reset.
- Determinism where all system events have the same propagation delay and provide unique identification of a specific system event source.
- Distributed programming model where each system event source control and all status fields are independent of all others.

- Slave control port which provides access to all SEC registers for configuration, status, and interrupt or fault service model.
- Global locking supports a register level protection model to prevent writes to “locked” registers.

SEC Functional Description

The following sections provide a functional description of the SEC.

The *SEC/GIC Interrupt Signal Flow* figure shows an overview of the interrupt systems.

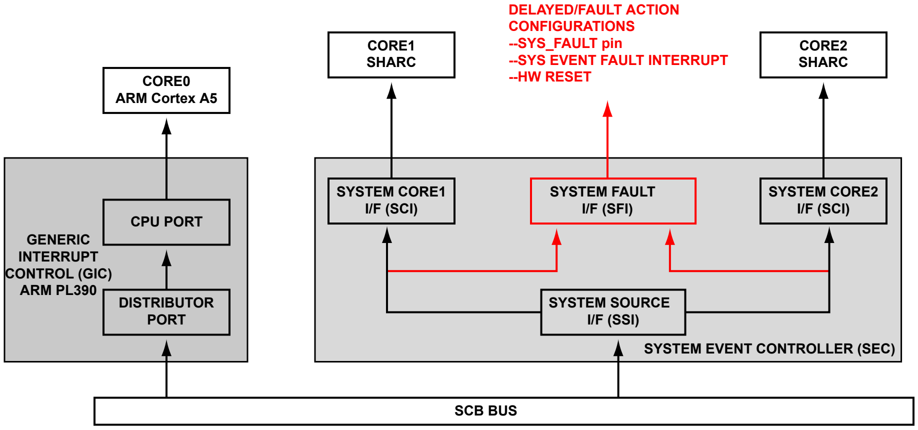

DELAYED/FAULT ACTION
CONFIGURATIONS
--SYS_FAULT pin
--SYS EVENT FAULT INTERRUPT
--HW RESET
CORE1
SHARC
CORE2
SHARC
CORE0
ARM Cortex A5
CPU PORT
GENERIC
INTERRUPT
CONTROL (GIC)
ARM PL390
DISTRIBUTOR
PORT
SYSTEM CORE1
I/F (SCI)
SYSTEM FAULT
I/F (SFI)
SYSTEM CORE2
I/F (SCI)
SYSTEM SOURCE
I/F (SSI)
SYSTEM EVENT CONTROLLER (SEC)
SCB BUS

Figure 7-1: SEC/GIC Interrupt Signal Flow

ADSP-SC58x SEC Register List

The System Event Controller (SEC) manages the system fault sources, including control features such as enable/ disable, priority, and active/pending source status. For more information on SEC functionality, see the SEC register descriptions.

Table 7-1: ADSP-SC58x SEC Register List

Name	Description
`SEC_CACT[n]`	SCI Active Register n
`SEC_CCTL[n]`	SCI Control Register n
`SEC_CGMSK[n]`	SCI Group Mask Register n
`SEC_CPLVL[n]`	SCI Priority Level Register n
`SEC_CPMSK[n]`	SCI Priority Mask Register n
`SEC_CPND[n]`	Core Pending Register n
`SEC_CSID[n]`	SCI Source ID Register n
`SEC_CSTAT[n]`	SCI Status Register n
`SEC_END`	Global End Register

Table 7-1: ADSP-SC58x SEC Register List (Continued)

Name	Description
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

ADSP-SC58x SEC Interrupt List

Table 7-2: ADSP-SC58x SEC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	SEC0_ERR	SEC0 Error	Level	

Combined SEC and GIC Interrupt List

The *Combined SEC and GIC Interrupt List* table provides a complete list of the processor and Cortex interrupts. Note that the DAI has its own system interrupt controllers. For more information see the DAI System Interrupt Controller (SIC). Note in the table below many interrupts are supported by all cores. In the cases were an interrupt is not supported its ID number (GIC_ID for ARM Cortex-A5 or SEC_ID for SHARC+) is reserved.

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List*

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
GIC	SW Interrupt 0, core reset, core ID = 0	N/A	0	GIC_SOFT00
GIC	SW Interrupt 1, undefined instruction, core ID = 0	N/A	1	GIC_SOFT01

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
GIC	SW Interrupt 2, supervisor call, core ID = 0	N/A	2	GIC_SOFT02
GIC	SW Interrupt 3, prefetch call, core ID = 0	N/A	3	GIC_SOFT03
GIC	SW Interrupt 4, data abort, core ID = 0	N/A	4	GIC_SOFT04
GIC	SW Interrupt 5, reserved, core ID = 0	N/A	N/A	GIC_SOFT05
GIC	SW Interrupt 6, IRQ interrupt, core ID = 0	N/A	6	GIC_SOFT06
GIC	SW Interrupt 7, FIQ interrupt, core ID = 0	N/A	7	GIC_SOFT07
		N/A	8 – 31 Reserved	
SEC	SEC0 Error	0	32	SEC0_ERR
CGU	CGU0 Event	1	33	CGU0_EVT
CGU	CGU1 Event	2	34	CGU1_EVT
WDOG	WDOG0 Expiration	3	35	WDOG0_EXP
WDOG	WDOG1 Expiration	4	36	WDOG1_EXP
OTPC	OTPC0 Dual bit Error	5	37	OTPC0_ERR
TMU	TMU0 Fault	6	38	TMU0_FAULT
TAPC	Test/User Key Fail	7	39	TAPC0_KEYFAIL
L2CTL	L2CTL0 ECC Error	8	40	L2CTL0_ECC_ERR
		9 – 13 Reserved	41 – 45 Reserved	
SMPU	Core Data Read interrupt, core ID = 1	14	46	C1_IRQ0
SMPU	Core Data Write interrupt, core ID = 1	15	47	C1_IRQ1
SMPU	Core Instruction Fetch interrupt, core ID = 1	16	48	C1_IRQ2
SMPU	Core Idle instruction interrupt, core ID = 1	17	49	C1_IDLE
SMPU	Core Data Read interrupt, core ID = 2	18	50	C2_IRQ0
SMPU	Core Data Write interrupt, core ID = 2	19	51	C2_IRQ1
SMPU	Core Instruction Fetch interrupt, core ID = 2	20	52	C2_IRQ2
SMPU	Core Idle instruction interrupt, core ID = 2	21	53	C2_IDLE
CORE	L2 Cache interrupt, core ID = 0	22	54	C0_L2CC
CORE	L1 Parity interrupt, core ID = 0	23	55	C0_L1_PERR
DAI	DAI0 High Priority	24	56	DAI0_IRQH
DAI	DAI1 High Priority	25	57	DAI1_IRQH

Table 7-3: ADSP-SC58x ***Combined SEC and GIC Interrupt List*** (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
GPTIMER	GP TIMER0 Timer 0	26	58	TIMER0_TMR0
GPTIMER	GP TIMER0 Timer 1	27	59	TIMER0_TMR1
GPTIMER	GP TIMER0 Timer 2	28	60	TIMER0_TMR2
GPTIMER	GP TIMER0 Timer 3	29	61	TIMER0_TMR3
EPWM	EPWM0 PWMTMR Grouped	30	62	PWM0_SYNC
EPWM	EPWM0 Trip	31	63	PWM0_TRIP
EPWM	EPWM1 PWMTMR Grouped	32	64	PWM1_SYNC
EPWM	EPWM1 Trip	33	65	PWM1_TRIP
EPWM	EPWM2 PWMTMR Grouped	34	66	PWM2_SYNC
EPWM	EPWM2 Trip	35	67	PWM2_TRIP
ACM	ACM0 Event Miss	36	68	ACM0_EVT_MISS
ACM	ACM0 Event Complete	37	69	ACM0_EVT_COMPLETE
PINT	PINT0 Pin Interrupt Block	38	70	PINT0_BLOCK
PINT	PINT1 Pin Interrupt Block	39	71	PINT1_BLOCK
PINT	PINT2 Pin Interrupt Block	40	72	PINT2_BLOCK
PINT	PINT3 Pin Interrupt Block	41	73	PINT3_BLOCK
PINT	PINT4 Pin Interrupt Block	42	74	PINT4_BLOCK
PINT	PINT5 Pin Interrupt Block	43	75	PINT5_BLOCK
SEC	Software Interrupt 0	44	Reserved	SOFT0
SEC	Software Interrupt 1	45	Reserved	SOFT1
SEC	Software Interrupt 2	46	Reserved	SOFT2
SEC	Software Interrupt 3	47	Reserved	SOFT3
SEC	Software Interrupt 4	48	Reserved	SOFT4
SEC	Software Interrupt 5	49	Reserved	SOFT5
SEC	Software Interrupt 6	50	Reserved	SOFT6
SEC	Software Interrupt 7	51	Reserved	SOFT7
SPORT	SPORT0 Channel A DMA	52	84	SPORT0_A_DMA
SPORT	SPORT0 Channel A Status	53	85	SPORT0_A_STAT
SPORT	SPORT0 Channel B DMA	54	86	SPORT0_B_DMA
SPORT	SPORT0 Channel B Status	55	87	SPORT0_B_STAT
SPORT	SPORT1 Channel A DMA	56	88	SPORT1_A_DMA

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
SPORT	SPORT1 Channel A Status	57	89	SPORT1_A_STAT
SPORT	SPORT1 Channel B DMA	58	90	SPORT1_B_DMA
SPORT	SPORT1 Channel B Status	59	91	SPORT1_B_STAT
SPORT	SPORT4 Channel A DMA	60	92	SPORT4_A_DMA
SPORT	SPORT4 Channel A Status	61	93	SPORT4_A_STAT
SPORT	SPORT4 Channel B DMA	62	94	SPORT4_B_DMA
SPORT	SPORT4 Channel B Status	63	95	SPORT4_B_STAT
SPORT	SPORT5 Channel A DMA	64	96	SPORT5_A_DMA
SPORT	SPORT5 Channel A Status	65	97	SPORT5_A_STAT
SPORT	SPORT5 Channel B DMA	66	98	SPORT5_B_DMA
SPORT	SPORT5 Channel B Status	67	99	SPORT5_B_STAT
SPI	SPI2 TX DMA Channel	68	100	SPI2_TXDMA
SPI	SPI2 RX DMA Channel	69	101	SPI2_RXDMA
SPI	SPI2 Status	70	102	SPI2_STAT
SPI	SPI2 Error	71	103	SPI2_ERR
GPTIMER	GP TIMER0 Timer 4	72	104	GPTIMER0_TMR4
GPTIMER	GP TIMER0 Timer 5	73	105	GPTIMER0_TMR5
GPTIMER	GP TIMER0 Timer 6	74	106	GPTIMER0_TMR6
GPTIMER	GP TIMER0 Timer 7	75	107	GPTIMER0_TMR7
GPTIMER	GP TIMER0 Status	76	108	GPTIMER0_STAT
LP	LP0 DMA Data	77	109	LP0_DMA
LP	LP0 Status	78	110	LP0_STAT
LP	LP1 DMA Data	79	111	LP1_DMA
LP	LP1 Status	80	112	LP1_STAT
EPPI	EPPI0 DMA Channel 0	81	113	EPPI0_CH0_DMA
EPPI	EPPI0 DMA Channel 1	82	114	EPPI0_CH1_DMA
EPPI	EPPI0 Status	83	115	EPPI0_STAT
CAN	CAN0 Receive	84	116	CAN0_RX
CAN	CAN0 Transmit	85	117	CAN0_TX
CAN	CAN0 Status	86	118	CAN0_STAT
CAN	CAN1 Receive	87	119	CAN1_RX

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
CAN	CAN1 Transmit	88	120	CAN1_TX
CAN	CAN1 Status	89	121	CAN1_STAT
SPORT	SPORT2 Channel A DMA	90	122	SPORT2_A_DMA
SPORT	SPORT2 Channel A Status	91	123	SPORT2_A_STAT
SPORT	SPORT2 Channel B DMA	92	124	SPORT2_B_DMA
SPORT	SPORT2 Channel B Status	93	125	SPORT2_B_STAT
SPORT	SPORT3 Channel A DMA	94	126	SPORT3_A_DMA
SPORT	SPORT3 Channel A Status	95	127	SPORT3_A_STAT
SPORT	SPORT3 Channel B DMA	96	128	SPORT3_B_DMA
SPORT	SPORT3 Channel B Status	97	129	SPORT3_B_STAT
SPORT	SPORT6 Channel A DMA	98	130	SPORT6_A_DMA
SPORT	SPORT6 Channel A Status	99	131	SPORT6_A_STAT
SPORT	SPORT6 Channel B DMA	100	132	SPORT6_B_DMA
SPORT	SPORT6 Channel B Status	101	133	SPORT6_B_STAT
SPORT	SPORT7 Channel A DMA	102	134	SPORT7_A_DMA
SPORT	SPORT7 Channel A Status	103	135	SPORT7_A_STAT
SPORT	SPORT7 Channel B DMA	104	136	SPORT7_B_DMA
SPORT	SPORT7 Channel B Status	105	137	SPORT7_B_STAT
SPI	SPI0 TX DMA Channel	106	138	SPI0_TXDMA
SPI	SPI0 RX DMA Channel	107	139	SPI0_RXDMA
SPI	SPI0 Status	108	140	SPI0_STAT
SPI	SPI0 Error	109	141	SPI0_ERR
SPI	SPI1 TX DMA Channel	110	142	SPI1_TXDMA
SPI	SPI1 RX DMA Channel	111	143	SPI1_RXDMA
SPI	SPI1 Status	112	144	SPI1_STAT
SPI	SPI1 Error	113	145	SPI1_ERR
UART	UART0 Transmit DMA	114	146	UART0_TXDMA
UART	UART0 Receive DMA	115	147	UART0_RXDMA
UART	UART0 Status	116	148	UART0_STAT
UART	UART1 Transmit DMA	117	149	UART1_TXDMA
UART	UART1 Receive DMA	118	150	UART1_RXDMA

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
UART	UART1 Status	119	151	UART1_STAT
UART	UART2 Transmit DMA	120	152	UART2_TXDMA
UART	UART2 Receive DMA	121	153	UART2_RXDMA
UART	UART2 Status	122	154	UART2_STAT
TWI	TWI0 Data	123	155	TWI0_DATA
TWI	TWI1 Data	124	156	TWI1_DATA
TWI	TWI2 Data	125	157	TWI2_DATA
CNT	CNT0 Status	126	158	CNT0_STAT
CTI	CTI Event1, Core ID = 1	127	Reserved	ECT_C1_EVT
CTI	CTI Event2, Core ID = 2	128	Reserved	ECT_C2_EVT
PKIC	Public Key Interrupt (PKA, TRNG, SL)	129	161	PKIC0_IRQ
PKTE	Security Packet Engine	130	162	PKTE0_IRQ
MSI	MSI0 Status	131	163	MSI0_STAT
USB	USB0 Status/FIFO Data Ready	132	164	USB0_STAT
USB	USB0 DMA Status/Transfer Complete	133	165	USB0_DATA
USB	USB1 Status/FIFO Data Ready	134	166	USB1_STAT
USB	USB1 DMA Status/Transfer Complete	135	167	USB1_DATA
TRU	TRU0 Interrupt 4, core ID = 1	136	Reserved	TRU0_INT4
TRU	TRU0 Interrupt 5, core ID = 1	137	Reserved	TRU0_INT5
TRU	TRU0 Interrupt 6, core ID = 1	138	Reserved	TRU0_INT6
TRU	TRU0 Interrupt 7, core ID = 1	139	Reserved	TRU0_INT7
TRU	TRU0 Interrupt 8, core ID = 2	140	Reserved	TRU0_INT8
TRU	TRU0 Interrupt 9, core ID = 2	141	Reserved	TRU0_INT9
TRU	TRU0 Interrupt 10, core ID = 2	142	Reserved	TRU0_INT10
TRU	TRU0 Interrupt 11, core ID = 2	143	Reserved	TRU0_INT11
SINC	SINC0 Status	144	176	SINC0_STAT
DAI	DAI0 Low Priority	145	177	DAI0_IRQL
DAI	DAI1 Low Priority	146	178	DAI1_IRQL
EMAC	EMAC0 Status	148	180	EMAC0_STAT
EMAC	EMAC1 Status	149	181	EMAC1_STAT
FFTA	FFTA0 Transmit DMA	150	182	FFTA0_TXDMA

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
FFTA	FFTA0 Receive DMA	151	183	FFTA0_RXDMA
FFTA	FFTA0 Status	152	184	FFTA0_STAT
FIR	FIR0 DMA	153	185	FIR0_DMA
FIR	FIR0 Status	154	186	FIR0_STAT
IIR	IIR0 DMA	155	187	IIR0_DMA
IIR	IIR0 Status	156	188	IIR0_STAT
HADC	HADC0	157	189	HADC0_EVT
HAE	HAE0 RX DMA Channel 0	158	190	HAE0_RXDMA_CH0
HAE	HAE0 RX DMA Channel 1	159	191	HAE0_RXDMA_CH1
HAE	HAE0 TX DMA Channel 0	160	192	HAE0_TXDMA
HAE	HAE0 Status	161	193	HAE0_STAT
MLB	MLB0 Interrupt Channel 0 through 31	162	194	MLB0_INT0
MLB	MLB0 Interrupt Channel 32 thorugh 63	163	195	MLB0_INT1
MLB	MLB0 Status	164	196	MLB0_STAT
RTC	RTC Event	165	197	RTC0_EVT
MDMA	Maximum BW DMA Channel 0	166	198	MDMA3_SRC
MDMA	Maximum BW DMA Channel 1	167	199	MDMA3_DST
MDMA	Enhanced BW DMA Channel 0	168	200	MDMA2_SRC
MDMA	Enhanced BW DMA Channel 1	169	201	MDMA2_DST
EMDMA	EMDMA0 DMA Done	170	202	EMDMA0_DONE
EMDMA	EMDMA1 DMA Done	171	203	EMDMA1_DONE
CRC	Standard BW DMA/CRC0 Channel 0	172	204	MDMA0_SRC
CRC	Standard BW DMA/CRC0 Channel 1	173	205	MDMA0_DST
CRC	Standard BW DMA/CRC1 Channel 0	174	206	MDMA1_SRC
CRC	Standard BW DMA/CRC1 Channel 1	175	207	MDMA1_DST
CRC	CRC0 Datacount expiration	176	208	CRC0_DCNTEXP
CRC	CRC1 Datacount expiration	177	209	CRC1_DCNTEXP
CRC	CRC0 Error	178	210	CRC0_ERR
CRC	CRC1 Error	179	211	CRC1_ERR
SPORT	SPORT0 Channel A DMA Error	180	212	SPORT0_A_DMA_ERR
SPORT	SPORT0 Channel B DMA Error	181	213	SPORT0_B_DMA_ERR

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
SPORT	SPORT1 Channel A DMA Error	182	214	SPORT1_A_DMA_ERR
SPORT	SPORT1 Channel B DMA Error	183	215	SPORT1_B_DMA_ERR
SPORT	SPORT4 Channel A DMA Error	184	216	SPORT4_A_DMA_ERR
SPORT	SPORT4 Channel B DMA Error	185	217	SPORT4_B_DMA_ERR
SPORT	SPORT5 Channel A DMA Error	186	218	SPORT5_A_DMA_ERR
SPORT	SPORT5 Channel B DMA Error	187	219	SPORT5_B_DMA_ERR
SPI	SPI2 TX DMA Channel Error	188	220	SPI2_TXDMA_ERR
SPI	SPI2 RX DMA Channel Error	189	221	SPI2_RXDMA_ERR
SPORT	SPORT2 Channel A DMA Error	190	222	SPORT2_A_DMA_ERR
SPORT	SPORT2 Channel B DMA Error	191	223	SPORT2_B_DMA_ERR
SPORT	SPORT3 Channel A DMA Error	192	224	SPORT3_A_DMA_ERR
SPORT	SPORT3 Channel B DMA Error	193	225	SPORT3_B_DMA_ERR
SPORT	SPORT6 Channel A DMA Error	194	226	SPORT6_A_DMA_ERR
SPORT	SPORT6 Channel B DMA Error	195	227	SPORT6_B_DMA_ERR
SPORT	SPORT7 Channel A DMA Error	196	228	SPORT7_A_DMA_ERR
SPORT	SPORT7 Channel B DMA Error	197	229	SPORT7_B_DMA_ERR
SPI	SPI0 TX DMA Channel Error	198	230	SPI0_TXDMA_ERR
SPI	SPI0 RX DMA Channel Error	199	231	SPI0_RXDMA_ERR
SPI	SPI1 TX DMA Channel Error	200	232	SPI1_TXDMA_ERR
SPI	SPI1 RX DMA Channel Error	201	233	SPI1_RXDMA_ERR
UART	UART0 Transmit DMA Error	202	234	UART0_TXDMA_ERR
UART	UART0 Receive DMA Error	203	235	UART0_RXDMA_ERR
UART	UART1 Transmit DMA Error	204	236	UART1_TXDMA_ERR
UART	UART1 Receive DMA Error	205	237	UART1_RXDMA_ERR
UART	UART2 Transmit DMA Error	206	238	UART2_TXDMA_ERR
UART	UART2 Receive DMA Error	207	239	UART2_RXDMA_ERR
LP	LP0 DMA Data Error	208	240	LP0_DMA_ERR
LP	LP1 DMA Data Error	209	241	LP1_DMA_ERR
FFTA	FFTA0 Transmit DMA Error	210	242	FFT0_TXDMA_ERR
FFTA	FFTA0 Receive DMA Error	211	243	FFT0_RXDMA_ERR
HAE	HAE0 RX DMA Channel 0 Error	212	244	HAE0_RXDMA_CH0_ERR

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
HAE	HAE0 RX DMA Channel 1 Error	213	245	HAE0_RXDMA_CH1_ERR
HAE	HAE0 TX DMA Channel 0 Error	214	246	HAE0_TXDMA_ERR
EPPI	EPPI0 DMA Channel 0 Error	215	247	EPPI0_CH0_DMA_ERR
EPPI	EPPI0 DMA Channel 1 Error	216	248	EPPI0_CH1_DMA_ERR
MDMA	Standard BW Source 0 DMA Channel Error	217	249	MDMA0_SRC_ERR
MDMA	Standard BW Dest 0 DMA Channel Error	218	250	MDMA0_DST_ERR
MDMA	Standard BW Source 1 DMA Channel Error	219	251	MDMA1_SRC_ERR
MDMA	Standard BW Dest 1 DMA Channel Error	220	252	MDMA1_DST_ERR
MDMA	Enhanced BW DMA Channel 0 Error	221	253	MDMA2_SRC_ERR
MDMA	Enhanced BW DMA Channel 1 Error	222	254	MDMA2_DST_ERR
MDMA	Maximum BW DMA Channel 0 Error	223	255	MDMA3_SRC_ERR
MDMA	Maximum BW DMA Channel 1 Error	224	256	MDMA3_DST_ERR
SWU	SWU0 Event SMC	225	257	SWU0_EVT
SWU	SWU1 Event, L2 Memory DMA Port 0	226	258	SWU1_EVT
SWU	SWU2 Event, L2 Memory Core Port 0	227	259	SWU2_EVT
SWU	SWU3 Event, L2 Memory DMA Port 1	228	260	SWU3_EVT
SWU	SWU4 Event, L2 Memory Core Port 1	229	261	SWU4_EVT
SWU	SWU3 Event, L2 Memory DMA Port 2	230	262	SWU5_EVT
SWU	SWU4 Event, L2 Memory Core Port 2	231	263	SWU6_EVT
SWU	SWU7 Event, Core ID = 1 Slave Port 1	232	264	SWU7_EVT
SWU	SWU8 Event, Core ID = 1 Slave Port 2	233	265	SWU8_EVT
SWU	SWU9 Event, Core ID = 2 Slave Port 1	234	266	SWU9_EVT
SWU	SWU10 Event, Core ID = 2 Slave Port 2	235	267	SWU10_EVT
SWU	SWU11 Event SMMR	236	268	SWU11_EVT
SWU	SWU12 Event SPI L3 Memory	237	269	SWU12_EVT
SWU	SWU13 Event DMC0_A L3 Memory	238	270	SWU13_EVT
SWU	SWU14 Event DMC0_B L3 Memory	239	271	SWU14_EVT
SWU	SWU15 Event PCIe L3 Memory	240	272	SWU15_EVT
SPU	SPU0 Event	241	273	SPU0_INT
SMPU	SMPU Aggregated Event	242	274	SMPU0_AGGR_INT
PCIE	PCIe Reset Request	243	275	PCIE0_RESET

Table 7-3: ADSP-SC58x *Combined SEC and GIC Interrupt List* (Continued)

Module	Event/Interrupt	SEC ID	GIC ID	SEC/GIC Interrupt Name
PCIE	PCIe Status	244	276	PCIE0_STAT
PCIE	PCIe DMA Completion	245	277	PCIE0_DMA
		246 – 247 Reserved	278 – 279 Reserved	
TRU	TRU0 Interrupt 0, core ID = 0	Reserved	280	TRU_INT0
TRU	TRU0 Interrupt 1, core ID = 0	Reserved	281	TRU_INT1
TRU	TRU0 Interrupt 2, core ID = 0	Reserved	282	TRU_INT2
TRU	TRU0 Interrupt 3, core ID = 0	Reserved	283	TRU_INT3
CTI	CTI Event0, core ID = 0	Reserved	284	ECT_C0_EVT
PMU	Performance Monitoring Interrupt, core ID = 0	Reserved	285	C0_PMUIRQ

SEC Definitions

The event controller uses the following definitions.

System Events

System source indications including interrupts and faults.

System Source

Point of origin of system event.

SID (Identification, unique)

Source numeric identifier for each system source connected to the SEC.

SSI

SEC source interface, system event source control, and status subblock of the SEC.

SCI

SEC core interface, core interface subblock of the SEC

SPR

SEC prioritizer determines the highest priority pending interrupt and the highest priority active interrupt. The SPR provides these interrupts in the appropriate registers of the SCI for the priority and nesting model of the SCI.

SFI

SEC Fault Interface, fault management subblock of the SEC.

SEC Block Diagram

The ***SEC Block Diagram*** shows the event management architecture.

System sources connect to the SEC through the SSI. Each core has a dedicated SCI. The SFI provides fault action connections to the rest of the system.

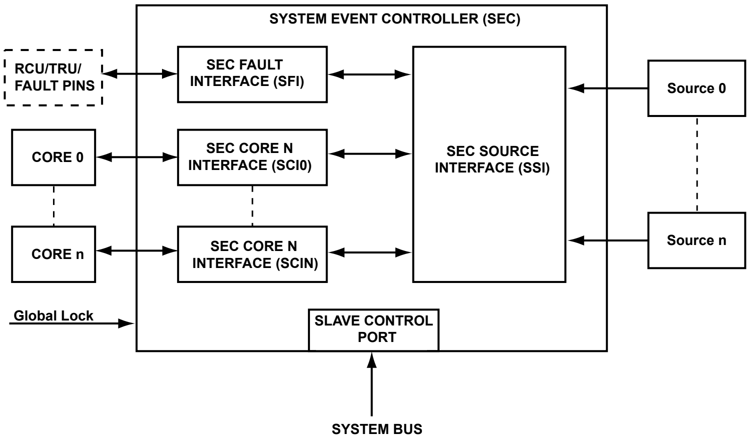

SYSTEM EVENT CONTROLLER (SEC)
RCU/TRU/ FAULT PINS
SEC FAULT INTERFACE (SFI)
CORE 0
SEC CORE N INTERFACE (SCI0)
CORE n
SEC CORE N INTERFACE (SCIN)
SEC SOURCE INTERFACE (SSI)
Source 0
Source n
Global Lock
SLAVE CONTROL PORT
SYSTEM BUS

Figure 7-2: SEC Block Diagram

SEC Fault Interface (SFI)

The SFI manages fault events and associated actions. The fault management support provided in the SEC helps satisfy the safety requirements of various applications. The SSI provides the highest priority pending source that is enabled as a fault. The SFI captures this value and enables a countdown, and once the countdown expires, takes the prescribed action.

Fault actions which can be configured, as shown in ***SFI Block Diagram***, include

- Trigger Output
- System Reset
- Fault Output
    - Computer Operating Properly (COP) mode
    - Fault Mode

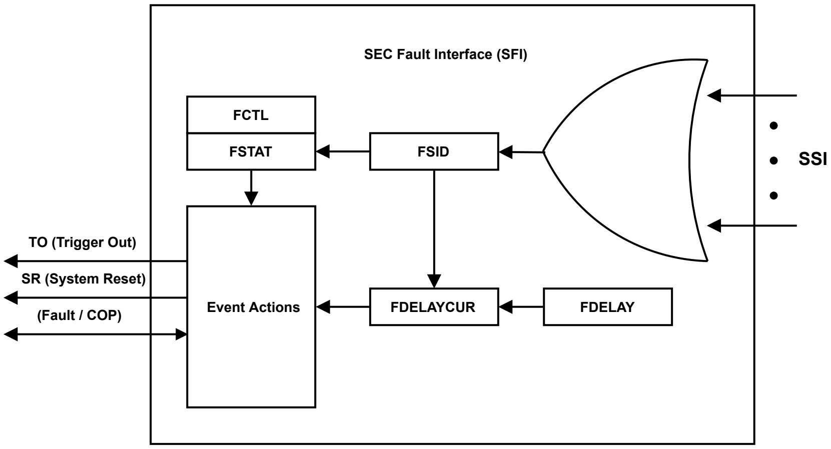


Figure 7-3: SFI Block Diagram

Fault Management

System sources can be enabled as fault sources in the `SEC_SCTL[n]` register. When a source enabled as a fault moves to pending, it is forwarded to the SFI as a fault indication. The pending bit (`SEC_FSTAT.PND`) indicates a source has signaled a fault assertion but it has not yet triggered the event actions (if delay is enabled). The SEC fault interface sets the `SEC_FSTAT.PND` bit when the fault source ID register (`SEC_FSID`) is updated on assertion of a fault source input. The system source pending triggers a fault pending and after a programmable delay the fault moves to active. Event actions then execute if appropriate action is not taken by the core. The `SEC_FSTAT.ACT` bit indicates that the SEC has received a fault source input, the delay has expired, and the fault actions are enabled.

The `SEC_FSTAT.NPND` bit indicates if one or more sources have signaled a fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the `SEC_FSTAT.NPND` bit when the fault interface detects assertion of any enabled fault source input, while either the `SEC_FSTAT.PND` or `SEC_FSTAT.ACT` bits are set. The SEC clears the `SEC_FSTAT.NPND` bit when there are no fault sources waiting.

A fault indication from an external device can also be detected on sampling the fault signals. When a fault is detected the `SEC_FSTAT.ACT` and `SEC_FSID.FEXT` bits are set. The assertion of either signal results in a fault input detection.

The `SEC_FEND` register receives a fault end indication from the core. The core writes the SID of the fault to the `SEC_FEND` register. If the SID matches the value in the `SEC_FSID` register, the `SEC_FSTAT.PND` and `SEC_FSTAT.ACT` bits are cleared.

More information can be found in the Fault Management Interface Programming Model section.

SEC Core Interface (SCI)

The SCI manages communication between the corresponding core and the SEC. The SEC prioritizer (SPR) of the SCI receives pending, active, and priority information from the SSI for each system event source assigned to this SCI. The SPR determines the highest-priority pending system event and the SCI determines whether it propagates to the core. The SCI maintains the coherency for the system event service model implemented on the connected core.

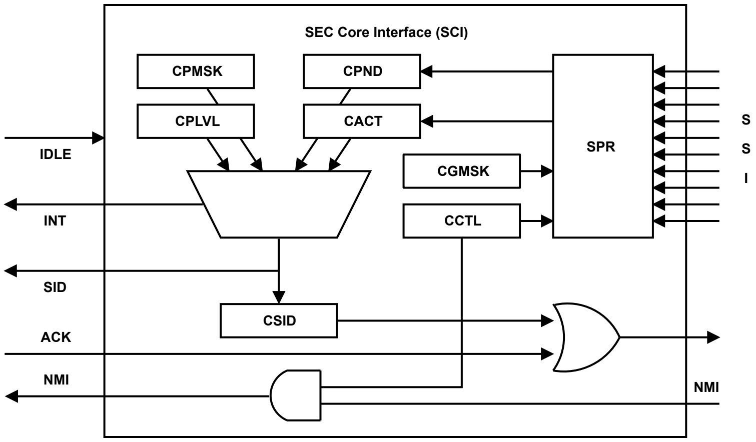

SEC Core Interface (SCI)
CPMSK
CPND
CPLVL
CACT
CGMSK
CCTL
SPR
S
S
I
IDLE
INT
SID
CSID
ACK
NMI
NMI

Figure 7-4: SCI Overview Block Diagram

SEC Source Interface (SSI)

The SSI manages all of the system event sources. It maintains the status of each source in the corresponding `SEC_SSTAT[n]` register. The corresponding `SEC_SCTL[n]` register manages the control of each source. A pending and enabled event passes its indication and priority to the SCI to which it is assigned for further processing.

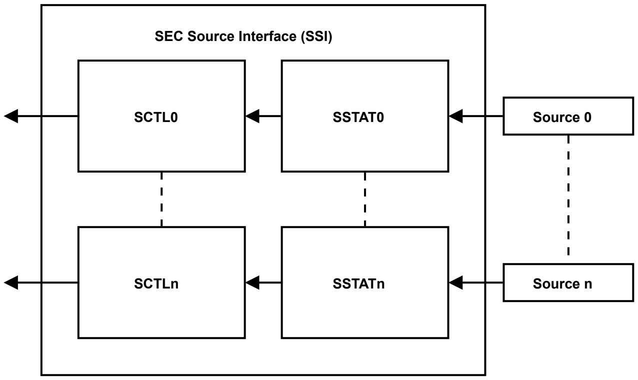

SEC Source Interface (SSI)
SCTL0
SSTAT0
Source 0
SCTLn
SSTATn
Source n

Figure 7-5: SSI Overview Block Diagram

SEC Architectural Concepts

The following sections describe SEC architectural features.

System Interrupt Acknowledge

A system interrupt acknowledge occurs when the core provides an indication that it has acquired the SID of the interrupt last issued by the SEC. The SEC core interface option allows generation by:

- A slave port write to the `SEC_CSID[n]` register.
- The assertion of an input acknowledge signal (the connected core generates the signal).

System Interrupt Groups

System sources can be assigned to groups using the `SEC_SCTL[n].GRP` bit field. Source groups allow fast context switching for system interrupts at each SCI. The `SEC_CGMSK[n]` register allows quick masking of interrupt groups of unlimited size with a single write operation.

System Interrupt Flow

An enabled and asserted system interrupt source is latched at the SSI and routed to the appropriate SCI based on the core target select (`SEC_SCTL[n].CTG`) bit field setting. The SEC priority ordering determines the highest priority pending system interrupt and the SCI updates the `SEC_CPND[n].SID` and `SEC_CACT[n].PRIO` bit field values. The SCI compares the `SEC_CPND[n]` register value against the highest priority active source in the `SEC_CACT[n]` register).

The priority level register (`SEC_CPLVL[n]`) determines how many of the MSBs the SEC uses in the comparison. The priority mask register (`SEC_CPMSK[n]`) and the group mask register (`SEC_CGMSK[n]`) determines which pending interrupt sources participate. If the `SEC_CPND[n]` register value is a higher priority (lower value) than the priority of the `SEC_CACT[n]` register from the comparison based on the `SEC_CPLVL[n]` register, the system interrupt output is asserted. The source ID register (`SEC_CSID[n]`) is updated with the `SEC_CPND[n].SID` bit field value and forwarded to the connected core.

After the core provides an interrupt acknowledgment, the interrupt source is active, until the SEC completes interrupt service with a write to the `SEC_END.SID` bit field with the same value. Note the following:

- Interrupt acknowledgement occurs with an MMR write of the `SEC_CSID[n]` register or the core version of the `SEC_CSID[n]` register.
- Interrupt active status indication is `SEC_SSTAT[n].ACT`==1.

The following sequence shows the example flow for a single interrupt.

1. The SEC compares the `SEC_CPND[n]` register value to the `SEC_CACT[n]` register value. If the interrupt in the `SEC_CPND[n]` register is higher priority, continue.
2. The SEC copies the `SEC_CPND[n]` register value to the `SEC_CSID[n]` register and asserts the interrupt signal.
3. The core reads the `SEC_CSID[n]` register (or core version).
4. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the interrupt signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source going active.
6. The core writes the `SEC_CSID[n]` of the active interrupt to the `SEC_END` register.
7. The SEC clears the`SEC_SSTAT[n].ACT` bit of the source being ended.

The following sequence shows the example flow for interrupt nesting where interrupt A is a lower priority and occurs earlier than interrupt B.

1. The SEC compares the `SEC_CPND[n]` (A) register value to the `SEC_CACT[n]` register and if the interrupt in the `SEC_CPND[n]` register is a higher priority, continue.
2. The SEC copies `SEC_CPND[n]` (A) register to the `SEC_CSID[n]` register and asserts the interrupt signal.
3. The core reads the `SEC_CSID[n]` (A) register (or core version).
4. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the `INT` signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source (A) going active.
6. The SEC compares the `SEC_CPND[n]` (B) register value to the `SEC_CACT[n]` (A) register value. If the `SEC_CACT[n]` (A) register value is a higher priority, continue.
7. The SEC copies the `SEC_CPND[n]` (B) register value to `SEC_CSID[n]` register and asserts the interrupt signal.
8. The core reads the `SEC_CSID[n]` (B) register (or core version).
9. The core writes to the `SEC_CSID[n]` register (or core version, asserts the acknowledge signal).
10. The SEC deasserts the `INT` signal and clears the `SEC_SSTAT[n].PND` bit and sets the `SEC_SSTAT[n].ACT` bit of the source (B) going active.
11. The core writes the `SEC_CSID[n]` of the active interrupt (B) to the `SEC_END` register.
12. The SEC clears the `SEC_SSTAT[n].ACT` bit of the source (B) being ended.
13. The core writes the `SEC_CSID[n]` of the active interrupt (A) to the `SEC_END` register.
14. The SEC clears the `SEC_SSTAT[n].ACT` bit of the source (A) being ended.

System Interrupt Priorities

Each system interrupt source has its own programmable priority level which is configured using the `SEC_SCTL[n].PRIO` bit field. The SCI evaluates the priority of all pending sources to determine the source of the highest-priority pending system interrupt for forwarding to the attached core. If more than one source of the pending system interrupt has the same priority setting, the SCI chooses the one with the lowest SID. For example, if SID 0, SID 1, and SID 2 are all pending and have the same priority setting, the SCI chooses SID 0 as the highest-priority source.

SEC Error

The processor includes an SEC error (`SEC_GSTAT.ERR`) as a source input to the SEC to allow for handling the error as an interrupt or fault.

SEC Programming Model

Implementing a system interrupt service model using the SEC requires, at a minimum:

- Proper configuration of a system interrupt source (for example a peripheral or DMA)
- A core interrupt or event service model

The core must be configured for response to system interrupts from the SEC. The SEC must be configured to enable and map the system interrupt source to the correct SCI and to forward interrupts to the connected core.

The system interrupt source must be configured to generate interrupt assertions. Alternatively, the processor can use software triggering for interrupt assertion. Software driven interrupts are generated by writing the source ID of the interrupt to be triggered to the `SEC_RAISE` register.

Programming Concepts

The following list provides the basic programming concepts necessary for configuring the SEC.

- Configuring an SSI as a system interrupt for a specific core.
- Configuring an SCI to provide system interrupts to the connected core. (See Configuring a System Source to Interrupt a Core.)
- Configuring an SSI as a system fault. (See Configuring a System Source as a Fault.)
- Configuring the SFI to manage system faults.

Programming Examples

This section provides example programming tasks that are typical for SEC usage.

Fault Management Interface Programming Model

The SFI interface can be programmed to manage fault events from system sources and associated actions such as issuing a system reset when watchdog expiration event occurs.

1. Set the `SEC_GCTL.EN` bit to enable the SEC.
2. Write to the `SEC_FCTL` register to configure specific fault actions.
    - Trigger Output. Set the `SEC_FCTL.TOEN` bit for the SEC to produce trigger outputs when a fault becomes active. The `SEC_FCTL.TES` bit can be programmed to select the event that directs the SEC to assert trigger output when a fault is pending or active. Configure slaves for SEC fault trigger master output.

      NOTE: If the `SEC_FCTL.TOEN` and or the `SEC_FCTL.TES` bits =1 (Trigger Output Enabled and Trigger on Fault Pending), an external fault (if enabled by the `SEC_FCTL.FIEN` bit) will not issue a trigger since Fault Pending is bypassed for external faults.

    - System Reset. The Reset Control Unit (RCU) controls how the functional units enter and exit reset. Configure the `RCU_CTL.SRSTREQEN` bit. This bit controls whether the sources of reset are enabled to perform a system reset. To issue a system reset request when a fault becomes active, set the `SEC_FCTL.SREN` bit. The SEC fault system reset delay register (`SEC_FSRDLY`) can be programmed for the delay, if required, from a fault becoming active to system reset request assertion.

- Fault Output. This configuration alows the SEC to indicate the fault status based on the `SEC_FCTL.CMS` bit configuration.
    - Computer Operating Normally (COP) mode. To configure fault output for COP mode, set the `SEC_FCTL.FOEN` bit to enable fault output. Set the `SEC_FCTL.CMS` bit to select COP mode to toggle the fault pin when no fault is active. Program the `SEC_FCOPP` period register with a desired width value for the COP toggled output pin.
    - Fault mode. Set the `SEC_FCTL.FOEN` bit to enable fault output. The `SEC_FCTL.CMS` bit should be set to Fault mode to toggle the fault pin when a fault is active.

3. If required, program the Fault Input to sample fault inputs from external devices on fault pins. Configure the `SEC_FCTL.FIEN` bit to enable the SEC to sample a fault input from an external device.

   ADDITIONAL INFORMATION: The `SEC_FCTL.FIEN` bit should be set only while the `SEC_FCTL.EN` bit is low. If the `SEC_FCTL.EN` bit is already high and the `SEC_FCTL.FIEN` bit needs to be set, the `SEC_FCTL.EN` bit should be cleared first. Fault input can only be enabled when Fault mode is selected by the `SEC_FCTL.CMS` bit.

4. Program the required fault delay to the `SEC_FDLY.COUNT` bit field if a delay between fault source assertion and the fault response is required.
5. Configure the `SEC_FCTL` register to enable the SEC.

   ADDITIONAL INFORMATION: The `SEC_FCTL.EN` bit should be set only while the `SEC_FSTAT.ACT` bit is low.

6. Write to the control register of a specific source register using the `SEC_SCTL[n]` register to enable the source as a fault.

Configuring a System Source to Interrupt a Core

To configure a system source to interrupt a core, the SEC itself must be enabled with the source interface (SSI) and core interface (SCI) properly initialized. Specifically, the SCI must be set up to accept interrupt signaling from the SEC and pass them to the specified core, and the SSI must properly enable each of the peripheral interrupt sources to generate interrupt signals and optionally define a priority scheme that overrides the default priority settings. In summary:

1. Write to the `SEC_GCTL` register to enable the SEC.
2. Write to the appropriate SCI `SEC_CCTL[n]` register to enable SEC interrupts to be sent to that core.
3. Write to the appropriate SSI `SEC_SCTL[n]` register to enable that peripheral as an interrupt source and to set the core target field to map the source to the desired SCI.
4. (Optional) By default, all the SEC interrupts are grouped as a single priority level, so passing of peripheral interrupt requests from the SEC is based solely on the default enumerated source ID. By programming the `SEC_CPLVL[n].PLVL` register, interrupt sources can be grouped into priority levels within the SEC such that arbitration is first performed by source ID within a grouped priority level before proceeding to the next

priority level, thus providing the flexibility to have lower-priority interrupt sources considered before higher-priority sources.

ADDITIONAL INFORMATION: The `SEC_CPMSK[n]` and `SEC_CGMSK[n]` registers must also can be programmed to mask the interrupts based on the customized levels and grouping.

Core/SEC Handshaking Requirements to Ensure Proper Interrupt Handling

Interrupt handling within an individual core requires specific handshaking with the SEC to ensure that nested interrupts are properly tracked and that new peripheral interrupts being raised within the SEC are either passed immediately to the core or held off and queued within the SEC for later servicing. Inside the SEC ISR, the following steps are required:

Use this procedure to write a custom dispatcher inside the Interrupt Service Routine. Note that the core needs to read the `SEC_CSID[n]` register and acknowledge it by writing the same value. It should also write to the `SEC_END` register after the ISR execution completes.

1. Read the `SEC_CSID[n]` register to obtain the source ID of the peripheral interrupt request.
2. Write the read value back to the `SEC_CSID[n]` register to send the acknowledge signal to the SEC that the core has accepted and begun processing for the interrupt request.
3. Execute the actual ISR (typically a call to a specific handler function from a look-up table based on the peripheral source ID). Write to the `SEC_GCTL` register to enable the SEC.
4. Write the `SEC_CSID[n]` of the active interrupt (read in step 1 above) to the `SEC_END` register to signal to the SEC that the interrupt has now been serviced.
5. Return from interrupt.

With this implementation in place, a higher-priority interrupt being raised by the SEC can be serviced by the core after step 2. The SEC knows what it passed to the core by virtue of its write to the `SEC_CSID[n]` register. After the core acknowledges that write, the SEC knows whether or not newly raised peripheral interrupts are higher priority than the highest-priority interrupt being processed by the core. If the new interrupt is higher priority, it pushes the current `SEC_CSID[n]` to an internal stack, writes the new `SEC_CSID[n]` value, and asserts a new SEC interrupt request. If it is lower priority, the SEC queues the interrupt until the core writes to the `SEC_END` register with the source ID of the higher-priority interrupt, thus confirming that it was fully processed, at which point that `SEC_CSID[n]` value is popped from the internal stack and any pending peripheral interrupt requests are arbitrated among before the SEC writes the new `SEC_CSID[n]` value and asserts a new interrupt request. Meanwhile, the core self-nests the latched SEC interrupt requests, as needed, when a higher-priority interrupt is presented to it, and the write to the `SEC_END` register in the SEC handler epilog code guarantees that each nested level has the required handshaking to signal to the SEC block that each individual source ID interrupt request is fully serviced. Please refer to the *SHARC+ Core Programming Reference* for more details regarding SEC handler code.

Configuring a System Source as a Fault

1. Write to the `SEC_GCTL` register to enable the SEC.

2. Write to the `SEC_FCTL` register to configure specific fault actions.
3. Write to the `SEC_FDLY` bit field to specify fault delay.
4. Write to the control register of a specific source to enable the source as a fault.

Configuring the WDOG Expiry Event to Issue a System Reset

Use the following procedure to configure the WDOG timer to issue a system reset.

1. Configure the `SEC_GCTL` register to enable the SEC.

   ADDITIONAL INFORMATION:

   ```
   *pREG_SEC0_GCTL = BITM_SEC_GCTL_EN;
   ```

2. Configure the `SEC_FCTL` register to choose the Fault response mode. In the following code example, the system reset is issued.

   ADDITIONAL INFORMATION:

   ```
   *pREG_RCU0_CTL  |= BITM_RCU_CTL_SRSTREQEN;
   *pREG_SEC0_FCTL |= BITM_SEC_FCTL_SREN;
   ```

   ADDITIONAL INFORMATION: Similarly, the program can also choose to signal the fault pin or choose to issue a trigger via the TRU as a response to the fault source (WDOG expiry is the fault source in this example) by programming the relevant bit fields in the `SEC_FCTL` register.

3. Configure the `SEC_SCTL[n].SEN` and `SEC_SCTL[n].FEN` bits in the Source Control 3 (n=3) register registers to determine how the fault source is handled. To configure the WDOG as the fault source, program the register. The program can configure any interrupt as the fault source by programming the corresponding register.

   ADDITIONAL INFORMATION:

   ```
    *pREG_SEC0_SCTL3 = BITM_SEC_SCTL_FEN|BITM_SEC_SCTL_SEN;
   ```

   ADDITIONAL INFORMATION: The SEC ID corresponding to WDOG0 is 3, as indicated in Table 7-3 ADSP-SC58x *Combined SEC and GIC Interrupt List*.

4. Write to the enable bit.

   ADDITIONAL INFORMATION:

   ```
   *pREG_SEC0_FCTL |= BITM_SEC_FCTL_EN;
   ```

SEC Programming Restrictions

Setting the `SEC_FCTL.EN` bit while the `SEC_FSTAT.ACT` bit is high can result in unpredictable behavior. To avoid this issue, set the `SEC_FCTL.EN` bit while the `SEC_FSTAT.ACT` bit is low. The `SEC_FSTAT.ACT` bit is only set when the `SEC_FCTL.EN` bit is high. Therefore, the problem can only occur if the `SEC_FCTL.EN` bit transitions from 1 to 0 and then to 1 again.

Writing to `SEC_FEND` to end a fault with both the `SEC_FCTL.FOEN` bit and the `SEC_FCTL.FIEN` bit set can result in erroneous external fault detection. If this operation (ending a fault) and configuration (fault input and fault output enabled) are required by the application, clear the `SEC_FCTL.FOEN` bit prior to writing to `SEC_FEND`. The recommended sequence for ending a fault with the `SEC_FCTL.FIEN` or `SEC_FCTL.FOEN`==1 is as follows:

1. Clear the `SEC_FCTL.FOEN` bit.
2. Write to the `SEC_FEND` register.
3. Set the `SEC_FCTL.FOEN` bit.

ADSP-SC58x SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 7-4: ADSP-SC58x SEC Register List

Name	Description
`SEC_CACT[n]`	SCI Active Register n
`SEC_CCTL[n]`	SCI Control Register n
`SEC_CGMSK[n]`	SCI Group Mask Register n
`SEC_CPLVL[n]`	SCI Priority Level Register n
`SEC_CPMSK[n]`	SCI Priority Mask Register n
`SEC_CPND[n]`	Core Pending Register n
`SEC_CSID[n]`	SCI Source ID Register n
`SEC_CSTAT[n]`	SCI Status Register n
`SEC_END`	Global End Register
`SEC_FCOPP`	Fault COP Period Register
`SEC_FCOPP_CUR`	Fault COP Period Current Register
`SEC_FCTL`	Fault Control Register
`SEC_FDLY`	Fault Delay Register
`SEC_FDLY_CUR`	Fault Delay Current Register
`SEC_FEND`	Fault End Register
`SEC_FSID`	Fault Source ID Register
`SEC_FSRDLY`	Fault System Reset Delay Register
`SEC_FSRDLY_CUR`	Fault System Reset Delay Current Register
`SEC_FSTAT`	Fault Status Register
`SEC_GCTL`	Global Control Register

Table 7-4: ADSP-SC58x SEC Register List (Continued)

Name	Description
`SEC_GSTAT`	Global Status Register
`SEC_RAISE`	Global Raise Register
`SEC_SCTL[n]`	Source Control Register n
`SEC_SSTAT[n]`	Source Status Register n

SCI Active Register n

The SEC SCI active interrupt register (`SEC_CACT[n]`) contains the source ID and priority of the highest priority active interrupt detected by the SEC prioritizer.

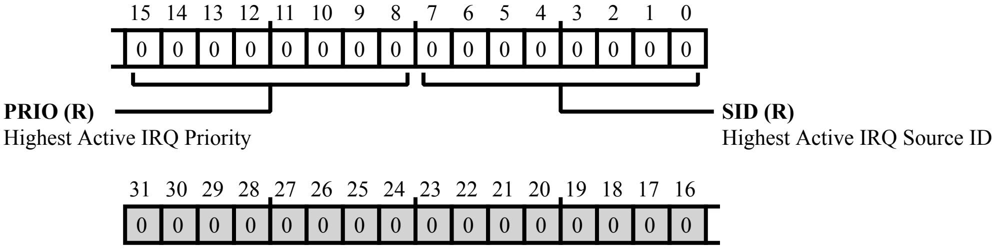

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PRIO (R)
Highest Active IRQ Priority
SID (R)
Highest Active IRQ Source ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-6: SEC_CACT[n] Register Diagram

Table 7-5: SEC_CACT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Active IRQ Priority. The `SEC_CACT[n].PRIO` indicates the priority value of the highest priority active interrupt for core n.
7:0 (R/NW)	SID	Highest Active IRQ Source ID. The `SEC_CACT[n].SID` identifies the source ID value of the highest priority active interrupt for core n.

SCI Control Register n

The SEC control register (`SEC_CCTL[n]`) contains SCI control bits for all system sources.

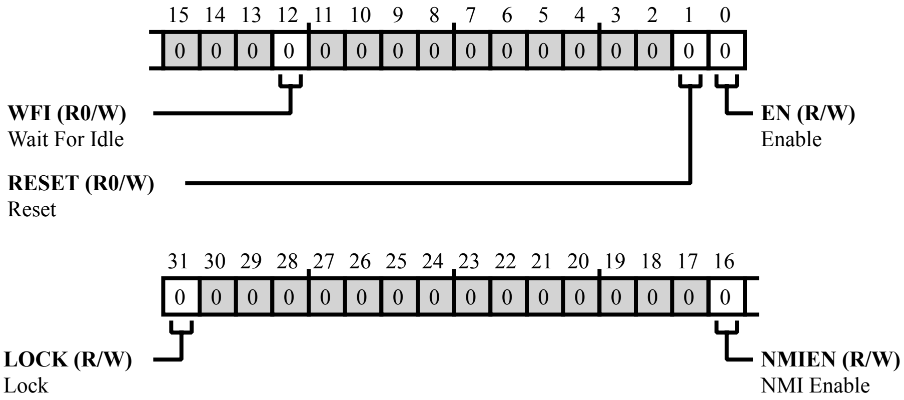

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WFI (R0/W)
Wait For Idle
RESET (R0/W)
Reset
EN (R/W)
Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
NMIEN (R/W)
NMI Enable

Figure 7-7: SEC_CCTL[n] Register Diagram

Table 7-6: SEC_CCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_CCTL[n].LOCK` bit is enabled, the `SEC_CCTL[n]` register is read only.	
		0	Unlock
		1	Lock
16 (R/W)	NMIEN	NMI Enable. The `SEC_CCTL[n].NMIEN` bit controls NMI propagation to the core. When the `SEC_CCTL[n].NMIEN` bit is enabled, the SCI allows NMIs to propagate to the core for servicing.	
		0	Disable
		1	Enable
12 (R0/W)	WFI	Wait For Idle. When set, the `SEC_CCTL[n].WFI` bit forces the SCI to wait for indication of core idle before the SCI resumes activity.	
		0	No Action
		1	Wait for Idle

Table 7-6: SEC_CCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W)	RESET	Reset. When set, the `SEC_CCTL[n].RESET` bit resets all SCI registers to their default values.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The `SEC_CCTL[n].EN` bit controls operation of the SCI. Clearing the `SEC_CCTL[n].EN` bit halts the execution of the SCI without resetting status registers. (The INT signal to a core is not affected.) Setting the `SEC_CCTL[n].EN` bit enables the SCI to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

SCI Group Mask Register n

The SEC SCI group mask register (`SEC_CGMSK[n]`) contains selections for a group mask, an ungroup mask, and a register lock. This register contains the system interrupt group masks for the connected core. The core uses the `SEC_CGMSK[n].UGRP` and `SEC_CGMSK[n].GRP` fields to mask (disable) interrupts from the specified groups.

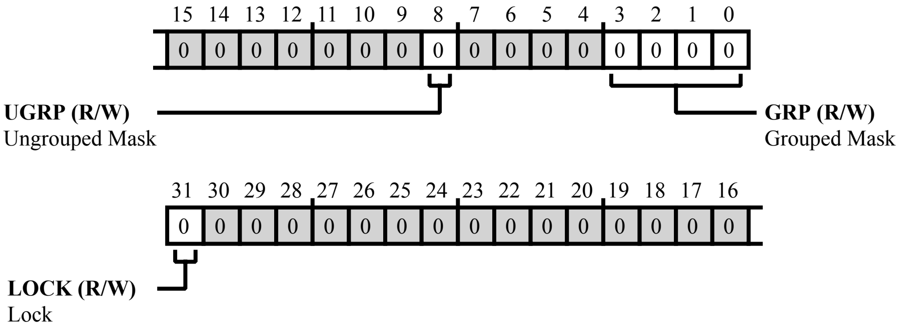

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
UGRP (R/W)
Ungrouped Mask
GRP (R/W)
Grouped Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 7-8: SEC_CGMSK[n] Register Diagram

Table 7-7: SEC_CGMSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_CGMSK[n].LOCK` bit is enabled, the `SEC_CGMSK[n]` register is read only.	
		0	Unlock
		1	Lock
8 (R/W)	UGRP	Ungrouped Mask. The `SEC_CGMSK[n].UGRP` bit masks interrupts (if set) for the ungrouped interrupt sources for core n.	
		0	Unmask Ungrouped Sources
		1	Mask Ungrouped Sources
3:0 (R/W)	GRP	Grouped Mask. The `SEC_CGMSK[n].GRP` field selects a group of interrupt sources to mask for core n. (For more information about interrupt source groups, see the `SEC_SCTL[n]` register description.)	
		0	No groups masked
		1	Mask group 0
		2	Mask group 1
		3	Mask groups 0, 1

Table 7-7: SEC_CGMSK[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	Mask group 2
		5	Mask groups 0, 2
		6	Mask groups 1, 2
		7	Mask groups 0, 1, 2
		8	Mask group 3
		9	Mask groups 0, 3
		10	Mask groups 1, 3
		11	Mask groups 0, 1, 3
		12	Mask groups 2, 3
		13	Mask groups 0, 2, 3
		14	Mask groups 1, 2, 3
		15	Mask groups 0, 1, 2, 3

SCI Priority Level Register n

The SEC SCI priority level register (`SEC_CPLVL[n]`) contains selections for priority levels and a register lock. This register is used to divide the total number of priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same level.

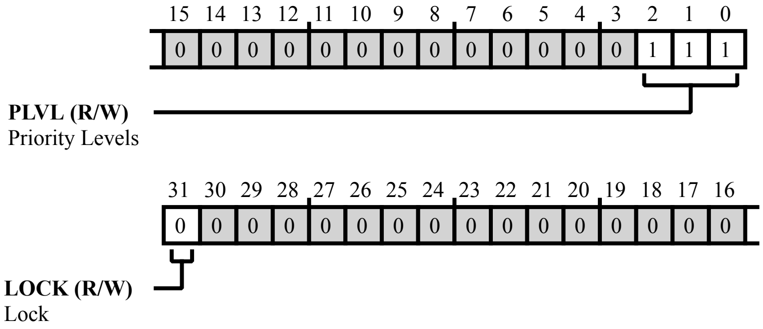


Figure 7-9: SEC_CPLVL[n] Register Diagram

Table 7-8: SEC_CPLVL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_CPLVL[n].LOCK` bit is enabled, the `SEC_CPLVL[n]` register is read only.	
		0	Unlock
		1	Lock
2:0 (R/W)	PLVL	Priority Levels. The `SEC_CPLVL[n].PLVL` field serves to divide the total number of interrupt priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same interrupt level. The sub-level priority value specifies the number of MSBs (minus 1) designated to interrupt levels, while the remaining LSBs are designated for sub-level specification. For example, if the `SEC_CPLVL[n].PLVL` field is set to two, the result is four priority levels are specified, because only the two MSBs are used for preemption evaluation. The remaining bits of the priority setting are used for sub-level prioritization.	
		0	1 MSBs (2 priority levels)
		1	2 MSBs (4 priority levels)
		2	3 MSBs (8 priority levels)
		3	4 MSBs (16 priority levels)
		4	5 MSBs (32 priority levels)
		5	6 MSBs (64 priority levels)

Table 7-8: SEC_CPLVL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	7 MSBs (128 priority levels)
		7	8 MSBs (256 priority levels)

SCI Priority Mask Register n

The SEC SCI priority mask register (`SEC_CPMSK[n]`) contains the SCI priority mask for core n and includes a register lock.

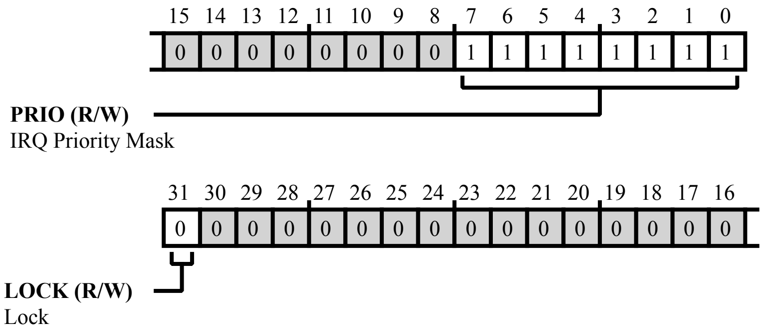

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
PRIO (R/W)
IRQ Priority Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 7-10: SEC_CPMSK[n] Register Diagram

Table 7-9: SEC_CPMSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_CPMSK[n].LOCK` bit is enabled, the `SEC_CPMSK[n]` register is read only.	
		0	Unlock
		1	Lock
7:0 (R/W)	PRIO	IRQ Priority Mask. The `SEC_CPMSK[n].PRIO` contains the system interrupt priority mask for core n. The core uses the `SEC_CPMSK[n].PRIO` field to mask (block) interrupts below the specified level.	
		0	Priority level 0 (highest)
		1-254	
		255	Priority level 255 (lowest)

Core Pending Register n

The SCI pending interrupt register (`SEC_CPND[n]`) contains the source ID and priority of the highest priority pending interrupt detected by the SEC prioritizer.

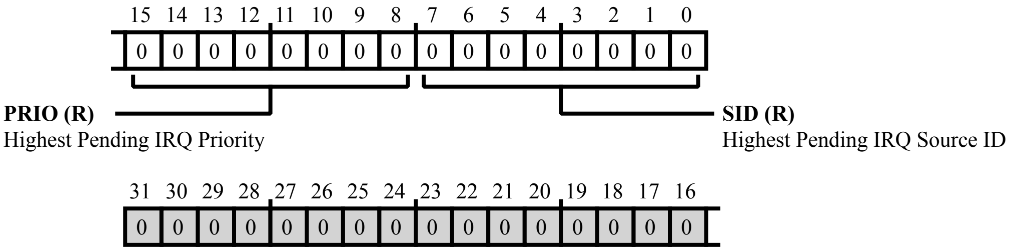

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PRIO (R)
Highest Pending IRQ Priority
SID (R)
Highest Pending IRQ Source ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-11: SEC_CPND[n] Register Diagram

Table 7-10: SEC_CPND[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Pending IRQ Priority. The `SEC_CPND[n].PRIO` indicates the priority value of the highest priority pending interrupt for core n.
7:0 (R/NW)	SID	Highest Pending IRQ Source ID. The `SEC_CPND[n].SID` identifies the source ID value of the highest priority pending interrupt for core n.

SCI Source ID Register n

The SCI source ID register (`SEC_CSID[n]`) contains the source ID of the interrupt last issued to core n. The `SEC_CSID[n]` register value is loaded by the SCI when a system interrupt indication is sent to core n. The SCI does not change the `SEC_CSID[n]` until after the interface receives an interrupt acknowledge from core n. Writing to the `SEC_CSID[n]` register generates an interrupt acknowledge, but does not update the value in the register.

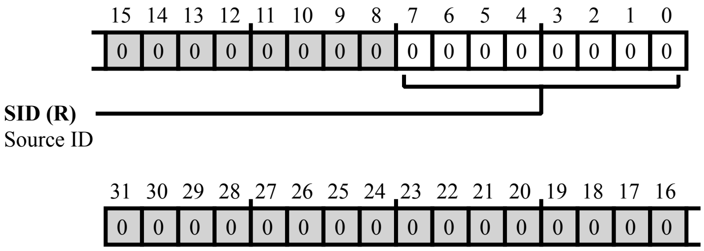

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SID (R)
Source ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-12: SEC_CSID[n] Register Diagram

Table 7-11: SEC_CSID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	SID	Source ID. The `SEC_CSID[n].SID` bit is the source ID of the interrupt last issued to core n.

SCI Status Register n

The SCI status register (`SEC_CSTAT[n]`) contains status bits, indicating the operational status of the SCI.

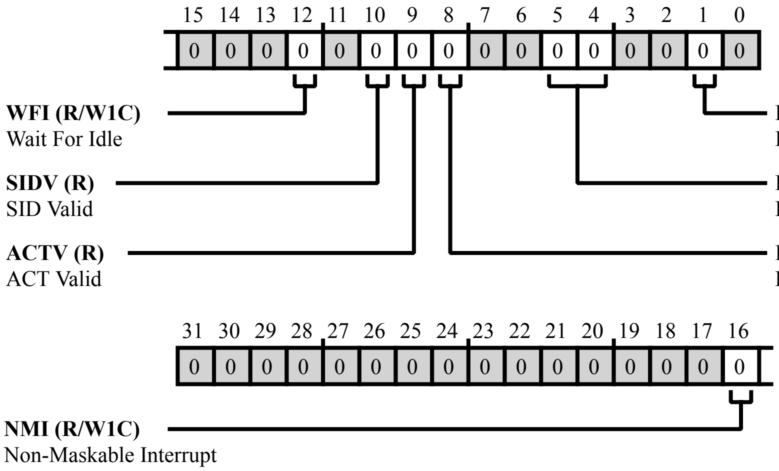

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WFI (R/W1C)
Wait For Idle
SIDV (R)
SID Valid
ACTV (R)
ACT Valid
ERR (R/W1C)
Error
ERRC (R)
Error Cause
PNDV (R)
PND Valid
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NMI (R/W1C)
Non-Maskable Interrupt

Figure 7-13: SEC_CSTAT[n] Register Diagram

Table 7-12: SEC_CSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	NMI	Non-Maskable Interrupt. The `SEC_CSTAT[n].NMI` bit indicates whether an NMI has occurred since the bit was last cleared.	
		0	No NMI Occurred
		1	NMI Occurred
12 (R/W1C)	WFI	Wait For Idle. The `SEC_CSTAT[n].WFI` bit indicates (if set) that the SCI is temporarily disabled, pending a core idle indication. This bit is set when `SEC_CCTL[n].WFI` is set.	
		0	Not Waiting
		1	Waiting
10 (R/NW)	SIDV	SID Valid. The `SEC_CSTAT[n].SIDV` bit indicates (if set) that the current value in the `SEC_CSID[n]` register is valid. The SCI sets the `SEC_CSTAT[n].SIDV` bit when the updating the `SEC_CSID[n]` register with a new value. The `SEC_CSTAT[n].SIDV` bit is cleared when the `SEC_CSID[n]` register is written. This status indication may be used to extract all pending interrupts in a single interrupt service routine.	
		0	Invalid
		1	Valid

Table 7-12: SEC_CSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	ACTV	ACT Valid. The `SEC_CSTAT[n].ACTV` bit indicates (if set) that the current value in the `SEC_CACT[n]` register is valid. The SCI sets the `SEC_CSTAT[n].ACTV` bit when updating the `SEC_CACT[n]` registers with a new value. The `SEC_CSTAT[n].ACTV` bit is cleared when the `SEC_CSID[n]` register is written.	
		0	Invalid
		1	Valid
8 (R/NW)	PNDV	PND Valid. The `SEC_CSTAT[n].PNDV` bit indicates (if set) that the current value in the `SEC_CPND[n]` register is valid. The SCI sets the `SEC_CSTAT[n].PNDV` bit when updating the `SEC_CPND[n]` register with a new value. The `SEC_CSTAT[n].PNDV` bit is cleared when the `SEC_CSID[n]` register is written.	
		0	Invalid
		1	Valid
5:4 (R/NW)	ERRC	Error Cause. The `SEC_CSTAT[n].ERRC` bits are updated on assertion of the `SEC_CSTAT[n].ERR` bit to indicate the SCI error type. `SEC_CSTAT[n].ERRC` is only updated on the assertion of `SEC_CSTAT[n].ERR`. Subsequent errors while `SEC_CSTAT[n].ERR` is asserted do not update `SEC_CSTAT[n].ERRC`.	
		0	Reserved
		1	Acknowledge Error. SCI has received an acknowledge without a pending, unacknowledged interrupt present.
		2	Reserved
		3	Reserved
1 (R/W1C)	ERR	Error. The `SEC_CSTAT[n].ERR` bit indicates that an error has occurred in the SCI. When `SEC_CSTAT[n].ERR` is set, the SCI updates the `SEC_CSTAT[n].ERRC` field to the value of the corresponding error cause.	
		0	No Error
		1	Error Occurred

Global End Register

The SEC global end register (SEC_END) contains a source ID interrupt service end field (SEC_END.SID). When a core has finished servicing an interrupt, the core writes the SEC_END.SID field in the SEC_END register. This write causes the SEC to clear the SEC_SSTAT[n].ACT bit in the SEC_SSTAT[n] register of the corresponding interrupt.

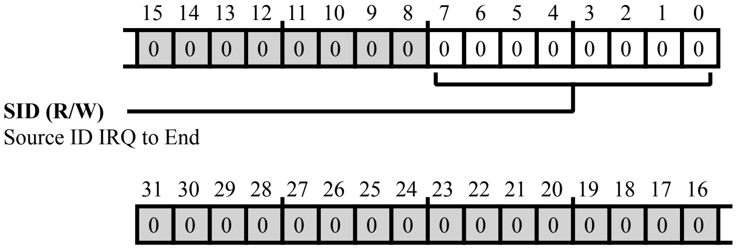

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SID (R/W)
Source ID IRQ to End
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-14: SEC_END Register Diagram

Table 7-13: SEC_END Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID IRQ to End. The SEC_END.SID bit field contains the source ID interrupt service end value.

Fault COP Period Register

The SEC fault COP period register (`SEC_FCOPP`) contains the width value (count in (SEC) clock cycles) for the high and low phase of the computer operating properly (COP) toggled output on the COP pin. Note that the actual high/low phase value is the `SEC_FCOPP.COUNT` programmed value plus 1.

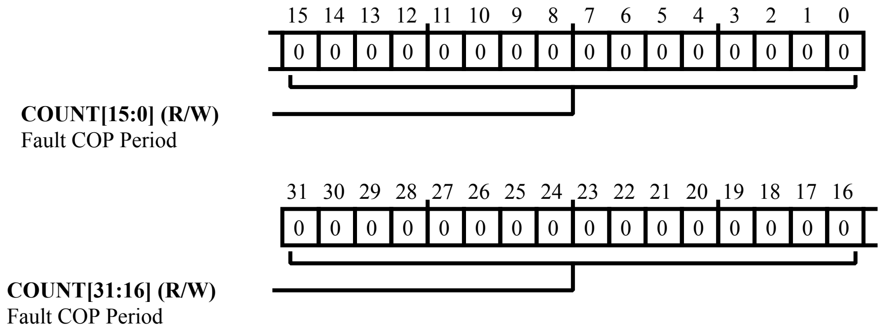

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R/W)
Fault COP Period
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R/W)
Fault COP Period

Figure 7-15: SEC_FCOPP Register Diagram

Table 7-14: SEC_FCOPP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault COP Period. The `SEC_FCOPP.COUNT` bit field is the width value for the high and low phase of the computer operating properly (COP) toggled output on the COP pin.

Fault COP Period Current Register

The SEC fault COP period current register (SEC_FCOPP_CUR) contains the active count (in (SEC) clock periods) for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin. The SEC loads the SEC_FCOPP_CUR register from the SEC_FCOPP register when the SEC_FCOPP_CUR.COUNT field is cleared and the SEC is in COP mode (SEC_FCTL.CMS bit =1). The SEC decrements the SEC_FCOPP_CUR count each (SEC) clock cycle while SEC_FCTL.CMS is set and the SEC_FSTAT.ACT bit is not set.

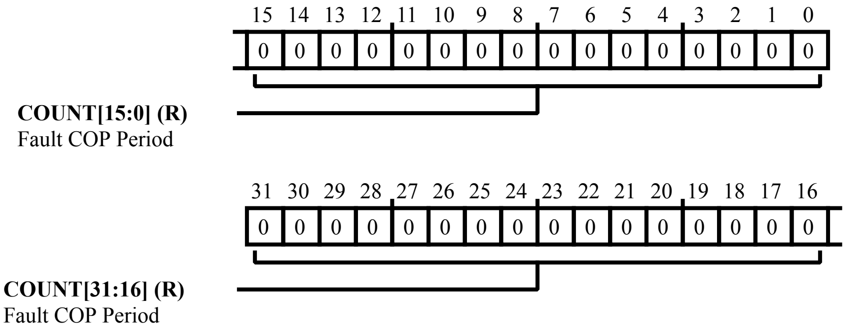

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R)
Fault COP Period
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R)
Fault COP Period

Figure 7-16: SEC_FCOPP_CUR Register Diagram

Table 7-15: SEC_FCOPP_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault COP Period. The SEC_FCOPP_CUR.COUNT bit field is the active count for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin.

Fault Control Register

The SEC fault control register (SEC_FCTL) contains fault control bits for all SEC channels. This register controls the operation of the System Fault Management Interface (SFI).

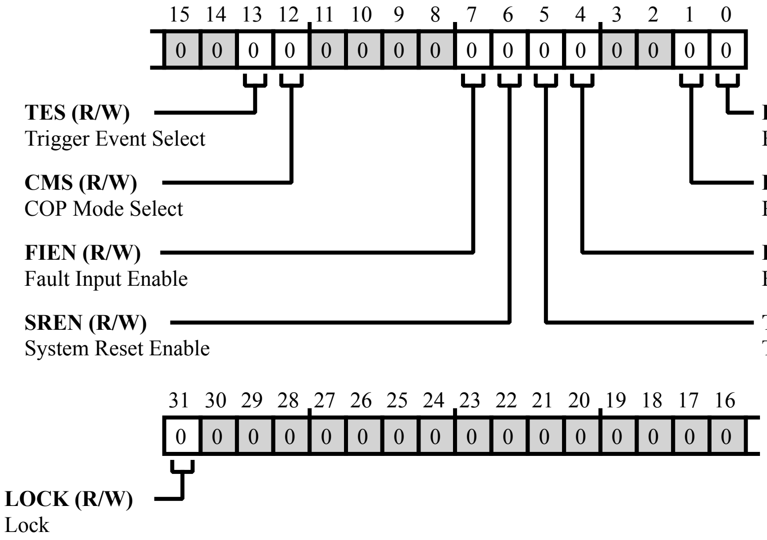

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TES (R/W)
Trigger Event Select
CMS (R/W)
COP Mode Select
FIEN (R/W)
Fault Input Enable
SREN (R/W)
System Reset Enable
EN (R/W)
Enable
RESET (R0/W)
Reset
FOEN (R/W)
Fault Output Enable
TOEN (R/W)
Trigger Output Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 7-17: SEC_FCTL Register Diagram

Table 7-16: SEC_FCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_FCTL.LOCK` bit is enabled, the `SEC_FCTL` register is read only.	
		0	UnLock
		1	Lock
13 (R/W)	TES	Trigger Event Select. The `SEC_FCTL.TES` bit selects the event that directs the SEC to assert trigger output. In fault pending mode, the SEC asserts trigger output when a fault is pending. In fault active mode, the SEC asserts trigger output when a fault is active.	
		0	Fault Active Mode
		1	Fault Pending Mode

Table 7-16: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	CMS	COP Mode Select. The `SEC_FCTL.CMS` selects the SEC mode for handling fault input. In COP mode, the SEC toggles the COP pin to indicate that no fault is active and ceases toggling the pin to indicate that a fault is active. In fault mode, the SEC deasserts the fault pin (=0) and fault_b pin (=1) when no fault is active and asserts the fault pin (=1) and fault_b pin (=0) when a fault is active. Not all processors feature both the fault and fault_b pins. Refer to the product data sheet for details.	
		0	Fault Mode
		1	COP Mode
7 (R/W)	FIEN	Fault Input Enable. The `SEC_FCTL.FIEN` bit enables the SEC to sample fault input. If `SEC_FCTL.FIEN` is set (=1), a fault indication from an external device sets the `SEC_FSTAT.ACT` bit and `SEC_FSID.FEXT` bit.	
		0	Disable
		1	Enable
6 (R/W)	SREN	System Reset Enable. The `SEC_FCTL.SREN` bit enables the SEC to issue a system reset request when a fault becomes active.	
		0	Disable
		1	Enable
5 (R/W)	TOEN	Trigger Output Enable. The `SEC_FCTL.TOEN` bit enables the SEC to produce trigger output when a fault becomes active.	
		0	Disable
		1	Enable
4 (R/W)	FOEN	Fault Output Enable. The `SEC_FCTL.FOEN` bit enables the SEC to indicate fault status, according to the `SEC_FCTL.CMS` bit configuration.	
		0	Disable
		1	Enable
1 (R0/W)	RESET	Reset. Setting the `SEC_FCTL.RESET` bit resets ALL SEC registers to their default values.	
		0	No Action
		1	Reset

Table 7-16: SEC_FCTL Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">0
(R/W)</td><td rowspan="3">EN</td><td colspan="2">Enable.
The <code>SEC_FCTL.EN</code> bit controls the operational state of the SEC. Clearing the <code>SEC_FCTL.EN</code> bit halts the execution of the SEC without resetting status registers. Setting the <code>SEC_FCTL.EN</code> bit enables the SEC to begin or resume operation with the current configuration and status.</td></tr>
<tr><td>0</td><td>Disable</td></tr>
<tr><td>1</td><td>Enable</td></tr>
</table>

Fault Delay Register

The SEC fault delay register (`SEC_FDLY`) contains the number (`SEC_FDLY.COUNT` field) of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

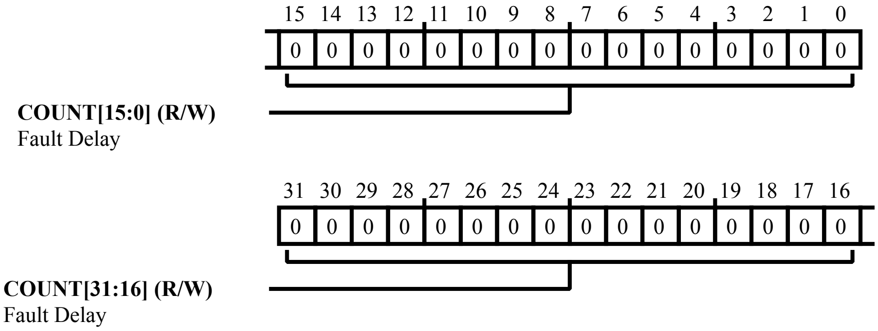

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R/W)
Fault Delay
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R/W)
Fault Delay

Figure 7-18: SEC_FDLY Register Diagram

Table 7-17: SEC_FDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault Delay. The `SEC_FDLY.COUNT` bit field is the number of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

Fault Delay Current Register

The SEC fault delay current register (`SEC_FDLY_CUR`) contains the active count (`SEC_FDLY_CUR.COUNT` field) in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled. The count is loaded from the `SEC_FDLY` register when a fault becomes pending (`SEC_FSTAT.PND` bit is set). The SEC decrements the value in `SEC_FDLY_CUR` each (SEC) clock cycle while the `SEC_FSTAT.PND` bit is set.

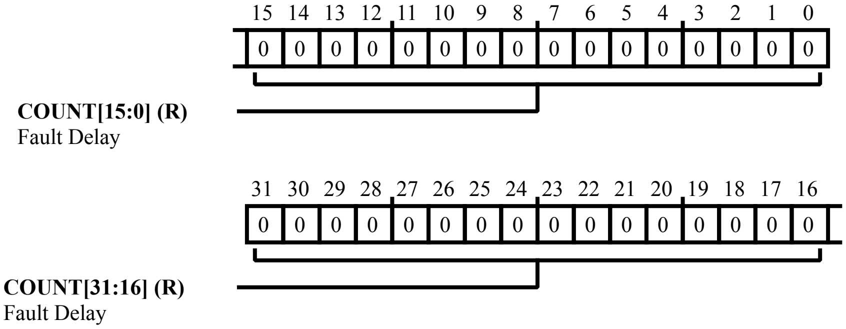

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R)
Fault Delay
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R)
Fault Delay

Figure 7-19: SEC_FDLY_CUR Register Diagram

Table 7-18: SEC_FDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault Delay. The `SEC_FDLY_CUR.COUNT` bit field is the active count in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled.

Fault End Register

The SEC fault end register (`SEC_FEND`) contains fault source ID and internal/external fields. This register receives fault end indication from a core.

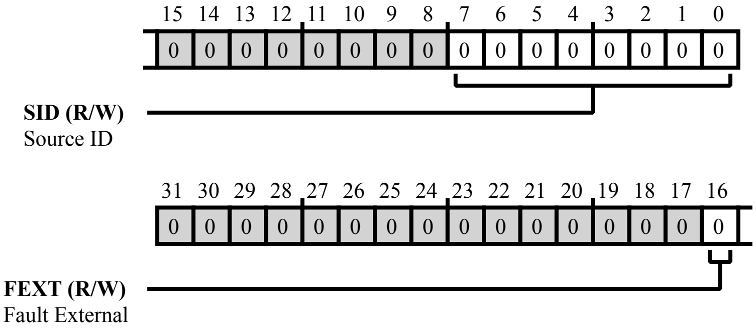

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SID (R/W)
Source ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FEXT (R/W)
Fault External

Figure 7-20: SEC_FEND Register Diagram

Table 7-19: SEC_FEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FEXT	Fault External. Setting the `SEC_FEND.FEXT` bit, when the `SEC_FEND.SID` field is cleared, clears an active fault from an external source.	
		0	Fault Internal
		1	Fault External
7:0 (R/W)	SID	Source ID. The `SEC_FEND.SID` identifies a fault to be ended as indicated to the SEC by the core. The core loads the `SEC_FEND.SID` field value. If the `SEC_FEND.SID` value matches the `SEC_FSID.SID` value, the `SEC_FSTAT.PND` bit and `SEC_FSTAT.ACT` bit are cleared.	

Fault Source ID Register

The SEC fault source ID register (`SEC_FSID`) contains a fault source ID and internal/external fields.

NOTE:These bits are not reset by system reset so that a fault that automatically triggers a system reset to avoid a fault may be analyzed after the reset occurs.

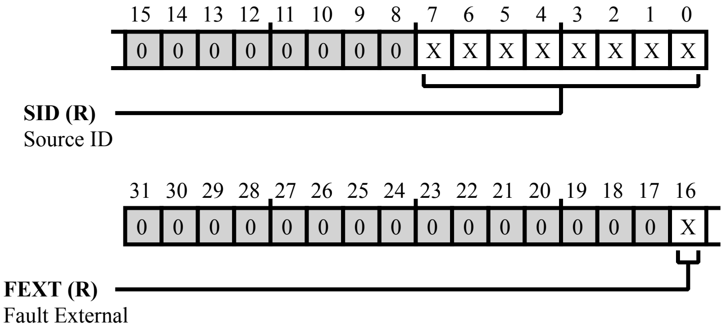


Figure 7-21: SEC_FSID Register Diagram

Table 7-20: SEC_FSID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	FEXT	Fault External. The `SEC_FSID.FEXT` bit indicates that the last active fault was asserted by an external device. The SEC sets the `SEC_FSID.FEXT` bit when the `SEC_FSTAT.ACT` bit is set by the fault input pins. The `SEC_FSID.FEXT` bit is cleared when the `SEC_FSTAT.ACT` bit is set by an internal fault or when the external fault is ended. When the `SEC_FSID.FEXT` bit is set, the `SEC_FSID.SID` is cleared.	
		0	Fault Internal
		1	Fault External
7:0 (R/NW)	SID	Source ID. The `SEC_FSID.SID` identifies the fault assertion detected by the SEC fault interface. The SEC loads the `SEC_FSID.SID` field value when a system fault indication is asserted. The SEC fault interface does not change the `SEC_FSID.SID` value until the fault is no longer pending or active, as indicated by the `SEC_FSTAT.PND` bit and `SEC_FSTAT.ACT` bit being cleared in the `SEC_FSTAT` register.	

Fault System Reset Delay Register

The SEC fault system reset delay register (`SEC_FSRDLY`) contains the number (`SEC_FSRDLY.COUNT` field) of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion, if enabled.

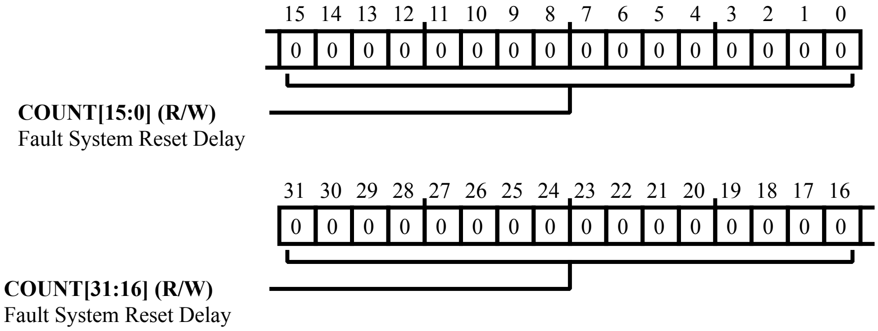

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R/W)
Fault System Reset Delay
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R/W)
Fault System Reset Delay

Figure 7-22: SEC_FSRDLY Register Diagram

Table 7-21: SEC_FSRDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault System Reset Delay. The `SEC_FSRDLY.COUNT` bit field is the number of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion.

Fault System Reset Delay Current Register

The SEC fault system reset delay current register (`SEC_FSRDLY_CUR`) contains the active count (`SEC_FSRDLY_CUR.COUNT` field) in (SEC) clock periods for the delay from fault active to system reset assertion, if enabled. The count is loaded from the `SEC_FSRDLY` register when a fault becomes active (`SEC_FSTAT.ACT` bit is set). The SEC decrements the value in `SEC_FSRDLY_CUR` each (SEC) clock cycle while the `SEC_FSTAT.ACT` bit is set.

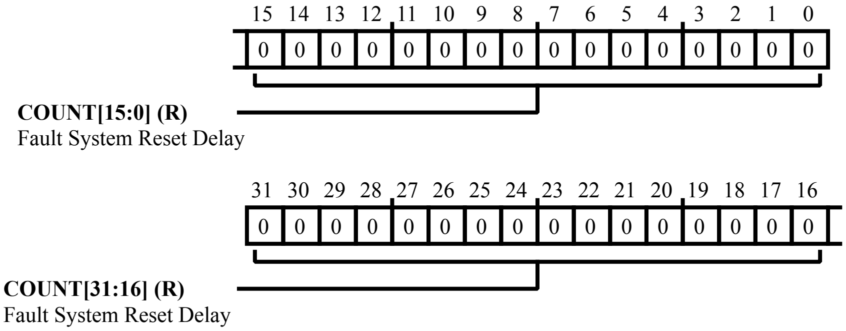

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[15:0] (R)
Fault System Reset Delay
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT[31:16] (R)
Fault System Reset Delay

Figure 7-23: SEC_FSRDLY_CUR Register Diagram

Table 7-22: SEC_FSRDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault System Reset Delay. The `SEC_FSRDLY_CUR.COUNT` bit field is the active count in (SEC) clock periods for the delay from fault active to system reset assertion.

Fault Status Register

The SEC fault status register (SEC_FSTAT) indicates the operational status of the SFI.

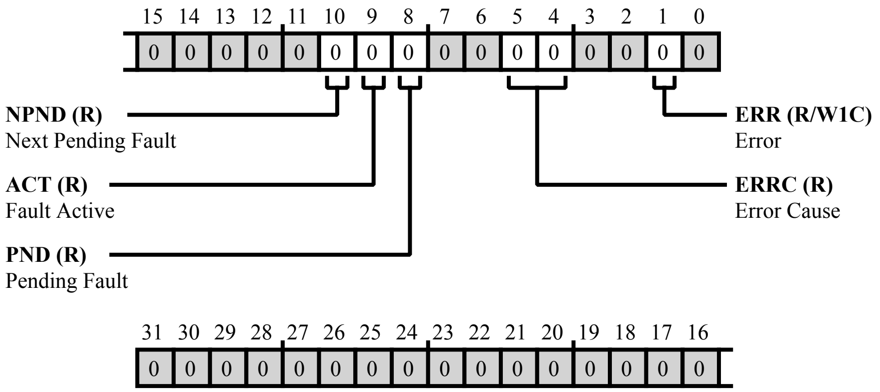

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NPND (R)
Next Pending Fault
ACT (R)
Fault Active
PND (R)
Pending Fault
ERR (R/W1C)
Error
ERRC (R)
Error Cause
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-24: SEC_FSTAT Register Diagram

Table 7-23: SEC_FSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	NPND	Next Pending Fault. The SEC_FSTAT.NPND bit indicates that one or more sources have signaled fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the SEC_FSTAT.NPND bit when the fault interface detects assertion of any enabled fault source input, while either the SEC_FSTAT.PND or SEC_FSTAT.ACT bits are set. The SEC clears the SEC_FSTAT.NPND bit when there are no fault sources waiting.	
		0	Not Pending
		1	Pending
9 (R/NW)	ACT	Fault Active. The SEC_FSTAT.ACT bit indicates that the SEC has received a fault source input, the current fault delay count (in the SEC_FDLY_CUR register) has expired, and the fault actions are enabled. The SEC also sets the SEC_FSTAT.ACT bit on fault input detection if the SEC_FCTL.FIEN bit is set. The SEC_FSTAT.ACT bit is cleared by writing the ID value of the asserted fault from SEC_FSID register to the SEC_FEND register.	
		0	No Fault
		1	Active Fault

Table 7-23: SEC_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	PND	Pending Fault. The `SEC_FSTAT.PND` bit indicates a fault source has signaled a fault assertion to the SEC, but the SEC has not yet triggered the event actions due to the delay selected with the `SEC_FDLY` register. The SEC fault interface sets the `SEC_FSTAT.PND` bit when the `SEC_FSID` is updated on assertion of a fault source input. The `SEC_FSTAT.PND` bit is only set when the `SEC_FSTAT.ACT` bit is cleared. The SEC updates the `SEC_FSID` register with the SID value when the `SEC_FSTAT.PND` bit is set. The `SEC_FSTAT.PND` bit is cleared *either* by the SEC fault interface when the current delay count in the `SEC_FDLY_CUR` register expires *or* by writing the `SEC_FSID.SID` field value (which indicates the ID of the asserted fault) to the `SEC_FEND` register.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the `SEC_FSTAT.ERR` bit is asserted, the SEC updates `SEC_FSTAT.ERRC` field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. When the error type is end error, the status indicates that an end was received for a source while the `SEC_FSTAT.ACT` bit was not set.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. The `SEC_FSTAT.ERR` bit indicates an SEC fault interface error. When `SEC_FSTAT.ERR` is set, the SEC updates the `SEC_FSTAT.ERRC` field to indicate the corresponding error cause. When multiple errors occur, the `SEC_FSTAT` register captures the status for the first error and does not capture subsequent errors until the status is cleared.	
		0	No Error
		1	Error Occurred

Global Control Register

The SEC global control register (`SEC_GCTL`) provides register locking, reset, and enable for the SEC module.

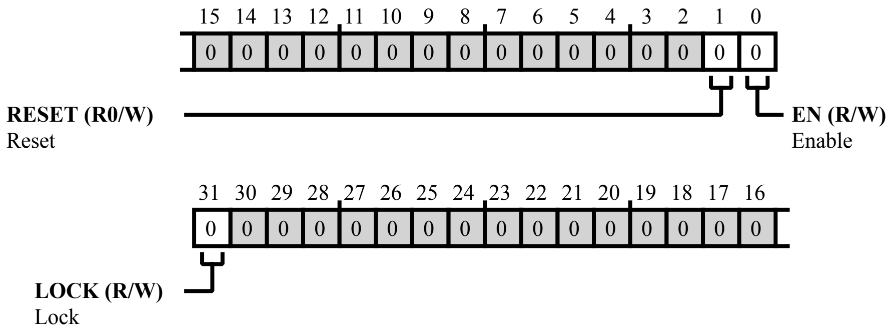

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RESET (R0/W)
Reset
EN (R/W)
Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 7-25: SEC_GCTL Register Diagram

Table 7-24: SEC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_GCTL.LOCK` bit is enabled, the `SEC_GCTL` register is read only.	
		0	Unlock
		1	Lock
1 (R0/W)	RESET	Reset. The `SEC_GCTL.RESET` bit is write-1-action and triggers a soft reset to all SEC registers.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The `SEC_GCTL.EN` bit is read/write and must be set for the SEC to begin/resume SEC operation with the current configuration and status. Clearing the `SEC_GCTL.EN` bit halts the execution of the SFI and all SCIs. All SSIs remain active, along with all error detection, without resetting status registers.	
		0	Disable
		1	Enable

Global Status Register

The SEC global status register (`SEC_GSTAT`) contains global status bits for the SEC.

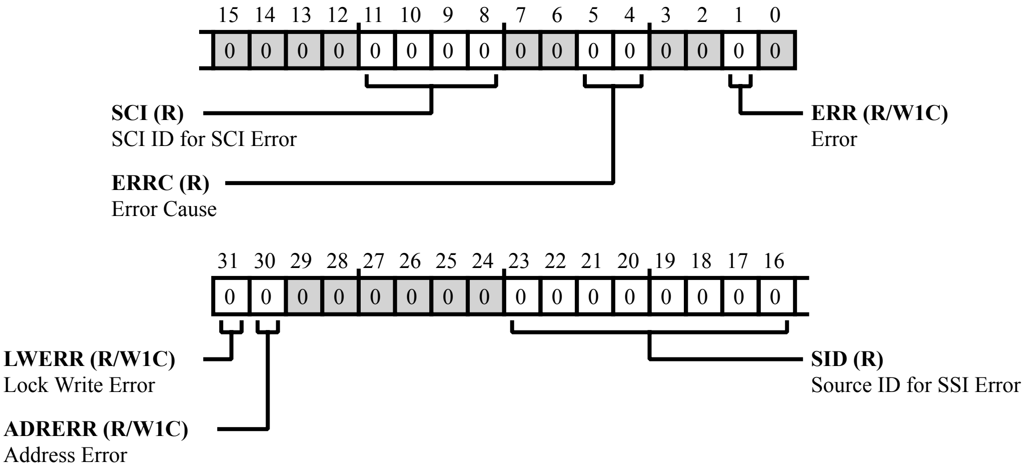

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCI (R)
SCI ID for SCI Error
ERRC (R)
Error Cause
ERR (R/W1C)
Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LWERR (R/W1C)
Lock Write Error
ADRERR (R/W1C)
Address Error
SID (R)
Source ID for SSI Error

Figure 7-26: SEC_GSTAT Register Diagram

Table 7-25: SEC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The `SEC_GSTAT.LWERR` bit indicates (when set) there was an attempted write to an SEC register while the `SEC_GCTL.LOCK` bit was set and while the global lock bit was enabled (`SPU_CTL.GLCK` bit =1). This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
30 (R/W1C)	ADRERR	Address Error. The `SEC_GSTAT.ADRERR` bit indicates that the SEC generated and address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
23:16 (R/NW)	SID	Source ID for SSI Error. The `SEC_GSTAT.SID` bits indicate the source ID that generated the last SSI error conveyed in the `SEC_GSTAT.ERRC` field.	
11:8 (R/NW)	SCI	SCI ID for SCI Error. The `SEC_GSTAT.SCI` bits indicate the number for the specific SCI that generated the last SCI error conveyed in the `SEC_GSTAT.ERRC` field.	

Table 7-25: SEC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/NW)	ERRC	Error Cause. When the SEC updates the `SEC_GSTAT.ERR` bit, the SEC updates the `SEC_GSTAT.ERRC` bits to indicate the error type. Note that for SCI errors, the error status represents an OR of all the errors from each SCI. Note that for SSI errors, the error status indicates an error is active for any SSI input. This error is an OR of all the interrupt source errors.	
		0	SFI Error
		1	SCI Error
		2	SSI Error
		3	Reserved
1 (R/W1C)	ERR	Error. The `SEC_GSTAT.ERR` bit indicates an error has occurred in the SEC. When the SEC asserts this bit (=1), the SEC updates the `SEC_GSTAT.ERRC` field to indicate the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of this bit. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred

Global Raise Register

The SEC global raise register (`SEC_RAISE`) contains a source ID event set-to-pending field (`SEC_RAISE.SID`). When a source ID value is written to this field, the SEC raises the source's event status to pending.

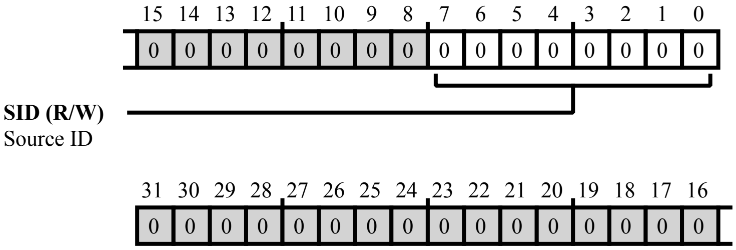

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SID (R/W)
Source ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-27: SEC_RAISE Register Diagram

Table 7-26: SEC_RAISE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID. The `SEC_RAISE.SID` bit field is the source ID of event that is set to pending status.

Source Control Register n

The SEC source control register (`SEC_SCTL[n]`) contains control bits to configure the SEC event sources. This register controls the configuration of the corresponding SEC event source.

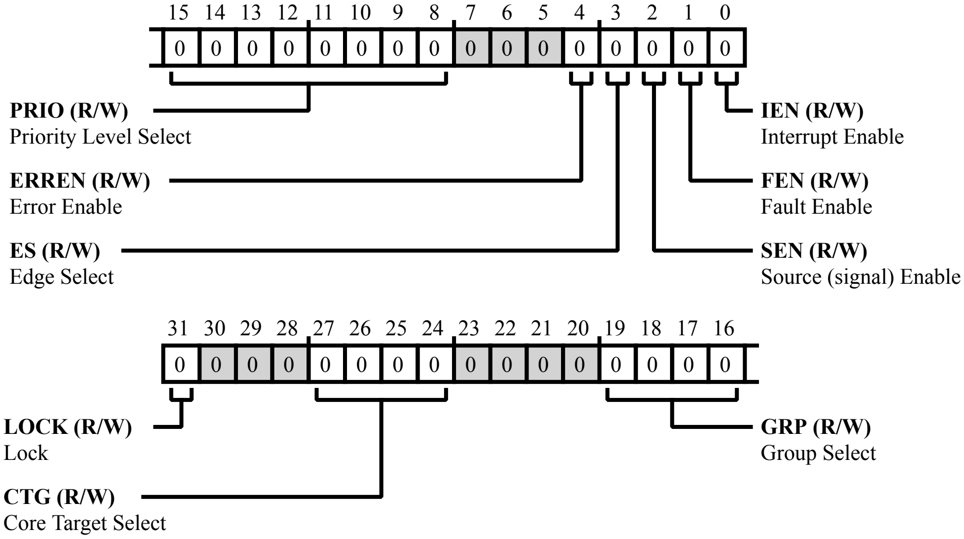

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PRIO (R/W)
Priority Level Select
ERREN (R/W)
Error Enable
ES (R/W)
Edge Select
IEN (R/W)
Interrupt Enable
FEN (R/W)
Fault Enable
SEN (R/W)
Source (signal) Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
CTG (R/W)
Core Target Select
GRP (R/W)
Group Select

Figure 7-28: SEC_SCTL[n] Register Diagram

Table 7-27: SEC_SCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `SEC_SCTL[n].LOCK` bit is enabled, the `SEC_SCTL[n]` register is read only.	
		0	Unlock
		1	Lock
27:24 (R/W)	CTG	Core Target Select. The `SEC_SCTL[n].CTG` bits selects the specific SEC core interface to which the interrupt is mapped. Each system interrupt is mapped uniquely to one specific SEC core interface and (as a result) to a specific core.	

Table 7-27: SEC_SCTL[n] Register Fields (Continued)

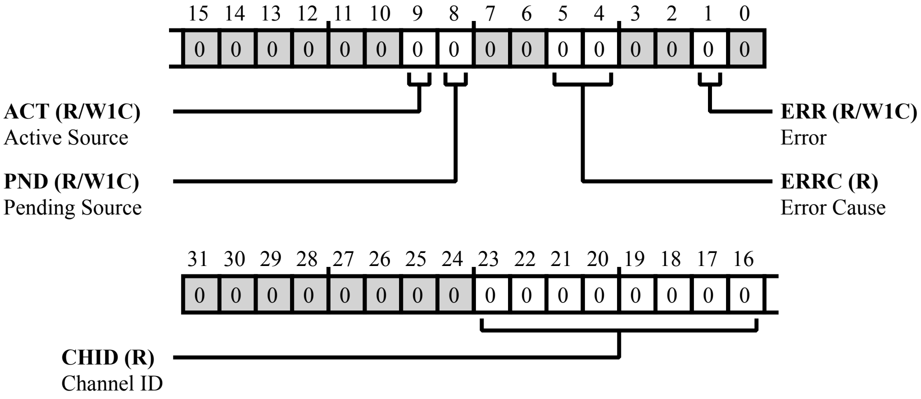
Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	GRP	Group Select. The `SEC_SCTL[n].GRP` bits each select a specific group for the interrupt. Each system interrupt can be assigned to any combination of groups supported by the `SEC_SCTL[n].GRP` field. For example, consider the situation where `SEC_SCTL[n].GRP`0 represents interrupt group 0, `SEC_SCTL[n].GRP`1 represents interrupt group 1, and so on. One group might be used for all enabled interrupts (for example, group 0) and an additional group might be used for all wakeup interrupts (for example, group 1). This approach supports a model of all interrupts and just the wakeup subset. Before going to idle or sleep, all non-wakeup interrupts can be masked off to allow only wakeup interrupts to be enabled for service. Selecting no group (all `SEC_SCTL[n].GRP` bits = 0) places the interrupt source in the category of "ungrouped".	
15:8 (R/W)	PRIO	Priority Level Select. The `SEC_SCTL[n].PRIO` bits sets the relative priority for an interrupt request. A pending interrupt request forwards its `SEC_SCTL[n].PRIO` value to the SEC core interface.	
4 (R/W)	ERREN	Error Enable. The `SEC_SCTL[n].ERREN` bit permits the `SEC_SSTAT[n].ERR` status bit to be set on error detection. If `SEC_SCTL[n].ERREN` is cleared, no errors are detected.	
		0	Disable
		1	Enable
3 (R/W)	ES	Edge Select. The `SEC_SCTL[n].ES` bit selects the operational and sensitivity mode of the SEC source interface input.	
		0	Level Sensitive
		1	Edge Sensitive
2 (R/W)	SEN	Source (signal) Enable. The `SEC_SCTL[n].SEN` bit controls whether the system event source input signal may affect the pending status of the source. Clearing the `SEC_SCTL[n].SEN` bit disables the source input signal from affecting the pending status. Setting `SEC_SCTL[n].SEN` enables the source input signal to affect the pending status.	
		0	Disable
		1	Enable

Table 7-27: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FEN	Fault Enable. The `SEC_SCTL[n].FEN` bit controls whether the SEC may forward an interrupt request to the SEC fault interface as a fault source. This bit does not affect the ability of an interrupt source to set an interrupt as pending. The `SEC_SCTL[n].FEN` bit only affects whether the pending request may be forwarded to the SEC fault interface.	
		0	Disable
		1	Enable
0 (R/W)	IEN	Interrupt Enable. The `SEC_SCTL[n].IEN` bit controls whether the SEC may forward an interrupt request to a core for servicing. This bit does not affect the ability of an interrupt source to set an interrupt as pending.	
		0	Disable
		1	Enable

Source Status Register n

The SEC event source status register (`SEC_SSTAT[n]`) contains bits indicating the status of the corresponding event source n. An event source may be: pending, active, active and pending, or neither pending nor active.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ACT (R/W1C)
Active Source
PND (R/W1C)
Pending Source
ERR (R/W1C)
Error
ERRC (R)
Error Cause
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHID (R)
Channel ID

Figure 7-29: SEC_SSTAT[n] Register Diagram

Table 7-28: SEC_SSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/NW)	CHID	Channel ID. The `SEC_SSTAT[n].CHID` bits indicate the ID of the specific source (from a set of sources sharing one SEC source interface input) that asserted the SEC source interface input. An SEC source interface input may support multiple system sources, in which case the assertion must be qualified by an identifier to determine the channel that generated the assertion. The `SEC_SSTAT[n].CHID` field provides this value in the form of a numeric reference that is mapped to a specific interrupt source. The prioritization for simultaneously asserted sources is according to ID, with 0 being the highest priority. The `SEC_SSTAT[n].CHID` is captured when the SEC source interface input is acknowledged.	
9 (R/W1C)	ACT	Active Source. The `SEC_SSTAT[n].ACT` bit indicates the source has been accepted by a core for servicing, but the service is not yet complete. An `SEC_SSTAT[n].ACT` bit is set by the SEC when the specific system interrupt is acknowledged by the core through the SEC core interface. An `SEC_SSTAT[n].ACT` bit is cleared by the SEC when the core provides interrupt service end indication for the specific system interrupt through the SEC core interface.	
		0	Not Active
		1	Active

Table 7-28: SEC_SSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	PND	Pending Source. The `SEC_SSTAT[n].PND` bit indicates the source has signaled an event request, but the event request has not been (or is not currently being) serviced. A `SEC_SSTAT[n].PND` bit is set by the SEC on detection of an assertion of the corresponding system source input. A `SEC_SSTAT[n].PND` bit is cleared by the SEC when the specific system event is acknowledged by the core through the SEC core interface or by a W1C operation.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the `SEC_SSTAT[n].ERR` bit is asserted, the SEC updates `SEC_SSTAT[n].ERRC` field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. When the error type is end error, the status indicates that an end was received for a source while the `SEC_SSTAT[n].ACT` bit was not set.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. The `SEC_SSTAT[n].ERR` bit indicates an error for a specific system interrupt source. When the `SEC_SSTAT[n].ERR` bit is set, the SEC updates the `SEC_SSTAT[n].ERRC` field to the value of the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of the `SEC_SSTAT[n].ERR` bit.	
		0	No Error
		1	Error Occurred

GIC Overview

The generic interrupt controller (GIC) provides an interface to the uniprocessor Cortex A5 core in the processor and collects up to 270 interrupt requests from all processor system sources. In addition, the GIC also supports eight software generated interrupts that are internal to the Cortex A5 core and not connected to the SECs. All GIC interrupts are also connected to the SEC in the same order.

Each interrupt can be configured as a normal or a secure interrupt. Software force registers and software priority masking are also supported. The "Register Descriptions" section in this chapter provide brief descriptions of these ARM-based registers. For complete information refer to the *ARM® Generic Interrupt Controller Architecture version 1.0 Architecture Specification*.

GIC Functional Description

The GIC splits logically into a GICPORT0 (distributor block) and one GICPORT1 (CPU interface blocks).

General Interrupt Controller Port0 (GIC Distributer)

The distributor block provides a programming interface to perform the following tasks.

- Globally enable the forwarding of interrupts to the CPU interfaces
- Enable or disable each interrupt
- Set the priority level of each interrupt
- Set the target processor list of each interrupt
- Set each peripheral interrupt to be level-sensitive or edge-triggered
- Set each interrupt as either Group 0 or Group 1
- Forward an SGI to one or more target processors

In addition, the Distributor provides:

- Visibility of the state of each interrupt
- A mechanism for software to set or clear the pending state of a peripheral interrupt.

General Interrupt Controller Port1 (GIC CPU)

GICPORT1 (CPU interface) block performs priority masking and preemption handling for a connected processor in the system. GICPORT1 supports 8 SGIs (software generated interrupts) and 262 SPIs (shared peripheral interrupts).

Each CPU interface provides a programming interface to perform the following tasks.

- Enable signaling of interrupt requests to the processor
- Acknowledge interrupts
- Indicate that interrupt processing is complete
- Set interrupt priority masks for the processor
- Define the preemption policy for the processor
- Determine the highest priority pending interrupt for the processor

GIC Block Diagram

The *GIC Block Diagram* shows the event management architecture.

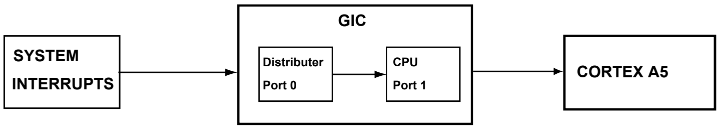


Figure 7-30: GIC Block Diagram

ADSP-SC58x GICDST Register List

GIC Distributor Port

Table 7-29: ADSP-SC58x GICDST Register List

Name	Description
GICDST_EN	GIC Port 0 Enable
GICDST_SGI_PRIO[n]	Software Generated Interrupt Priority Register
GICDST_SPI_PRIO[n]	Shared Peripheral Interrupt Priority Register
GICDST_SGI_ACTIVE	Software Generated Interrupt Active Register
GICDST_SGI_CTL	Software Generated Interrupt Control Register
GICDST_SGI_PND_CLR	Software Generated Interrupt Clear-Pending Register
GICDST_SGI_PND_SET	Software Generated Interrupt Pending Set Register
GICDST_SGI_SECURITY	Software Generated Interrupt Security Register
GICDST_SPI[n]	Shared Peripheral Interrupt Register
GICDST_SPI_ACTIVE[n]	Shared Peripheral Interrupt Active Register
GICDST_SPI_CFG[n]	Shared Peripheral Interrupt Configuration Register
GICDST_SPI_EN_CLR[n]	Shared Peripheral Interrupt Enable Clear Register
GICDST_SPI_EN_SET[n]	Shared Peripheral Interrupt Enable Set Register
GICDST_SPI_PND_CLR[n]	Shared Peripheral Interrupt Pending Clear Register
GICDST_SPI_PND_SET[n]	Shared Peripheral Interrupt Pending Set Register
GICDST_SPI_SECURITY[n]	Shared Peripheral Interrupt Security Register
GICDST_SPI_TRGT[n]	Shared Peripheral Interrupt Processor Targets Register

ADSP-SC58x GICCPU Register List

GIC CPU Port

Table 7-30: ADSP-SC58x GICCPU Register List

Name	Description
GICCPU_BIN_PT_ALIAS	Aliased Binary Point Register (ICCABPR)
GICCPU_BIN_PT	Binary Point Register (ICCBPR)
GICCPU_CTL	CPU Interface Control Register (ICCICR)
GICCPU_EOI	End of Interrupt Register (ICCEOIR)
GICCPU_PND_HI	Highest Pending Interrupt Register (ICCHPIR)
GICCPU_INT_ACK	Interrupt Acknowledge Register (ICCIAR)
GICCPU_PRIO_MSK	Priority Mask Register (ICCIPMR)
GICCPU_RUN_PRIO	Running Priority Register (ICCRPR)

ADSP-SC58x GICDST Register Descriptions

GIC Distributor Port (GICDST) contains the following registers.

Table 7-31: ADSP-SC58x GICDST Register List

Name	Description
GICDST_EN	GIC Port 0 Enable
GICDST_SGI_PRIO[n]	Software Generated Interrupt Priority Register
GICDST_SPI_PRIO[n]	Shared Peripheral Interrupt Priority Register
GICDST_SGI_ACTIVE	Software Generated Interrupt Active Register
GICDST_SGI_CTL	Software Generated Interrupt Control Register
GICDST_SGI_PND_CLR	Software Generated Interrupt Clear-Pending Register
GICDST_SGI_PND_SET	Software Generated Interrupt Pending Set Register
GICDST_SGI_SECURITY	Software Generated Interrupt Security Register
GICDST_SPI[n]	Shared Peripheral Interrupt Register
GICDST_SPI_ACTIVE[n]	Shared Peripheral Interrupt Active Register
GICDST_SPI_CFG[n]	Shared Peripheral Interrupt Configuration Register
GICDST_SPI_EN_CLR[n]	Shared Peripheral Interrupt Enable Clear Register
GICDST_SPI_EN_SET[n]	Shared Peripheral Interrupt Enable Set Register
GICDST_SPI_PND_CLR[n]	Shared Peripheral Interrupt Pending Clear Register
GICDST_SPI_PND_SET[n]	Shared Peripheral Interrupt Pending Set Register
GICDST_SPI_SECURITY[n]	Shared Peripheral Interrupt Security Register
GICDST_SPI_TRGT[n]	Shared Peripheral Interrupt Processor Targets Register

GIC Port 0 Enable

The `GICDST_EN` register enables global monitoring of the peripheral interrupt signals and forwarding pending interrupts to the CPU interfaces.

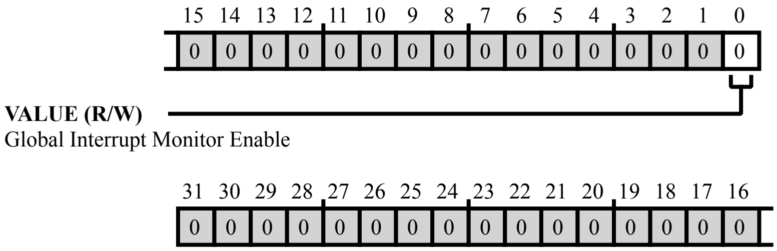

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Global Interrupt Monitor Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-31: GICDST_EN Register Diagram

Table 7-32: GICDST_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALUE	Global Interrupt Monitor Enable. The `GICDST_EN.VALUE` bit field enables global monitoring of the peripheral interrupt signals and forwarding pending interrupts to the CPU interfaces.

Software Generated Interrupt Priority Register

The `GICDST_SGI_PRIO[n]` register provides the 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

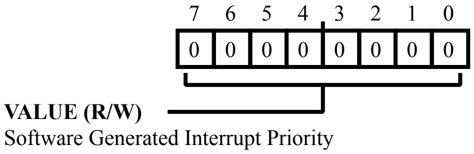


Figure 7-32: GICDST_SGI_PRIO[n] Register Diagram

Table 7-33: GICDST_SGI_PRIO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Software Generated Interrupt Priority. The `GICDST_SGI_PRIO[n].VALUE` bit field contains the 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

Shared Peripheral Interrupt Priority Register

The `GICDST_SPI_PRIO[n]` registers provide an 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

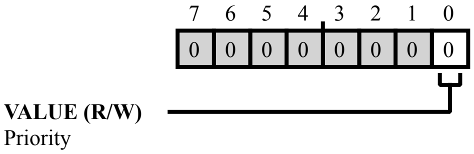


Figure 7-33: GICDST_SPI_PRIO[n] Register Diagram

Table 7-34: GICDST_SPI_PRIO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALUE	Priority. The `GICDST_SPI_PRIO[n].VALUE` bit field stores the priority of the corresponding interrupt (byte offset 3 to Byte offset 0).

Software Generated Interrupt Active Register

The `GICDST_SGI_ACTIVE` registers provide a Set-active bit for each interrupt that the GIC supports. Writing to a Set-active bit Activates the corresponding interrupt. These registers are used when preserving and restoring GIC state.

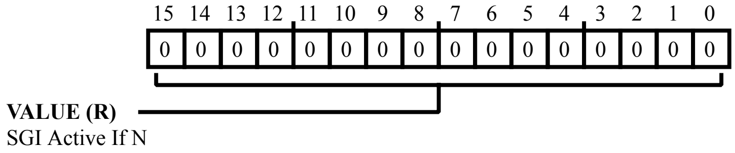


Figure 7-34: GICDST_SGI_ACTIVE Register Diagram

Table 7-35: GICDST_SGI_ACTIVE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	SGI Active If N. The `GICDST_SGI_ACTIVE.VALUE` bit field provides a Set-active bit for each interrupt that the GIC supports.

Software Generated Interrupt Control Register

The `GICDST_SGI_CTL` register controls the generation of SGIs.It is implementation defined whether this register has any effect when the forwarding of interrupts by Distributor is disabled by the GICD_CTLR settings.

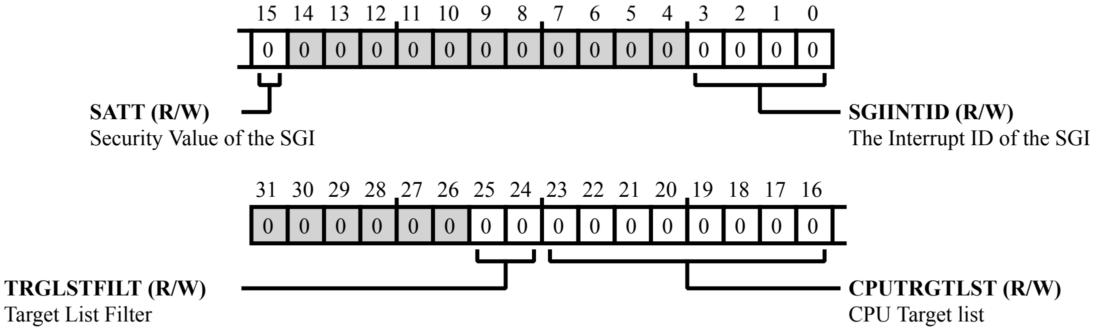

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SATT (R/W)
Security Value of the SGI
SGIINTID (R/W)
The Interrupt ID of the SGI
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRGLSTFILT (R/W)
Target List Filter
CPUTRGTLST (R/W)
CPU Target list

Figure 7-35: GICDST_SGI_CTL Register Diagram

Table 7-36: GICDST_SGI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25:24 (R/W)	TRGLSTFILT	Target List Filter. The `GICDST_SGI_CTL.TRGLSTFILT` bit field determines how the distributor must process the requested SGI.	
		0	Forward the interrupt to the CPU interfaces specified in the CPUTargetList field
		1	Forward the interrupt to all CPU interfaces except that of the processor that requested the interrupt
		2	Forward the interrupt only to the CPU interface of the processor that requested the interrupt
		3	Reserved
23:16 (R/W)	CPUTRGTLST	CPU Target list. When the `GICDST_SGI_CTL.CPUTRGTLST` bit field TargetList Filter = 0b00, defines the CPU interfaces to which the Distributor must forward the interrupt. Each bit of the `GICDST_SGI_CTL.CPUTRGTLST` bit field refers to the corresponding CPU interface, for example CPUTargetList[0] corresponds to CPU interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface. If this field is 0x00 when TargetListFilter is 0b00, the Distributor does not forward the interrupt to any CPU interface.	

Table 7-36: GICDST_SGI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SATT	Security Value of the SGI. The `GICDST_SGI_CTL.SATT` bit is implemented only if the GIC includes the Security Extensions. This field is writable only by a Secure access. Any Non-secure write to the GICD_SGIR generates an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit[15] of the write.	
		0	Forward the SGI specified in the SGIINTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface.
		1	Forward the SGI specified in the SGIINTID field to a specified CPU interfaces only if the SGI is configured as Group 1 on that interface.
3:0 (R/W)	SGIINTID	The Interrupt ID of the SGI.	

Software Generated Interrupt Clear-Pending Register

The `GICDST_SGI_PND_CLR` register provides a clear pending bit for each interrupt supported by the GIC. Writing 1 to a clear-pending bit clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

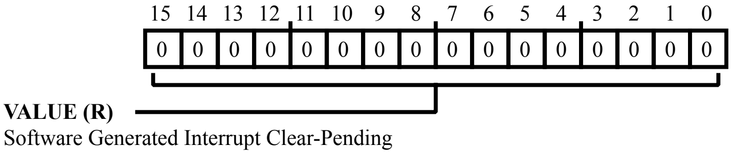

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Software Generated Interrupt Clear-Pending

Figure 7-36: GICDST_SGI_PND_CLR Register Diagram

Table 7-37: GICDST_SGI_PND_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Software Generated Interrupt Clear-Pending. Writing 1 to a clear-pending bit in the `GICDST_SGI_PND_CLR.VALUE` bit field clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

Software Generated Interrupt Pending Set Register

The `GICDST_SGI_PND_SET` register provides a set-pending bit for each interrupt supported by the GIC.

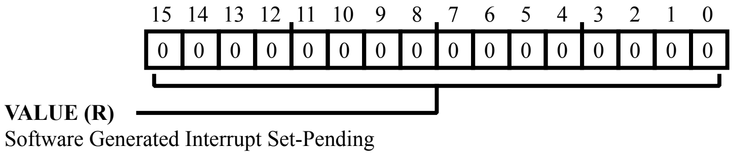

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Software Generated Interrupt Set-Pending

Figure 7-37: GICDST_SGI_PND_SET Register Diagram

Table 7-38: GICDST_SGI_PND_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Software Generated Interrupt Set-Pending. Writing 1 to a Set-pending bit in the `GICDST_SGI_PND_SET.VALUE` bit field sets the status of the corresponding peripheral interrupt to pending. Reading a bit identifies whether the interrupt is pending.

Software Generated Interrupt Security Register

The `GICDST_SGI_SECURITY` registers provide a status bit for each interrupt supported by the GIC. Each bit controls whether the corresponding interrupt is in Group 0 or Group 1. Typically, when used with a processor that implements the ARM Security Extensions, Group 0 interrupts are Secure interrupts, and Group 1 interrupts are Non-secure interrupts,

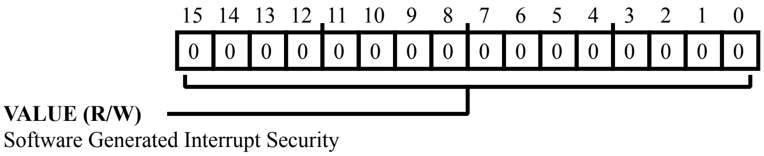

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Software Generated Interrupt Security

Figure 7-38: GICDST_SGI_SECURITY Register Diagram

Table 7-39: GICDST_SGI_SECURITY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Software Generated Interrupt Security. Each bit in the `GICDST_SGI_SECURITY.VALUE` bit field controls whether the corresponding interrupt is in Group 0 or Group 1.

Shared Peripheral Interrupt Register

The `GICDST_SPI[n]` register contains bits that provide the status of the SPI[987:0] inputs.

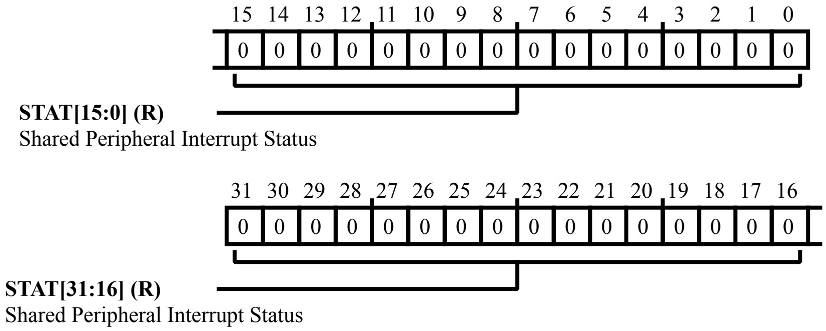

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STAT[15:0] (R)
Shared Peripheral Interrupt Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STAT[31:16] (R)
Shared Peripheral Interrupt Status

Figure 7-39: GICDST_SPI[n] Register Diagram

Table 7-40: GICDST_SPI[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	STAT	Shared Peripheral Interrupt Status. The `GICDST_SPI[n].STAT` bit field returns the status of the SPI[987:0] inputs on the distributor where bit [x] = 0 SPI[x] is low and bit [x] = 1 SPI[x] is high.

Shared Peripheral Interrupt Active Register

The `GICDST_SPI_ACTIVE[n]` register provides an active bit for each interrupt supported by the GIC.

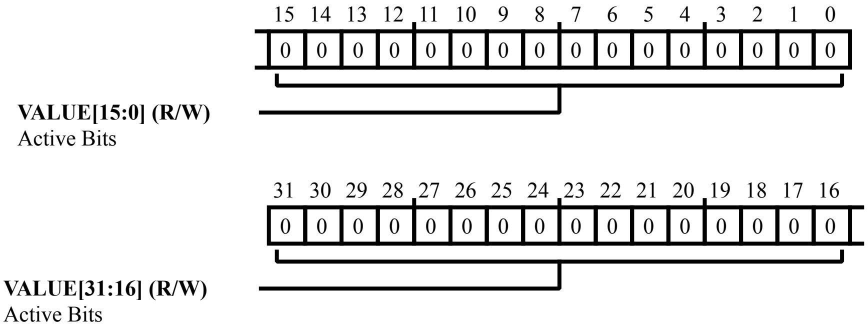

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Active Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Active Bits

Figure 7-40: GICDST_SPI_ACTIVE[n] Register Diagram

Table 7-41: GICDST_SPI_ACTIVE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Active Bits. The `GICDST_SPI_ACTIVE[n].VALUE` bit field contains an Active bit for each interrupt supported by the GIC. Reading an active bit identifies whether the corresponding interrupt is active (=1) or not active (=0).

Shared Peripheral Interrupt Configuration Register

The `GICDST_SPI_CFG[n]` register provides a 2-bit Int_config field for each interrupt supported by the GIC.

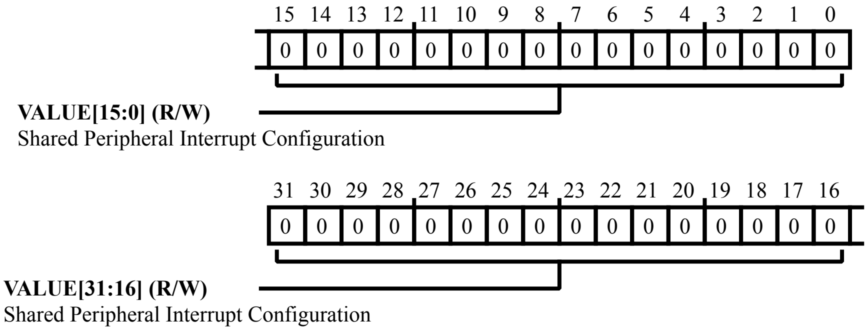

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Configuration
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Configuration

Figure 7-41: GICDST_SPI_CFG[n] Register Diagram

Table 7-42: GICDST_SPI_CFG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Configuration. The `GICDST_SPI_CFG[n].VALUE` bit field identifies whether the corresponding interrupt is: edge-triggered or level-sensitive handled using the 1-N model or using the N-N model

Shared Peripheral Interrupt Enable Clear Register

The `GICDST_SPI_EN_CLR[n]` register provides a clear-enable bit for each interrupt supported by the GIC.

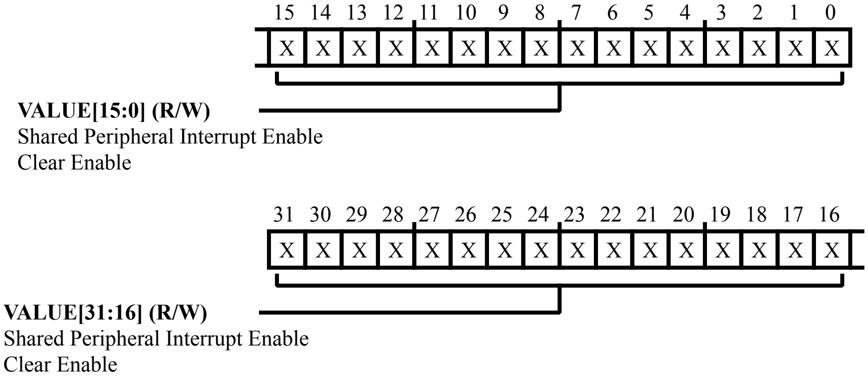

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Enable Clear Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Enable Clear Enable

Figure 7-42: GICDST_SPI_EN_CLR[n] Register Diagram

Table 7-43: GICDST_SPI_EN_CLR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Enable Clear Enable. Writing 1 to a `GICDST_SPI_EN_CLR[n].VALUE` bit disables forwarding of the corresponding interrupt to the CPU interfaces. Reading a bit identifies whether the interrupt is enabled.

Shared Peripheral Interrupt Enable Set Register

The `GICDST_SPI_EN_SET[n]` register provides a set-enable bit for each interrupt supported by the GIC.

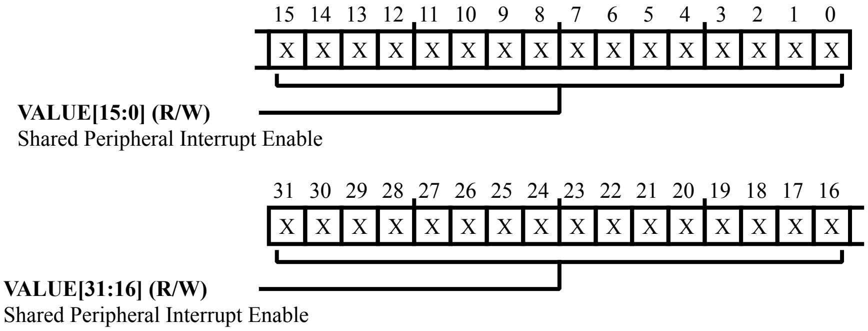

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Enable

Figure 7-43: GICDST_SPI_EN_SET[n] Register Diagram

Table 7-44: GICDST_SPI_EN_SET[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Enable. Writing 1 to a `GICDST_SPI_EN_SET[n].VALUE` bit enables forwarding of the corresponding interrupt to the CPU interfaces. Reading a bit identifies whether the interrupt is enabled.

Shared Peripheral Interrupt Pending Clear Register

The `GICDST_SPI_PND_CLR[n]` register provides a clear-pending bit for each interrupt supported by the GIC.

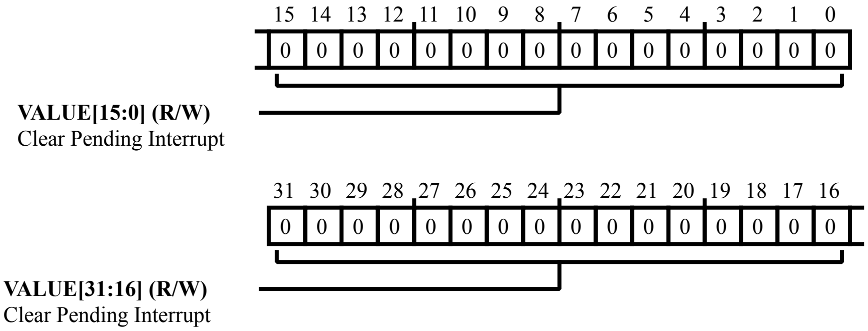

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Clear Pending Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Clear Pending Interrupt

Figure 7-44: GICDST_SPI_PND_CLR[n] Register Diagram

Table 7-45: GICDST_SPI_PND_CLR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clear Pending Interrupt. Writing 1 to a `GICDST_SPI_PND_CLR[n].VALUE` bit clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

Shared Peripheral Interrupt Pending Set Register

The `GICDST_SPI_PND_SET[n]` register provides a set-pending bit for each interrupt supported by the GIC.

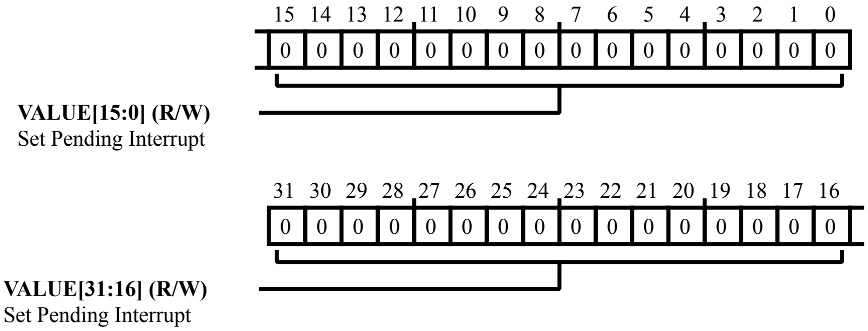

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Set Pending Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Set Pending Interrupt

Figure 7-45: GICDST_SPI_PND_SET[n] Register Diagram

Table 7-46: GICDST_SPI_PND_SET[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Set Pending Interrupt. Writing 1 to a `GICDST_SPI_PND_SET[n].VALUE` bit sets the status of the corresponding peripheral interrupt to pending. Reading a bit identifies whether the interrupt is pending.

Shared Peripheral Interrupt Security Register

The `GICDST_SPI_SECURITY[n]` register provides a security status bit for each interrupt supported by the GIC.

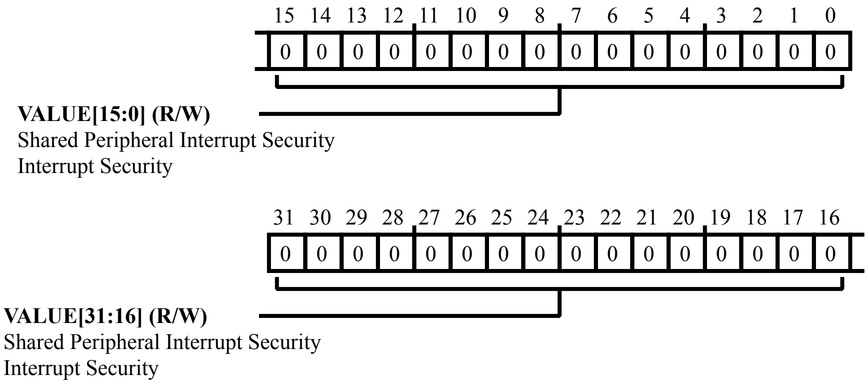

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Security
Interrupt Security
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Security
Interrupt Security

Figure 7-46: GICDST_SPI_SECURITY[n] Register Diagram

Table 7-47: GICDST_SPI_SECURITY[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Security Interrupt Security. The `GICDST_SPI_SECURITY[n].VALUE` bits control the security status of the corresponding interrupt.

Shared Peripheral Interrupt Processor Targets Register

The `GICDST_SPI_TRGT[n]` register provides an 8-bit CPU targets field for each interrupt supported by the GIC.

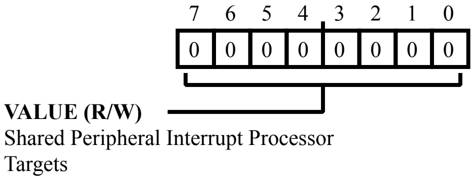


Figure 7-47: GICDST_SPI_TRGT[n] Register Diagram

Table 7-48: GICDST_SPI_TRGT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Shared Peripheral Interrupt Processor Targets. The `GICDST_SPI_TRGT[n].VALUE` bit field stores the list of processors that the interrupt is sent to if it is asserted.

ADSP-SC58x GICCPU Register Descriptions

GIC CPU Port (GICCPU) contains the following registers.

Table 7-49: ADSP-SC58x GICCPU Register List

Name	Description
`GICCPU_BIN_PT_ALIAS`	Aliased Binary Point Register (ICCABPR)
`GICCPU_BIN_PT`	Binary Point Register (ICCBPR)
`GICCPU_CTL`	CPU Interface Control Register (ICCICR)
`GICCPU_EOI`	End of Interrupt Register (ICCEOIR)
`GICCPU_PND_HI`	Highest Pending Interrupt Register (ICCHPIR)
`GICCPU_INT_ACK`	Interrupt Acknowledge Register (ICCIAR)
`GICCPU_PRIO_MSK`	Priority Mask Register (ICCIPMR)
`GICCPU_RUN_PRIO`	Running Priority Register (ICCRPR)

Aliased Binary Point Register (ICCABPR)

The `GICCPU_BIN_PT_ALIAS` register Provides an alias for the non secure `GICCPU_BIN_PT` register.

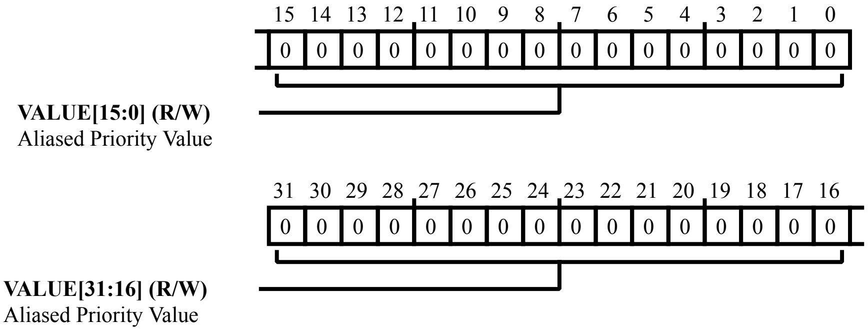

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Aliased Priority Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Aliased Priority Value

Figure 7-48: GICCPU_BIN_PT_ALIAS Register Diagram

Table 7-50: GICCPU_BIN_PT_ALIAS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Aliased Priority Value.

Binary Point Register (ICCBPR)

The `GICCPU_BIN_PT` register defines the point at which the priority value fields split into two parts, the group priority field and the sub-priority field. The group priority field is used to determine interrupt preemption.

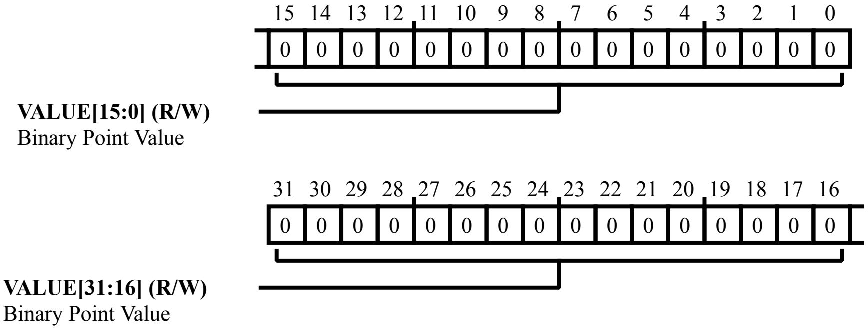

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Binary Point Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Binary Point Value

Figure 7-49: GICCPU_BIN_PT Register Diagram

Table 7-51: GICCPU_BIN_PT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Binary Point Value. The `GICCPU_BIN_PT.VALUE` bit field defines the point at which the priority value fields split into two parts.

CPU Interface Control Register (ICCICR)

The `GICCPU_CTL` register enables the signaling of interrupts to the target processors. In a GIC that implements the Security Extensions, provides additional global controls for handling Secure interrupts.

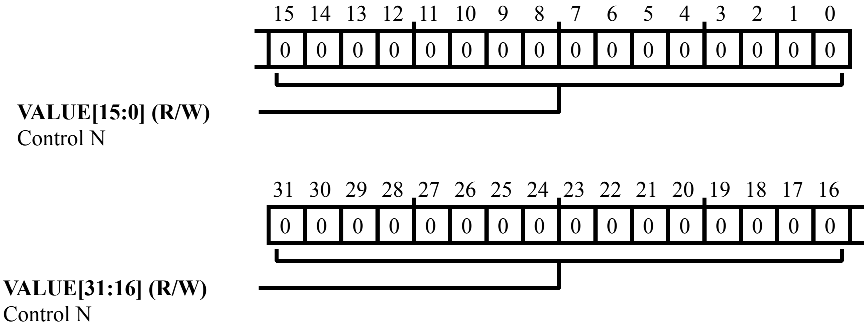

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Control N
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Control N

Figure 7-50: GICCPU_CTL Register Diagram

Table 7-52: GICCPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Control N.

End of Interrupt Register (ICCEOIR)

A processor writes to the `GICCPU_EOI` register to inform the CPU interface that it has completed its interrupt service routine for the specified interrupt.

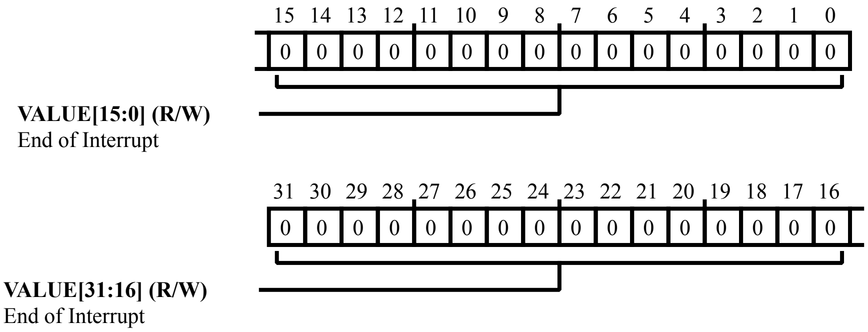

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
End of Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
End of Interrupt

Figure 7-51: GICCPU_EOI Register Diagram

Table 7-53: GICCPU_EOI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	End of Interrupt. The `GICCPU_EOI.VALUE` bit field indicates to the CPU interface that it has completed its interrupt service routine for the specified interrupt.

Highest Pending Interrupt Register (ICCHPIR)

The `GICCPU_PND_HI` register indicates the Interrupt ID, and processor ID if appropriate, of the pending interrupt with the highest priority on the CPU interface.

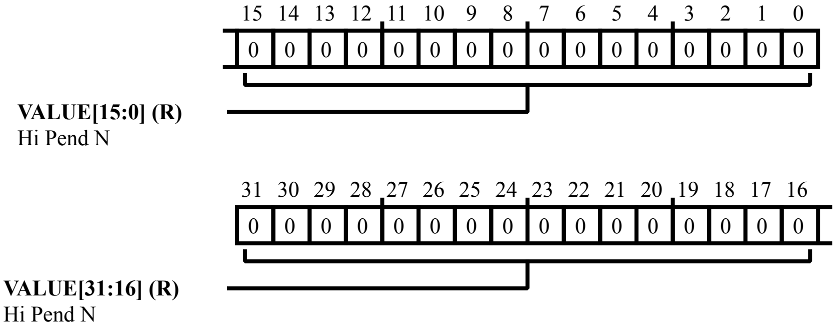

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Hi Pend N
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Hi Pend N

Figure 7-52: GICCPU_PND_HI Register Diagram

Table 7-54: GICCPU_PND_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Hi Pend N. Highest Pending Interrupt Register (ICCHPIR)

Interrupt Acknowledge Register (ICCIAR)

The processor reads the `GICCPU_INT_ACK` register to obtain the interrupt ID of the signaled interrupt. This read acts as an acknowledge for the interrupt.

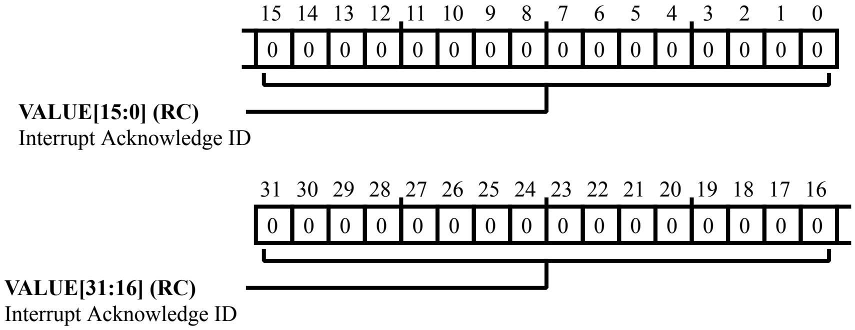

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (RC)
Interrupt Acknowledge ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (RC)
Interrupt Acknowledge ID

Figure 7-53: GICCPU_INT_ACK Register Diagram

Table 7-55: GICCPU_INT_ACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RC/NW)	VALUE	Interrupt Acknowledge ID. The `GICCPU_INT_ACK.VALUE` bit field contains the interrupt ID of the signaled interrupt.

Priority Mask Register (ICCIPMR)

The GICCPU_PRIO_MSK register provides an interrupt priority filter. Only interrupts with higher priority than the value in this register can be signaled to the processor.

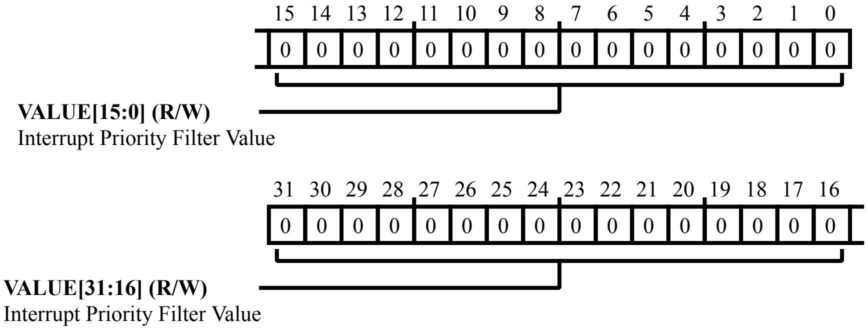

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Interrupt Priority Filter Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Interrupt Priority Filter Value

Figure 7-54: GICCPU_PRIO_MSK Register Diagram

Table 7-56: GICCPU_PRIO_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Interrupt Priority Filter Value. The `GICCPU_PRIO_MSK.VALUE` bit field contains the interrupt priority filter value.

Running Priority Register (ICCRPR)

The GICCPU_RUN_PRIO register indicates the priority of the highest priority interrupt that is active on the CPU interface.

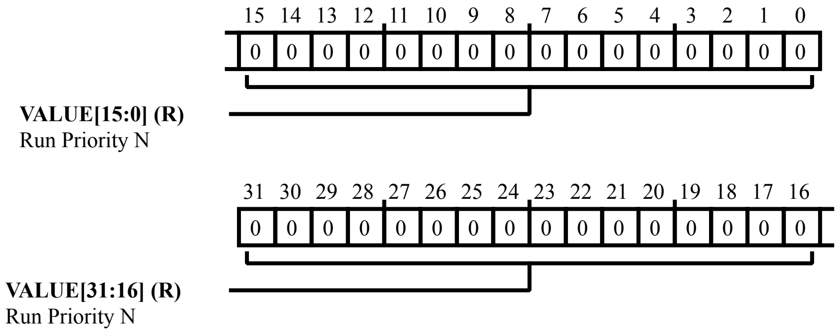


Figure 7-55: GICCPU_RUN_PRIO Register Diagram

Table 7-57: GICCPU_RUN_PRIO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Run Priority N. The `GICCPU_RUN_PRIO.VALUE` bit field contains the priority value of the highest priority interrupt that is active on the CPU interface. If there is no active interrupt on the CPU interface, and the GIC implements 8-bit priority fields, a read of this register returns the value 0xFF, corresponding to the lowest possible interrupt priority. If the GIC implements priority fields of less than 8 bits, the read might return the register reset value of 0xFF, or might return a value corresponding to the lowest possible interrupt priority. Software cannot determine the number of implemented priority bits from a read of this register.

ADSP-SC58x GICDST Register Descriptions

GIC Distributor Port (GICDST) contains the following registers.

Table 7-58: ADSP-SC58x GICDST Register List

Name	Description
GICDST_EN	GIC Port 0 Enable
GICDST_SGI_PRIO[n]	Software Generated Interrupt Priority Register
GICDST_SPI_PRIO[n]	Shared Peripheral Interrupt Priority Register
GICDST_SGI_ACTIVE	Software Generated Interrupt Active Register
GICDST_SGI_CTL	Software Generated Interrupt Control Register
GICDST_SGI_PND_CLR	Software Generated Interrupt Clear-Pending Register

Table 7-58: ADSP-SC58x GICDST Register List (Continued)

Name	Description
GICDST_SGI_PND_SET	Software Generated Interrupt Pending Set Register
GICDST_SGI_SECURITY	Software Generated Interrupt Security Register
GICDST_SPI[n]	Shared Peripheral Interrupt Register
GICDST_SPI_ACTIVE[n]	Shared Peripheral Interrupt Active Register
GICDST_SPI_CFG[n]	Shared Peripheral Interrupt Configuration Register
GICDST_SPI_EN_CLR[n]	Shared Peripheral Interrupt Enable Clear Register
GICDST_SPI_EN_SET[n]	Shared Peripheral Interrupt Enable Set Register
GICDST_SPI_PND_CLR[n]	Shared Peripheral Interrupt Pending Clear Register
GICDST_SPI_PND_SET[n]	Shared Peripheral Interrupt Pending Set Register
GICDST_SPI_SECURITY[n]	Shared Peripheral Interrupt Security Register
GICDST_SPI_TRGT[n]	Shared Peripheral Interrupt Processor Targets Register

GIC Port 0 Enable

The GICDST_EN register enables global monitoring of the peripheral interrupt signals and forwarding pending interrupts to the CPU interfaces.

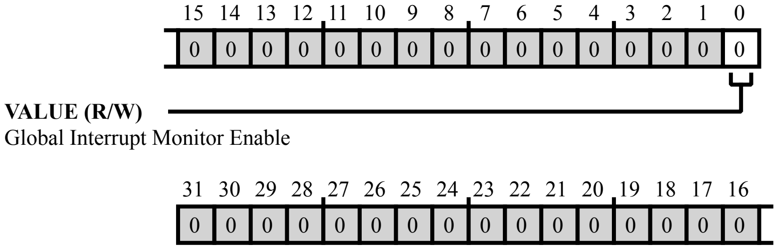

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Global Interrupt Monitor Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7-56: GICDST_EN Register Diagram

Table 7-59: GICDST_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALUE	Global Interrupt Monitor Enable. The `GICDST_EN.VALUE` bit field enables global monitoring of the peripheral interrupt signals and forwarding pending interrupts to the CPU interfaces.

Software Generated Interrupt Priority Register

The `GICDST_SGI_PRIO[n]` register provides the 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

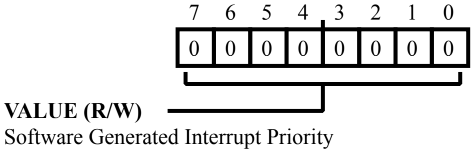


Figure 7-57: GICDST_SGI_PRIO[n] Register Diagram

Table 7-60: GICDST_SGI_PRIO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Software Generated Interrupt Priority. The `GICDST_SGI_PRIO[n].VALUE` bit field contains the 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

Shared Peripheral Interrupt Priority Register

The `GICDST_SPI_PRIO[n]` registers provide an 8-bit priority field for each interrupt supported by the GIC. This field stores the priority of the corresponding interrupt.

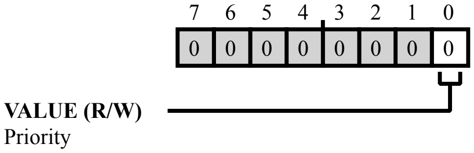


Figure 7-58: GICDST_SPI_PRIO[n] Register Diagram

Table 7-61: GICDST_SPI_PRIO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALUE	Priority. The `GICDST_SPI_PRIO[n].VALUE` bit field stores the priority of the corresponding interrupt (byte offset 3 to Byte offset 0).

Software Generated Interrupt Active Register

The `GICDST_SGI_ACTIVE` registers provide a Set-active bit for each interrupt that the GIC supports. Writing to a Set-active bit Activates the corresponding interrupt. These registers are used when preserving and restoring GIC state.

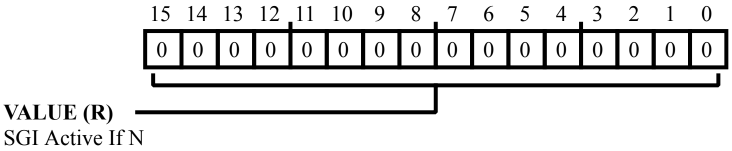


Figure 7-59: GICDST_SGI_ACTIVE Register Diagram

Table 7-62: GICDST_SGI_ACTIVE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	SGI Active If N. The `GICDST_SGI_ACTIVE.VALUE` bit field provides a Set-active bit for each interrupt that the GIC supports.

Software Generated Interrupt Control Register

The `GICDST_SGI_CTL` register controls the generation of SGIs.It is implementation defined whether this register has any effect when the forwarding of interrupts by Distributor is disabled by the GICD_CTLR settings.

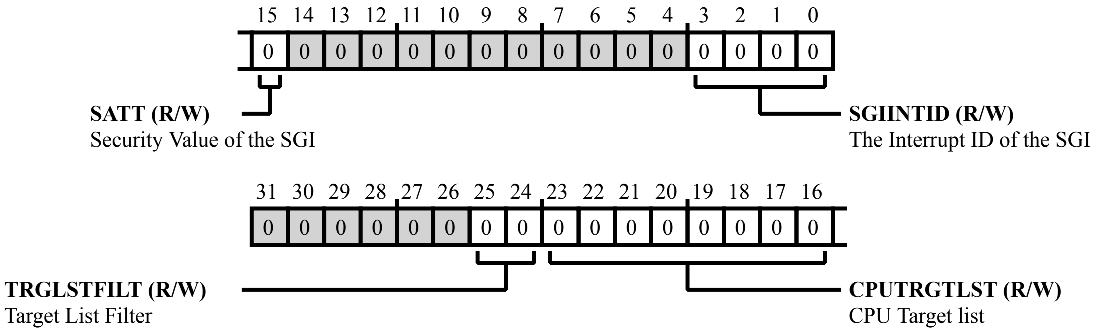

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SATT (R/W)
Security Value of the SGI
SGIINTID (R/W)
The Interrupt ID of the SGI
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRGLSTFILT (R/W)
Target List Filter
CPUTRGTLST (R/W)
CPU Target list

Figure 7-60: GICDST_SGI_CTL Register Diagram

Table 7-63: GICDST_SGI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25:24 (R/W)	TRGLSTFILT	Target List Filter. The `GICDST_SGI_CTL.TRGLSTFILT` bit field determines how the distributor must process the requested SGI.	
		0	Forward the interrupt to the CPU interfaces specified in the CPUTargetList field
		1	Forward the interrupt to all CPU interfaces except that of the processor that requested the interrupt
		2	Forward the interrupt only to the CPU interface of the processor that requested the interrupt
		3	Reserved
23:16 (R/W)	CPUTRGTLST	CPU Target list. When the `GICDST_SGI_CTL.CPUTRGTLST` bit field TargetList Filter = 0b00, defines the CPU interfaces to which the Distributor must forward the interrupt. Each bit of the `GICDST_SGI_CTL.CPUTRGTLST` bit field refers to the corresponding CPU interface, for example CPUTargetList[0] corresponds to CPU interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface. If this field is 0x00 when TargetListFilter is 0b00, the Distributor does not forward the interrupt to any CPU interface.	

Table 7-63: GICDST_SGI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SATT	Security Value of the SGI. The `GICDST_SGI_CTL.SATT` bit is implemented only if the GIC includes the Security Extensions. This field is writable only by a Secure access. Any Non-secure write to the GICD_SGIR generates an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit[15] of the write.	
		0	Forward the SGI specified in the SGIINTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface.
		1	Forward the SGI specified in the SGIINTID field to a specified CPU interfaces only if the SGI is configured as Group 1 on that interface.
3:0 (R/W)	SGIINTID	The Interrupt ID of the SGI.	

Software Generated Interrupt Clear-Pending Register

The `GICDST_SGI_PND_CLR` register provides a clear pending bit for each interrupt supported by the GIC. Writing 1 to a clear-pending bit clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

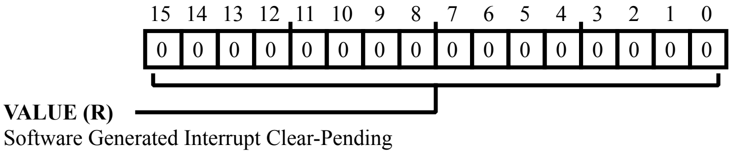

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Software Generated Interrupt Clear-Pending

Figure 7-61: GICDST_SGI_PND_CLR Register Diagram

Table 7-64: GICDST_SGI_PND_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Software Generated Interrupt Clear-Pending. Writing 1 to a clear-pending bit in the `GICDST_SGI_PND_CLR.VALUE` bit field clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

Software Generated Interrupt Pending Set Register

The `GICDST_SGI_PND_SET` register provides a set-pending bit for each interrupt supported by the GIC.

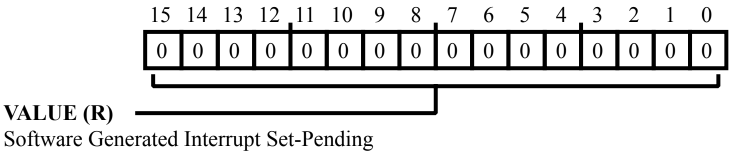

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Software Generated Interrupt Set-Pending

Figure 7-62: GICDST_SGI_PND_SET Register Diagram

Table 7-65: GICDST_SGI_PND_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Software Generated Interrupt Set-Pending. Writing 1 to a Set-pending bit in the `GICDST_SGI_PND_SET.VALUE` bit field sets the status of the corresponding peripheral interrupt to pending. Reading a bit identifies whether the interrupt is pending.

Software Generated Interrupt Security Register

The `GICDST_SGI_SECURITY` registers provide a status bit for each interrupt supported by the GIC. Each bit controls whether the corresponding interrupt is in Group 0 or Group 1. Typically, when used with a processor that implements the ARM Security Extensions, Group 0 interrupts are Secure interrupts, and Group 1 interrupts are Non-secure interrupts,

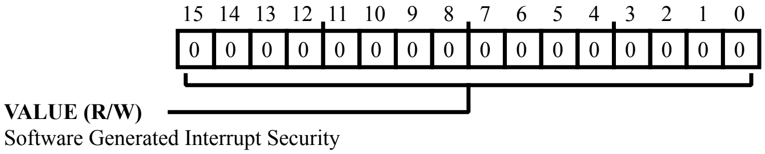

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Software Generated Interrupt Security

Figure 7-63: GICDST_SGI_SECURITY Register Diagram

Table 7-66: GICDST_SGI_SECURITY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Software Generated Interrupt Security. Each bit in the `GICDST_SGI_SECURITY.VALUE` bit field controls whether the corresponding interrupt is in Group 0 or Group 1.

Shared Peripheral Interrupt Register

The `GICDST_SPI[n]` register contains bits that provide the status of the SPI[987:0] inputs.

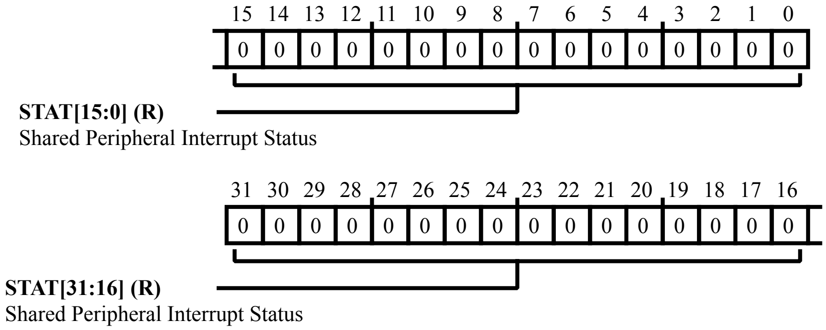

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STAT[15:0] (R)
Shared Peripheral Interrupt Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STAT[31:16] (R)
Shared Peripheral Interrupt Status

Figure 7-64: GICDST_SPI[n] Register Diagram

Table 7-67: GICDST_SPI[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	STAT	Shared Peripheral Interrupt Status. The `GICDST_SPI[n].STAT` bit field returns the status of the SPI[987:0] inputs on the distributor where bit [x] = 0 SPI[x] is low and bit [x] = 1 SPI[x] is high.

Shared Peripheral Interrupt Active Register

The `GICDST_SPI_ACTIVE[n]` register provides an active bit for each interrupt supported by the GIC.

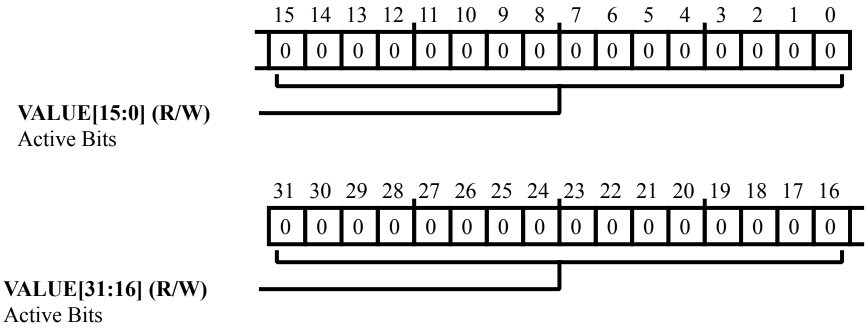

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Active Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Active Bits

Figure 7-65: GICDST_SPI_ACTIVE[n] Register Diagram

Table 7-68: GICDST_SPI_ACTIVE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Active Bits. The `GICDST_SPI_ACTIVE[n].VALUE` bit field contains an Active bit for each interrupt supported by the GIC. Reading an active bit identifies whether the corresponding interrupt is active (=1) or not active (=0).

Shared Peripheral Interrupt Configuration Register

The `GICDST_SPI_CFG[n]` register provides a 2-bit Int_config field for each interrupt supported by the GIC.

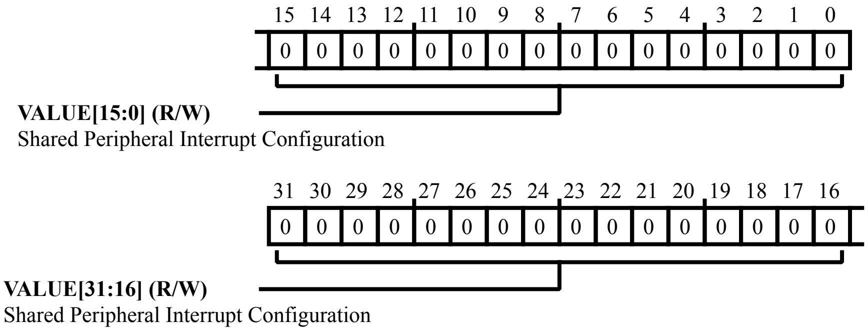

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Configuration
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Configuration

Figure 7-66: GICDST_SPI_CFG[n] Register Diagram

Table 7-69: GICDST_SPI_CFG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Configuration. The `GICDST_SPI_CFG[n].VALUE` bit field identifies whether the corresponding interrupt is: edge-triggered or level-sensitive handled using the 1-N model or using the N-N model

Shared Peripheral Interrupt Enable Clear Register

The `GICDST_SPI_EN_CLR[n]` register provides a clear-enable bit for each interrupt supported by the GIC.

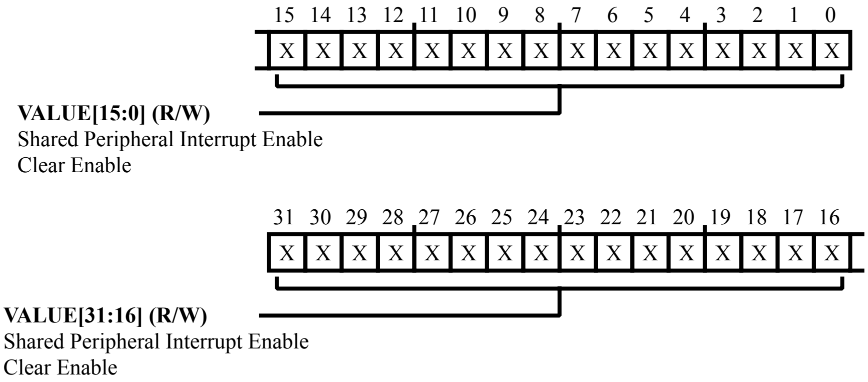

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Enable Clear Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Enable Clear Enable

Figure 7-67: GICDST_SPI_EN_CLR[n] Register Diagram

Table 7-70: GICDST_SPI_EN_CLR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Enable Clear Enable. Writing 1 to a `GICDST_SPI_EN_CLR[n].VALUE` bit disables forwarding of the corresponding interrupt to the CPU interfaces. Reading a bit identifies whether the interrupt is enabled.

Shared Peripheral Interrupt Enable Set Register

The `GICDST_SPI_EN_SET[n]` register provides a set-enable bit for each interrupt supported by the GIC.

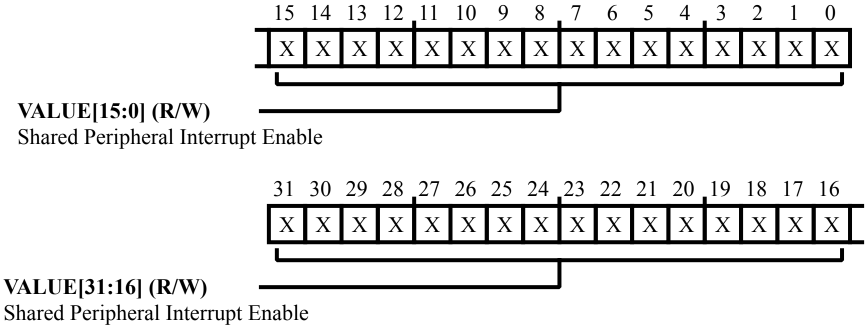

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Enable

Figure 7-68: GICDST_SPI_EN_SET[n] Register Diagram

Table 7-71: GICDST_SPI_EN_SET[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Enable. Writing 1 to a `GICDST_SPI_EN_SET[n].VALUE` bit enables forwarding of the corresponding interrupt to the CPU interfaces. Reading a bit identifies whether the interrupt is enabled.

Shared Peripheral Interrupt Pending Clear Register

The `GICDST_SPI_PND_CLR[n]` register provides a clear-pending bit for each interrupt supported by the GIC.

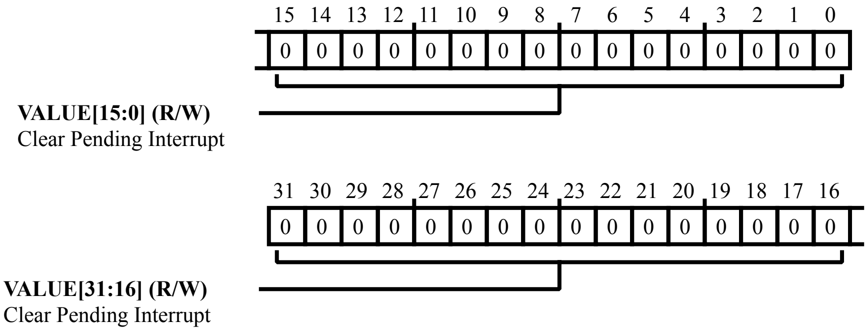

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Clear Pending Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Clear Pending Interrupt

Figure 7-69: GICDST_SPI_PND_CLR[n] Register Diagram

Table 7-72: GICDST_SPI_PND_CLR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clear Pending Interrupt. Writing 1 to a `GICDST_SPI_PND_CLR[n].VALUE` bit clears the pending status of the corresponding peripheral interrupt. Reading a bit identifies whether the interrupt is pending.

Shared Peripheral Interrupt Pending Set Register

The `GICDST_SPI_PND_SET[n]` register provides a set-pending bit for each interrupt supported by the GIC.

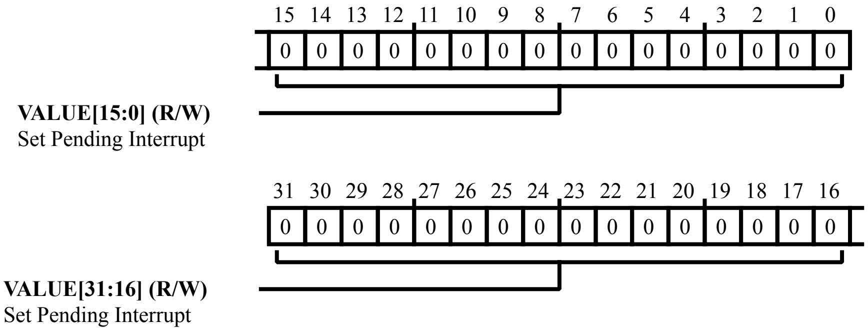

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Set Pending Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Set Pending Interrupt

Figure 7-70: GICDST_SPI_PND_SET[n] Register Diagram

Table 7-73: GICDST_SPI_PND_SET[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Set Pending Interrupt. Writing 1 to a `GICDST_SPI_PND_SET[n].VALUE` bit sets the status of the corresponding peripheral interrupt to pending. Reading a bit identifies whether the interrupt is pending.

Shared Peripheral Interrupt Security Register

The `GICDST_SPI_SECURITY[n]` register provides a security status bit for each interrupt supported by the GIC.

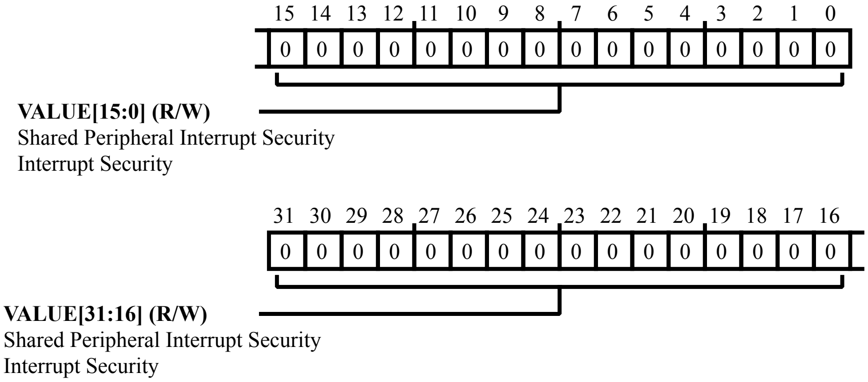

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Shared Peripheral Interrupt Security
Interrupt Security
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Shared Peripheral Interrupt Security
Interrupt Security

Figure 7-71: GICDST_SPI_SECURITY[n] Register Diagram

Table 7-74: GICDST_SPI_SECURITY[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Shared Peripheral Interrupt Security Interrupt Security. The `GICDST_SPI_SECURITY[n].VALUE` bits control the security status of the corresponding interrupt.

Shared Peripheral Interrupt Processor Targets Register

The `GICDST_SPI_TRGT[n]` register provides an 8-bit CPU targets field for each interrupt supported by the GIC.

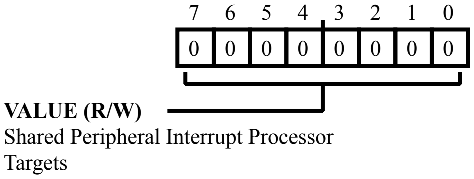

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
VALUE (R/W)
Shared Peripheral Interrupt Processor Targets

Figure 7-72: GICDST_SPI_TRGT[n] Register Diagram

Table 7-75: GICDST_SPI_TRGT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Shared Peripheral Interrupt Processor Targets. The `GICDST_SPI_TRGT[n].VALUE` bit field stores the list of processors that the interrupt is sent to if it is asserted.

8 Trigger Routing Unit (TRU)

The TRU provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering
- Synchronization of concurrent activities

TRU Features

The TRU supports the following features:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes. Once a DMA channel completes data transfer, it can act as a Trigger Master and signal an internal trigger pulse to the programmed Trigger Slave which can also be another DMA channel. The Slave Trigger connected to the DMA channel kicks off the DMA transfer automatically. None of this requires core intervention once the initialization is done.
- Software triggers. The best use of triggers is to minimize core intervention. It is also possible to initiate a trigger pulse to a Trigger Slave, in the software.
- Synchronization of concurrent activities. A single Trigger Master can initiate a trigger pulse to multiple Trigger Slaves so that several system level activities can be synchronized on an internally or externally generated event.
- Configuration protection through register-level lock bits and global lock indication

TRU Functional Description

The following sections provide a description of the TRU.

ADSP-SC58x TRU Register List

The Trigger Routing Unit (TRU) provides simple sequence control of distributed modules without the penalties associated with core intervention (for example, interrupt overhead). The TRU receives trigger inputs from all master trigger inputs (MTI) and the TRU master trigger register (`TRU_MTR`). Based on these inputs, the TRU logic generates trigger outputs that initiate slave operations in the processor core and peripherals. A set of registers governs TRU operations. For more information on TRU functionality, see the TRU register descriptions.

Table 8-1: ADSP-SC58x TRU Register List

Name	Description
`TRU_ERRADDR`	Error Address Register
`TRU_GCTL`	Global Control Register
`TRU_MTR`	Master Trigger Register
`TRU_SSR[n]`	Slave Select Register
`TRU_STAT`	Status Information Register

ADSP-SC58x TRU Interrupt List

Table 8-2: ADSP-SC58x TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
136	TRU0_SLV4	TRU0 Interrupt 4, Core ID = 1 only	Edge	
137	TRU0_SLV5	TRU0 Interrupt 5, Core ID = 1 only	Edge	
138	TRU0_SLV6	TRU0 Interrupt 6, Core ID = 1 only	Edge	
139	TRU0_SLV7	TRU0 Interrupt 7, Core ID = 1 only	Edge	
140	TRU0_SLV8	TRU0 Interrupt 8, Core ID = 2 only	Edge	
141	TRU0_SLV9	TRU0 Interrupt 9, Core ID = 2 only	Edge	
142	TRU0_SLV10	TRU0 Interrupt 10, Core ID = 2 only	Edge	
143	TRU0_SLV11	TRU0 Interrupt 11, Core ID = 2 only	Edge	
280	TRU0_SLV0	TRU0 Interrupt 0, Core ID = 0 only	Edge	
281	TRU0_SLV1	TRU0 Interrupt 1, Core ID = 0 only	Edge	
282	TRU0_SLV2	TRU0 Interrupt 2, Core ID = 0 only	Edge	
283	TRU0_SLV3	TRU0 Interrupt 3, Core ID = 0 only	Edge	

ADSP-SC58x Trigger List

Table 8-3: ADSP-SC58x Trigger List Masters

Trigger ID	Name	Description	Sensitivity
0	`RESERVED_TRIG0`	NC_RES0	
1	`CGU0_EVT`	CGU0 Event	Edge
2	`CGU1_EVT`	CGU1 Event	Edge
3	`C0_SNDEVT`	Core0 Send Event	
4	`C0_WFI`	Core0 Wait For Interrupt	
5	`C0_WFE`	Core0 Wait For Event	
6	`TIMER0_TMR0_MST`	TIMER0 Timer 0	Edge
7	`TIMER0_TMR1_MST`	TIMER0 Timer 1	Edge
8	`TIMER0_TMR2_MST`	TIMER0 Timer 2	Edge
9	`TIMER0_TMR3_MST`	TIMER0 Timer 3	Edge
10	`TIMER0_TMR4_MST`	TIMER0 Timer 4	Edge
11	`TIMER0_TMR5_MST`	TIMER0 Timer 5	Edge
12	`TIMER0_TMR6_MST`	TIMER0 Timer 6	Edge
13	`TIMER0_TMR7_MST`	TIMER0 Timer 7	Edge
14	`PINT0_BLOCK`	PINT0 Pin Interrupt Block	Level
15	`PINT1_BLOCK`	PINT1 Pin Interrupt Block	Level
16	`PINT2_BLOCK`	PINT2 Pin Interrupt Block	Level
17	`PINT3_BLOCK`	PINT3 Pin Interrupt Block	Level
18	`PINT4_BLOCK`	PINT4 Pin Interrupt Block	Level
19	`PINT5_BLOCK`	PINT5 Pin Interrupt Block	Level
20	`CNT0_STAT`	CNT0 Status	Level
21	`PWM0_SYNC`	PWM0 PWMTMR Grouped	Edge
22	`PWM1_SYNC`	PWM1 PWMTMR Grouped	Edge
23	`PWM2_SYNC`	PWM2 PWMTMR Grouped	Edge
24	`SPORT0_A_DMA`	SPORT0 Channel A DMA	Edge
25	`SPORT0_B_DMA`	SPORT0 Channel B DMA	Edge
26	`SPORT1_A_DMA`	SPORT1 Channel A DMA	Edge
27	`SPORT1_B_DMA`	SPORT1 Channel B DMA	Edge
28	`SPORT2_A_DMA`	SPORT2 Channel A DMA	Edge
29	`SPORT2_B_DMA`	SPORT2 Channel B DMA	Edge

Table 8-3: ADSP-SC58x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
30	`SPORT3_A_DMA`	SPORT3 Channel A DMA	Edge
31	`SPORT3_B_DMA`	SPORT3 Channel B DMA	Edge
32	`SPORT4_A_DMA`	SPORT4 Channel A DMA	Edge
33	`SPORT4_B_DMA`	SPORT4 Channel B DMA	Edge
34	`SPORT5_A_DMA`	SPORT5 Channel A DMA	Edge
35	`SPORT5_B_DMA`	SPORT5 Channel B DMA	Edge
36	`SPORT6_A_DMA`	SPORT6 Channel A DMA	Edge
37	`SPORT6_B_DMA`	SPORT6 Channel B DMA	Edge
38	`SPORT7_A_DMA`	SPORT7 Channel A DMA	Edge
39	`SPORT7_B_DMA`	SPORT7 Channel B DMA	Edge
40	`SPI0_TXDMA`	SPI0 TX DMA Channel	Edge
41	`SPI0_RXDMA`	SPI0 RX DMA Channel	Edge
42	`SPI1_TXDMA`	SPI1 TX DMA Channel	Edge
43	`SPI1_RXDMA`	SPI1 RX DMA Channel	Edge
44	`SPI2_TXDMA`	SPI2 TX DMA Channel	Edge
45	`SPI2_RXDMA`	SPI2 RX DMA Channel	Edge
46	`HAE0_RXDMA_CH0`	HAE0 RX DMA Channel 0	Level
47	`HAE0_RXDMA_CH1`	HAE0 RX DMA Channel 1	Level
48	`HAE0_TXDMA`	HAE0 TX DMA Channel 0	Level
49	`SINC0_P0_OVLD`	SINC0 Pair 0 Overload Indicator	Edge
50	`SINC0_P1_OVLD`	SINC0 Pair 1 Overload Indicator	Edge
51	`SINC0_P2_OVLD`	SINC0 Pair 2 Overload Indicator	Edge
52	`SINC0_P3_OVLD`	SINC0 Pair 3 Overload Indicator	Edge
53	`SINC0_DATA0`	SINC0 Data Move 0	Edge
54	`SINC0_DATA1`	SINC0 Data Move 1	Edge
55	`EMAC0_STAT`	EMAC0 Status	None
56	`EMAC1_STAT`	EMAC1 Status	None
57	`FFTA0_TXDMA`	FFTA0 Transmit DMA	
58	`FFTA0_RXDMA`	FFTA0 Receive DMA	
59	`FFTA0_TRIGOUT`	FFTA0 Trigger Out	
60	`FIR0_DMA`	FIR0 DMA	Edge

Table 8-3: ADSP-SC58x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
61	`IIR0_DMA`	IIR0 DMA	Edge
62	`EPPI0_CH0_DMA`	EPPI0 Channel 0 DMA	Edge
63	`EPPI0_CH1_DMA`	EPPI0 Channel 1 DMA	Edge
64	`LP0_DMA`	LP0 DMA Channel	
65	`LP1_DMA`	LP1 DMA Channel	
66	`UART0_TXDMA`	UART0 Transmit DMA	Edge
67	`UART0_RXDMA`	UART0 Receive DMA	Edge
68	`UART1_TXDMA`	UART1 Transmit DMA	Edge
69	`UART1_RXDMA`	UART1 Receive DMA	Edge
70	`UART2_TXDMA`	UART2 Transmit DMA	Edge
71	`UART2_RXDMA`	UART2 Receive DMA	Edge
72	`USB0_DATA`	USB0 DMA Status/Transfer Complete	Level
73	`USB1_DATA`	USB1 DMA Status/Transfer Complete	Level
74	`MDMA0_SRC`	Standard BW MDMA Channel 0 Source (CRC IN)	
75	`MDMA0_DST`	Standard BW MDMA Channel 0 Destination (CRC OUT)	
76	`MDMA1_SRC`	Standard BW MDMA Channel 1 Source (CRC IN)	
77	`MDMA1_DST`	Standard BW MDMA Channel 1 Destination (CRC OUT)	
78	`MDMA2_SRC`	Enh BW MDMA Channel 2 Source	
79	`MDMA2_DST`	Enh BW MDMA Channel 2 Destination	
80	`MDMA3_SRC`	Max BW MDMA Channel 3 Source	
81	`MDMA3_DST`	Max BW MDMA Channel 3 Destination	
82	`EMDMA0_DONE`	EMDMA0 DMA Done	Edge
83	`EMDMA1_DONE`	EMDMA1 DMA Done	Edge
84	`CTI3_MST0`	CTI3 SYSCTI (CTI3) System Halt Slave 0	Edge
85	`CTI3_MST1`	CTI3 SYSCTI (CTI3) System Halt Slave 1	Edge
86	`CTI3_MST2`	CTI3 SYSCTI (CTI3) System Halt Slave 2	Edge
87	`CTI3_MST3`	CTI3 SYSCTI (CTI3) System Halt Slave 3	Edge
88	`CTI3_MST4`	CTI3 SYSCTI (CTI3) System Halt Slave 4	Edge

Table 8-3: ADSP-SC58x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
89	`CTI3_MST5`	CTI3 SYSCTI (CTI3) System Halt Slave 5	Edge
90	`CTI3_MST6`	CTI3 SYSCTI (CTI3) System Halt Slave 6	Edge
91	`CTI3_MST7`	CTI3 SYSCTI (CTI3) System Halt Slave 7	Edge
92	`SEC0_FAULT`	SEC0 Fault	Edge
93	`SOFT0_MST`	Software-driven Trigger 0	
94	`SOFT1_MST`	Software-driven Trigger 1	
95	`SOFT2_MST`	Software-driven Trigger 2	
96	`SOFT3_MST`	Software-driven Trigger 3	
97	`SOFT4_MST`	Software-driven Trigger 4	
98	`SOFT5_MST`	Software-driven Trigger 5	
99	`SWU0_EVT`	SWU0 Event	None
100	`SWU2_EVT`	SWU2 Event	None
101	`SWU1_EVT`	SWU1 Event	None
102	`SWU4_EVT`	SWU4 Event	None
103	`SWU3_EVT`	SWU3 Event	None
104	`SWU6_EVT`	SWU6 Event	None
105	`SWU5_EVT`	SWU5 Event	None
106	`SWU7_EVT`	SWU7 Event	None
107	`SWU8_EVT`	SWU8 Event	None
108	`SWU9_EVT`	SWU9 Event	None
109	`SWU10_EVT`	SWU10 Event	None
110	`SWU11_EVT`	SWU11 Event	None
111	`SWU12_EVT`	SWU12 Event	None
112	`SWU13_EVT`	SWU13 Event	None
113	`SWU14_EVT`	SWU14 Event	None
114	`SWU15_EVT`	SWU15 Event	None
115	`SWU0_DBG`	SWU0 Debug	Edge
116	`SWU2_DBG`	SWU2 Debug	Edge
117	`SWU1_DBG`	SWU1 Debug	Edge
118	`SWU4_DBG`	SWU4 Debug	Edge
119	`SWU3_DBG`	SWU3 Debug	Edge

Table 8-3: ADSP-SC58x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
120	`SWU6_DBG`	SWU6 Debug	Edge
121	`SWU5_DBG`	SWU5 Debug	Edge
122	`SWU7_DBG`	SWU7 Debug	Edge
123	`SWU8_DBG`	SWU8 Debug	Edge
124	`SWU9_DBG`	SWU9 Debug	Edge
125	`SWU10_DBG`	SWU10 Debug	Edge
126	`SWU11_DBG`	SWU11 Debug	Edge
127	`SWU12_DBG`	SWU12 Debug	Edge
128	`SWU13_DBG`	SWU13 Debug	Edge
129	`SWU14_DBG`	SWU14 Debug	Edge
130	`SWU15_DBG`	SWU15 Debug	Edge
131	`TMU0_FAULT`	TMU0 Fault Event	
132	`ACM0_EVT_COMPLETE`	ACM0 Event Complete	
133	`HADC0_EOC`	HADC0 End of Conversion	Edge
134	`RTC0_EVT`	RTC0 Event	Level
135	`PCIE0_DMA`	PCIE0 DMA Done	
136	`MSI0_DONE`	MSI0 Transfer Done	Level
137	`C1_SID_ACK`	Core1 System Interface Disable Acknowledge	
138	`C2_SID_ACK`	Core2 System Interface Disable Acknowledge	

Table 8-4: ADSP-SC58x Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	`TIMER0_TMR0_SLV0`	TIMER0 Timer 0	Pulse
1	`TIMER0_TMR1_SLV0`	TIMER0 Timer 1	Pulse
2	`TIMER0_TMR2_SLV0`	TIMER0 Timer 2	Pulse
3	`TIMER0_TMR3_SLV0`	TIMER0 Timer 3	Pulse
4	`TIMER0_TMR4_SLV0`	TIMER0 Timer 4	Pulse
5	`TIMER0_TMR5_SLV0`	TIMER0 Timer 5	Pulse
6	`TIMER0_TMR6_SLV0`	TIMER0 Timer 6	Pulse
7	`TIMER0_TMR7_SLV0`	TIMER0 Timer 7	Pulse
8	`PWM0_TRIP_TRIG0`	PWM0 Trip Trigger Slave 0	Pulse

Table 8-4: ADSP-SC58x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
9	`PWM0_TRIP_TRIG1`	PWM0 Trip Trigger Slave 1	Pulse
10	`PWM0_TRIP_TRIG2`	PWM0 Trip Trigger Slave 2	Pulse
11	`PWM1_TRIP_TRIG0`	PWM1 Trip Trigger Slave 0	Pulse
12	`PWM1_TRIP_TRIG1`	PWM1 Trip Trigger Slave 1	Pulse
13	`PWM1_TRIP_TRIG2`	PWM1 Trip Trigger Slave 2	Pulse
14	`PWM2_TRIP_TRIG0`	PWM2 Trip Trigger Slave 0	Pulse
15	`PWM2_TRIP_TRIG1`	PWM2 Trip Trigger Slave 1	Pulse
16	`PWM2_TRIP_TRIG2`	PWM2 Trip Trigger Slave 2	Pulse
17	`SPORT0_A_DMA`	SPORT0 Channel A DMA	Pulse
18	`SPORT0_B_DMA`	SPORT0 Channel B DMA	Pulse
19	`SPORT1_A_DMA`	SPORT1 Channel A DMA	Pulse
20	`SPORT1_B_DMA`	SPORT1 Channel B DMA	Pulse
21	`SPORT2_A_DMA`	SPORT2 Channel A DMA	Pulse
22	`SPORT2_B_DMA`	SPORT2 Channel B DMA	Pulse
23	`SPORT3_A_DMA`	SPORT3 Channel A DMA	Pulse
24	`SPORT3_B_DMA`	SPORT3 Channel B DMA	Pulse
25	`SPORT4_A_DMA`	SPORT4 Channel A DMA	Pulse
26	`SPORT4_B_DMA`	SPORT4 Channel B DMA	Pulse
27	`SPORT5_A_DMA`	SPORT5 Channel A DMA	Pulse
28	`SPORT5_B_DMA`	SPORT5 Channel B DMA	Pulse
29	`SPORT6_A_DMA`	SPORT6 Channel A DMA	Pulse
30	`SPORT6_B_DMA`	SPORT6 Channel B DMA	Pulse
31	`SPORT7_A_DMA`	SPORT7 Channel A DMA	Pulse
32	`SPORT7_B_DMA`	SPORT7 Channel B DMA	Pulse
33	`SPI0_TXDMA`	SPI0 TX DMA Channel	Pulse
34	`SPI0_RXDMA`	SPI0 RX DMA Channel	Pulse
35	`SPI1_TXDMA`	SPI1 TX DMA Channel	Pulse
36	`SPI1_RXDMA`	SPI1 RX DMA Channel	Pulse
37	`SPI2_TXDMA`	SPI2 TX DMA Channel	Pulse
38	`SPI2_RXDMA`	SPI2 RX DMA Channel	Pulse
39	`HAE0_RXDMA_CH0`	HAE0 RX DMA Channel 0	Pulse

Table 8-4: ADSP-SC58x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
40	`HAE0_RXDMA_CH1`	HAE0 RX DMA Channel 1	Pulse
41	`HAE0_TXDMA`	HAE0 TX DMA Channel 0	Pulse
42	`SINC0_SYNC0`	SINC0 Synchronization Input 0	Pulse
43	`SINC0_SYNC1`	SINC0 Synchronization Input 1	Pulse
44	`FFTA0_TXDMA`	FFTA0 Transmit DMA	Pulse
45	`FFTA0_RXDMA`	FFTA0 Receive DMA	Pulse
46	`EPPI0_CH0_DMA`	EPPI0 Channel 0 DMA	Pulse
47	`EPPI0_CH1_DMA`	EPPI0 Channel 1 DMA	Pulse
48	`LP0_DMA`	LP0 DMA Channel	Pulse
49	`LP1_DMA`	LP1 DMA Channel	Pulse
50	`RCU0_SYSRST0`	RCU0 System Reset 0	Pulse
51	`RCU0_SYSRST1`	RCU0 System Reset 1	Pulse
52	`STM0_EVT0`	STM0 Event 0	Pulse
53	`STM0_EVT1`	STM0 Event 1	Pulse
54	`STM0_EVT2`	STM0 Event 2	Pulse
55	`STM0_EVT3`	STM0 Event 3	Pulse
56	`STM0_EVT4`	STM0 Event 4	Pulse
57	`STM0_EVT5`	STM0 Event 5	Pulse
58	`STM0_EVT6`	STM0 Event 6	Pulse
59	`STM0_EVT7`	STM0 Event 7	Pulse
60	`STM0_EVT8`	STM0 Event 8	Pulse
61	`STM0_EVT9`	STM0 Event 9	Pulse
62	`STM0_EVT10`	STM0 Event 10	Pulse
63	`STM0_EVT11`	STM0 Event 11	Pulse
64	`STM0_EVT12`	STM0 Event 12	Pulse
65	`STM0_EVT13`	STM0 Event 13	Pulse
66	`STM0_EVT14`	STM0 Event 14	Pulse
67	`STM0_EVT15`	STM0 Event 15	Pulse
68	`STM0_EVT16`	STM0 Event 16	Pulse
69	`STM0_EVT17`	STM0 Event 17	Pulse
70	`STM0_EVT18`	STM0 Event 18	Pulse

Table 8-4: ADSP-SC58x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
71	STM0_EVT19	STM0 Event 19	Pulse
72	STM0_EVT20	STM0 Event 20	Pulse
73	STM0_EVT21	STM0 Event 21	Pulse
74	STM0_EVT22	STM0 Event 22	Pulse
75	STM0_EVT23	STM0 Event 23	Pulse
76	STM0_EVT24	STM0 Event 24	Pulse
77	STM0_EVT25	STM0 Event 25	Pulse
78	STM0_EVT26	STM0 Event 26	Pulse
79	STM0_EVT27	STM0 Event 27	Pulse
80	STM0_EVT28	STM0 Event 28	Pulse
81	STM0_EVT29	STM0 Event 29	Pulse
82	STM0_EVT30	STM0 Event 30	Pulse
83	STM0_EVT31	STM0 Event 31	Pulse
84	TRU0_SLV0	TRU0 Interrupt Request 0, core ID = 0 only	Pulse
85	TRU0_SLV1	TRU0 Interrupt Request 1, core ID = 0 only	Pulse
86	TRU0_SLV2	TRU0 Interrupt Request 2, core ID = 0 only	Pulse
87	TRU0_SLV3	TRU0 Interrupt Request 3, core ID = 0 only	Pulse
88	TRU0_SLV4	TRU0 Interrupt Request 4, core ID = 1 only	Pulse
89	TRU0_SLV5	TRU0 Interrupt Request 5, core ID = 1 only	Pulse
90	TRU0_SLV6	TRU0 Interrupt Request 6, core ID = 1 only	Pulse
91	TRU0_SLV7	TRU0 Interrupt Request 7, core ID = 1 only	Pulse
92	TRU0_SLV8	TRU0 Interrupt Request 8, core ID = 2 only	Pulse
93	TRU0_SLV9	TRU0 Interrupt Request 9, core ID = 2 only	Pulse
94	TRU0_SLV10	TRU0 Interrupt Request 10, Core ID = 2 only	Pulse
95	TRU0_SLV11	TRU0 Interrupt Request 11, core ID = 2 only	Pulse
96	UART0_TXDMA	UART0 Transmit DMA	Pulse
97	UART0_RXDMA	UART0 Receive DMA	Pulse
98	UART1_TXDMA	UART1 Transmit DMA	Pulse
99	UART1_RXDMA	UART1 Receive DMA	Pulse
100	UART2_TXDMA	UART2 Transmit DMA	Pulse
101	UART2_RXDMA	UART2 Receive DMA	Pulse

Table 8-4: ADSP-SC58x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
102	`MDMA0_SRC`	Standard BW MDMA Channel 0 Source (CRC IN)	Pulse
103	`MDMA0_DST`	Standard BW MDMA Channel 0 Destination (CRC OUT)	Pulse
104	`MDMA1_SRC`	Standard BW MDMA Channel 1 Source (CRC IN)	Pulse
105	`MDMA1_DST`	Standard BW MDMA Channel 1 Destination (CRC OUT)	Pulse
106	`MDMA2_SRC`	Enh BW MDMA Source Channel	Pulse
107	`MDMA2_DST`	Enh BW MDMA Destination Channel	Pulse
108	`MDMA3_SRC`	Max BW MDMA Source Channel	Pulse
109	`MDMA3_DST`	Max BW MDMA Destination Channel	Pulse
110	`CTI3_SLV0`	CTI3 SYSCTI System Halt Master 0	Pulse
111	`CTI3_SLV1`	CTI3 SYSCTI System Halt Master 1	Pulse
112	`CTI3_SLV2`	CTI3 SYSCTI System Halt Master 2	Pulse
113	`CTI3_SLV3`	CTI3 SYSCTI System Halt Master 3	Pulse
114	`CTI3_SLV4`	CTI3 SYSCTI System Halt Master 4	Pulse
115	`CTI3_SLV5`	CTI3 SYSCTI System Halt Master 5	Pulse
116	`CTI3_SLV6`	CTI3 SYSCTI System Halt Master 6	Pulse
117	`CTI3_SLV7`	CTI3 SYSCTI System Halt Master 7	Pulse
118	`C0_EVT`	C0 Event input for wake-up from WFE state	Pulse
119	`SWU0_EN`	SWU0 Enable	Pulse
120	`SWU2_EN`	SWU2 Enable	Pulse
121	`SWU1_EN`	SWU1 Enable	Pulse
122	`SWU4_EN`	SWU4 Enable	Pulse
123	`SWU3_EN`	SWU3 Enable	Pulse
124	`SWU6_EN`	SWU6 Enable	Pulse
125	`SWU5_EN`	SWU5 Enable	Pulse
126	`SWU7_EN`	SWU7 Enable	Pulse
127	`SWU8_EN`	SWU8 Enable	Pulse
128	`SWU9_EN`	SWU9 Enable	Pulse
129	`SWU10_EN`	SWU10 Enable	Pulse

Table 8-4: ADSP-SC58x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
130	`SWU11_EN`	SWU11 Enable	Pulse
131	`SWU12_EN`	SWU12 Enable	Pulse
132	`SWU13_EN`	SWU13 Enable	Pulse
133	`SWU14_EN`	SWU14 Enable	Pulse
134	`SWU15_EN`	SWU15 Enable	Pulse
135	`ACM0_TRIG2`	ACM0 Trigger Input 2	Pulse
136	`ACM0_TRIG3`	ACM0 Trigger Input 3	Pulse

TRU Definitions

The following definitions are helpful when using the TRU module.

Trigger Master

A trigger master is any system module that provides trigger event indication to the TRU. Trigger master modules define trigger events and conditions for assertion.

Trigger Master ID

Trigger masters are assigned a unique numeric ID according to their physical connection to the TRU. Trigger master ID 0 is reserved and defined as null.

Trigger Slave

A trigger slave is any system module that receives a trigger event indication from the TRU. Trigger slave modules define a trigger event response.

TRU Block Diagram

Trigger masters and the Master Trigger Register (MTR) generate trigger assertions. Each trigger slave has a dedicated Slave Select Register (SSR) that specifies the unique trigger master from which it receives the trigger indication.

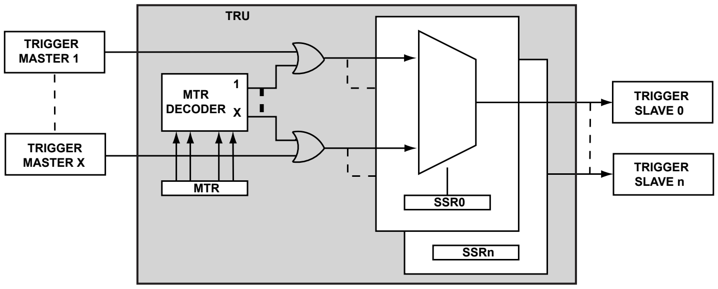


Figure 8-1: TRU Block Diagram

TRU Architectural Concepts

The TRU supports a simple trigger-in/trigger-out model for modules that comply with the triggering functional model. The TRU is the controller of the trigger system. Trigger outputs from trigger masters are mapped to trigger inputs of trigger slaves through a set of programmable registers (`TRU_SSR[n]`).

System modules are trigger master only, trigger slave only, or trigger master and trigger slave.

All of the trigger input and output signals are connected to a trigger routing unit (TRU) which manages the connections of triggers between modules.

In multi-processor systems, multiple TRU units are provided. These TRUs are networked together. Generic Trigger Ports (GTPs) are provided to forward trigger events from one TRU unit to another, forming a pathway from trigger masters to trigger slaves wherever they might lie in the system.

TRU Programming Model

Implementing sequence control using the TRU requires, at a minimum, proper configuration of a trigger slave, a trigger master, and the TRU module itself. The only requirement for the configuration procedure is that the trigger master is configured and enabled as the last step.

Complete the following other steps:

- Configure the trigger slave for response to triggers.
- Configure the TRU to map the trigger master to the trigger slave through the `TRU_SSR[n]` registers.
- Configure the trigger master to generate trigger assertions.
- Alternatively, use software triggering for trigger assertion. Writing the trigger master ID to the MTR register generates software triggers.

Programming Concepts

The following concepts aid in programming the TRU.

- *Trigger Sequence Configuration.* A simple sequence consists of one trigger master and one trigger slave. More complex trigger sequences consist of several trigger slaves functioning as trigger slave and trigger master. Additionally, trigger sequences can loopback to the original master forming a perpetual sequence.
- *Software Triggering.* Writing a trigger master ID to the MTR generates a trigger within the TRU from the trigger master ID specified.
- *Synchronization.* The TRU can be used to coarsely synchronize events by mapping multiple trigger slaves to the same trigger master or by generating multiple trigger master assertions simultaneously through the MTR.
- *Configuration Protection.* The `TRU_SSR[n].LOCK` bit and the `TRU_GCTL.LOCK` bit enable register level write-protection when the global lock is asserted in the SPU.

Programming Examples

The following examples shows the steps to create a single trigger and a Timer period expiry event automatically toggling a GPIO.

Configuring a Simple Trigger Sequence

The following example shows the steps to create a simple trigger.

1. Write to the `TRU_GCTL` register to enable the TRU.
2. Write to the `TRU_SSR[n]` register of a specific trigger slave to assign it to a specific trigger master.
3. Enable the trigger slave to wait for and accept a trigger.
4. Enable the trigger master to generate a trigger.

Toggle a GPIO on Timer Expiry Event

This example shows a case where a Timer period expiry event automatically toggles a GPIO. This is achieved by programming the Slave trigger register for GPIO with the Timer as Master.

1. Enable a specific pin in the PORT F to toggle up on trigger ( )
2. Enable timer to generate trigger up (`TIMER_TRG_MSK` register)
3. Program the Slave trigger 23 (`TRU_SSR[n]` register) (this toggles the PORTs) with the Timer as the Master.
4. Enable the TRU (`TRU_GCTL.EN` bit).

```
*pREG_PORTF_TRIG_TGL = 0x00000010;
*pREG_TIMER1_TRG_MSK &= ~BITM_TIMER_TRG_MSK_TMR03;
*pREG_TRU1_SSR23 = TRGM_TIMER1_TMR3_MST;
*pREG_TRU1_GCTL = 0x1;
```

TRU Event Control

The TRU is a major part of event control solutions. It is the center of the trigger functional model and can extend to support the interrupt and fault management models as well.

TRU Status and Error Signals

The TRU does not have dedicated status and error output signals other than the MMR interface. Slave errors are reported to the master over the standard peripheral bus protocol.

ADSP-SC58x TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 8-5: ADSP-SC58x TRU Register List

Name	Description
`TRU_ERRADDR`	Error Address Register
`TRU_GCTL`	Global Control Register
`TRU_MTR`	Master Trigger Register
`TRU_SSR[n]`	Slave Select Register
`TRU_STAT`	Status Information Register

Error Address Register

The TRU error address register (TRU_ERRADDR) holds the address from the memory-mapped register access generating an access error of TRU registers.

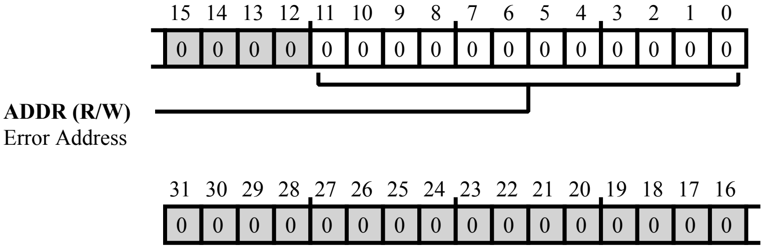

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR (R/W)
Error Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 8-2: TRU_ERRADDR Register Diagram

Table 8-6: TRU_ERRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ADDR	Error Address. The TRU_ERRADDR.ADDR holds the address from the memory-mapped register access generating an access error of TRU registers. These errors occur on access to the TRU_SSR[n] or TRU_MTR registers when these registers are locked or on access to an invalid address. See the TRU_SSR[n] and TRU_MTR register descriptions for more information about locking. The TRU_ERRADDR register holds the address of the first error to occur. In the event of multiple errors occurring, the TRU_ERRADDR register contains the address of the first error. To re-enable the TRU_ERRADDR register for update, both status bits (TRU_STAT.LWERR and TRU_STAT.ADDRERR) in the TRU_STAT register must be cleared.

Global Control Register

The TRU global control register (`TRU_GCTL`) provides register locking, TRU reset, and TRU enable.

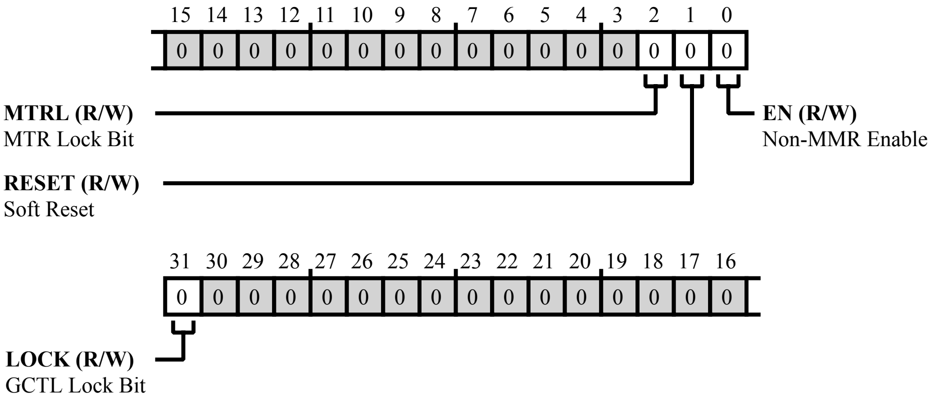

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MTRL (R/W)
MTR Lock Bit
EN (R/W)
Non-MMR Enable
RESET (R/W)
Soft Reset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
GCTL Lock Bit

Figure 8-3: TRU_GCTL Register Diagram

Table 8-7: TRU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	GCTL Lock Bit. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `TRU_GCTL.LOCK` bit is enabled, the `TRU_GCTL` register is read only.	
		0	Read write
		1	Read only
2 (R/W)	MTRL	MTR Lock Bit. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `TRU_GCTL.MTRL` bit is enabled, the `TRU_MTR` register is read only.	
		0	Read write
		1	Read only
1 (R/W)	RESET	Soft Reset. The `TRU_GCTL.RESET` bit is write-1-action and triggers a soft reset to all TRU registers.	
		0	No action
		1	Soft reset

Table 8-7: TRU_GCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Non-MMR Enable. The `TRU_GCTL.EN` bit is read/write and must be set for the TRU to propagate trigger events. All TRU register read/write operations continue to operate independent of the `TRU_GCTL.EN` bit.	
		0	No trigger events
		1	Propagate trigger events

Master Trigger Register

The TRU master trigger register (`TRU_MTR`) permits trigger generation through software by writing a trigger master ID value to one of the four fields in the `TRU_MTR` register. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `TRU_GCTL.LOCK` bit is set, the `TRU_MTR` register is read only. Note this register is primarily used for debug to trigger a TRU output

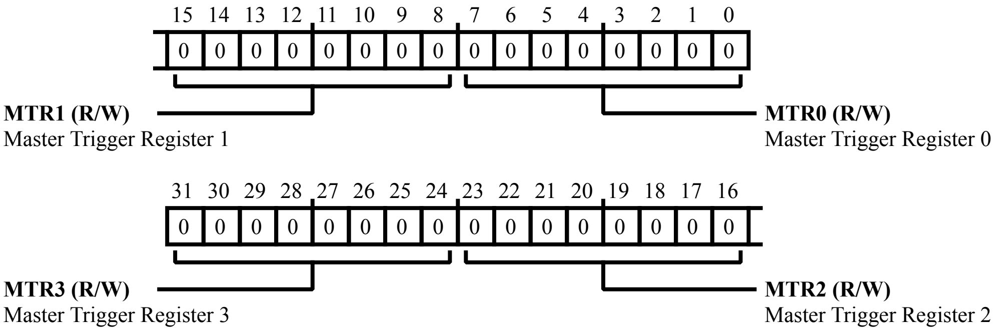

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MTR1 (R/W)
Master Trigger Register 1
MTR0 (R/W)
Master Trigger Register 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MTR3 (R/W)
Master Trigger Register 3
MTR2 (R/W)
Master Trigger Register 2

Figure 8-4: TRU_MTR Register Diagram

Table 8-8: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	MTR3	Master Trigger Register 3. The `TRU_MTR.MTR3` bit field is the trigger master ID value for master 3.	
		0	No master specified
		1-139	Range of valid masters
23:16 (R/W)	MTR2	Master Trigger Register 2. The `TRU_MTR.MTR2` bit field is the trigger master ID value for master 2.	
		0	No master specified
		1-139	Range of valid masters
15:8 (R/W)	MTR1	Master Trigger Register 1. The `TRU_MTR.MTR1` bit field is the trigger master ID value for master 1.	
		0	No master specified
		1-139	Range of valid masters
7:0 (R/W)	MTR0	Master Trigger Register 0. The `TRU_MTR.MTR0` bit field is the trigger master ID value for master 0.	
		0	No master specified
		1-139	Range of valid masters

Slave Select Register

The TRU slave select registers (`TRU_SSR[n]`) each provide slave selection and register locking.

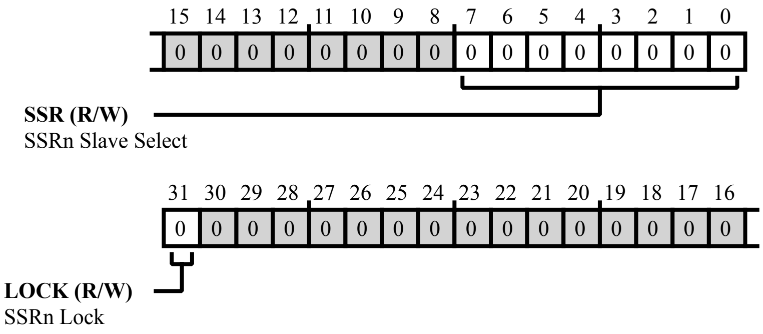

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SSR (R/W)
SSRn Slave Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
SSRn Lock

Figure 8-5: TRU_SSR[n] Register Diagram

Table 8-9: TRU_SSR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	SSRn Lock. If the global lock is enabled (`SPU_CTL.GLCK` bit =1) and the `TRU_SSR[n].LOCK` bit is enabled, the `TRU_SSR[n]` register is read only.	
		0	Unlock register
		1	Lock register
7:0 (R/W)	SSR	SSRn Slave Select. The `TRU_SSR[n]` register selects the trigger master ID to which the trigger slave responds. For example, when a `TRU_SSR[n]` register is set to respond to trigger master ID n, a trigger that is generated by trigger master ID n results in a trigger out to the slave.	
		0	No master specified
		1-139	Range of valid masters

Status Information Register

The TRU status register (TRU_STAT) contains the status of TRU_MTR and TRU_SSR[n] register writes and status of bus read/write errors.

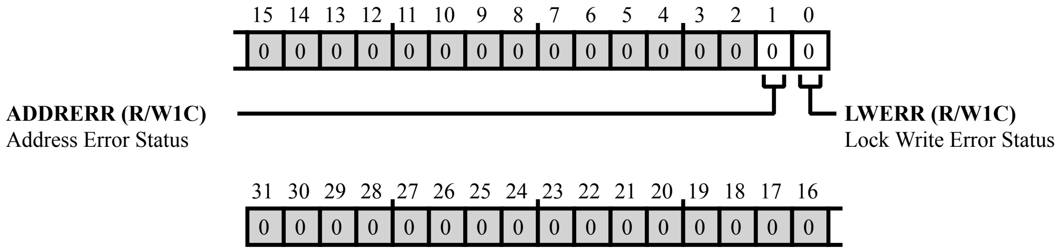

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRERR (R/W1C)
Address Error Status
LWERR (R/W1C)
Lock Write Error Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 8-6: TRU_STAT Register Diagram

Table 8-10: TRU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ADDRERR	Address Error Status. The `TRU_STAT.ADDRERR` bit is set when an invalid address is provided for an MMR access while the TRU is selected. Writing a one to this bit clears the error indication. The TRU_ERRADDR register also is updated when an address error occurs during an MMR access while the TRU is selected.	
		0	No error
		1	Error occurred
0 (R/W1C)	LWERR	Lock Write Error Status. If `TRU_STAT.LWERR` is set, a lock write error has occurred. Writing a one to this bit clears the error indication.	
		0	No error
		1	Error occurred

9 L2 System Memory

L2 system memories have significant bandwidth for core accesses, but it is important to note that L2 responds slower to core accesses than L1 memories. L2 SRAM is the ideal storage for multiple processor cores to share data and instruction resources, such as semaphores, shared buffers, and code libraries. Due to sophisticated data integrity protection and write protection, L2 SRAM is also ideal for data and instructions critical for safe operation of the application.

L2 System Memory Features

The L2 system memory features include:

- Operation at SYSCLK_0 frequency
- ECC protection of SRAM area
- ECC memory refresh

Three instances of L2 system memory: SRAM and 2x ROM.

- L2CTL0 contains 256K bytes of RAM grouped into eight banks, 32K bytes each and 32K bytes of boot ROM (ARM core).
- L2CTL1 contains 256K bytes of application ROM grouped into eight banks, 32K bytes each and 32K bytes of boot ROM (SHARC core).
- L2CTL2 contains 256K bytes of application ROM grouped into eight banks, 32K bytes each and 32K bytes of boot ROM (any of the cores).
- All instances supports exclusive access through SMPU.
- SHARC core data ports can access L2 memory through direct path as well as through ARM L2 cache.

L2 System Memory Functional Description

The L2 system memory manages all of the L2 SRAM and ROM memory banks. The system memory interface arbitrates competing accesses, write protection, and ensures SRAM data integrity. The L2 system memory domain is a unified instruction and data memory. It can hold any mixture of code and data required by the system design.

The following sections provide a functional description of the L2 system memory.

ADSP-SC58x L2CTL Interrupt List

Table 9-1: ADSP-SC58x L2CTL Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
8	L2CTL0_ECC_ERR	L2CTL0 ECC Error	Level	

ADSP-SC58x L2CTL Register List

The L2 memory controller (L2CTL) includes the controls to manage each L2 memory bank independently. A set of registers governs L2CTL operations. For more information on L2CTL functionality, see the L2CTL register descriptions.

Table 9-2: ADSP-SC58x L2CTL Register List

Name	Description
`L2CTL_CTL`	Control Register
`L2CTL_EADDR0`	Error Type 0 Address Register
`L2CTL_EADDR1`	Error Type 1 Address Register
`L2CTL_ERRADDR0`	ECC Error Address 0 Register
`L2CTL_ERRADDR1`	ECC Error Address 1 Register
`L2CTL_ERRADDR2`	ECC Error Address 2 Register
`L2CTL_ERRADDR3`	ECC Error Address 3 Register
`L2CTL_ERRADDR4`	ECC Error Address 4 Register
`L2CTL_ERRADDR5`	ECC Error Address 5 Register
`L2CTL_ERRADDR6`	ECC Error Address 6 Register
`L2CTL_ERRADDR7`	ECC Error Address 7 Register
`L2CTL_ET0`	Error Type 0 Register
`L2CTL_ET1`	Error Type 1 Register
`L2CTL_RFA`	Refresh Address Register
`L2CTL_RPCR`	Read Priority Count Register
`L2CTL_STAT`	Status Register
`L2CTL_WPCR`	Write Priority Count Register

L2 System Memory Block Diagram

The ***ADSP-SC58x Complete L2 System Block Diagram*** figure shows the complete L2 system memory, including the three memory block instances: L2CTL0, L2CTL1, and L2CTL2. The L2CTL0 block contains boot ROM code for

ARM and eight banks of L2 RAM containing 32 Kbytes each. ARM's 0x00000000 location (reset ISR) is mapped to this block. The L2CTL1 block contains one bank of boot ROM for SHARC+ ID = 1 and eight banks of application ROM.

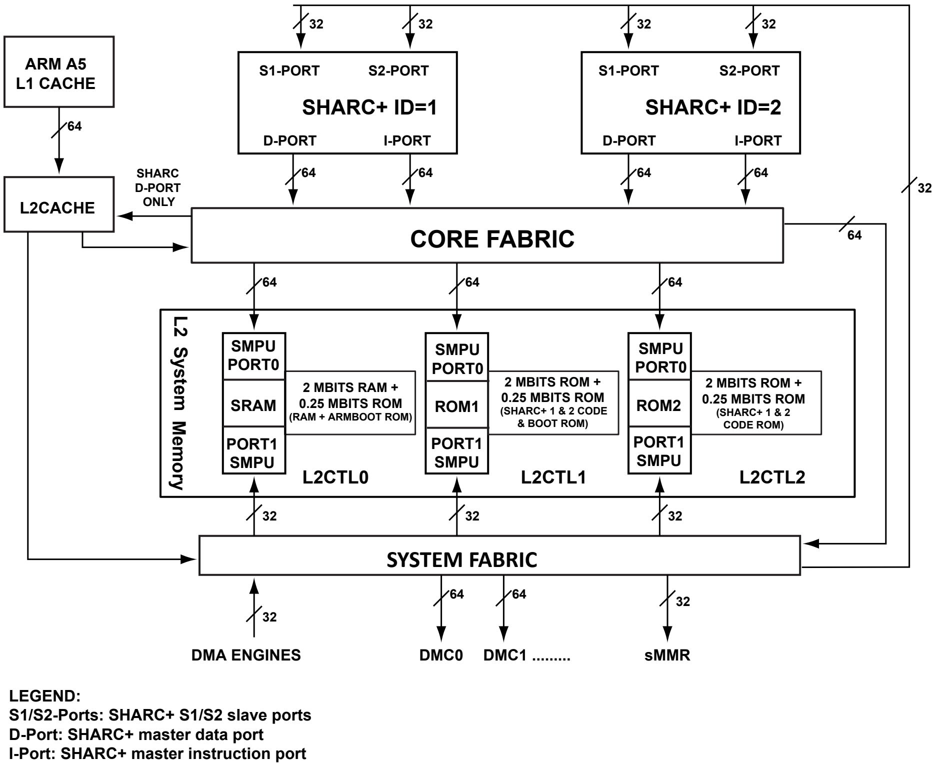

ARM A5
L1 CACHE
64
L2CACHE
SHARC
D-PORT
ONLY
32
32
S1-PORT
S2-PORT
SHARC+ ID=1
D-PORT
I-PORT
64
64
32
32
S1-PORT
S2-PORT
SHARC+ ID=2
D-PORT
I-PORT
64
64
32
CORE FABRIC
64
64
64
64
L2 System Memory
SMPU
PORT0
SRAM
PORT1
SMPU
2 MBITS RAM +
0.25 MBITS ROM
(RAM + ARMBOOT ROM)
L2CTL0
SMPU
PORT0
ROM1
PORT1
SMPU
2 MBITS ROM +
0.25 MBITS ROM
(SHARC+ 1 & 2 CODE
& BOOT ROM)
L2CTL1
SMPU
PORT0
ROM2
PORT1
SMPU
2 MBITS ROM +
0.25 MBITS ROM
(SHARC+ 1 & 2
CODE ROM)
L2CTL2
32
32
32
SYSTEM FABRIC
32
64
64
32
DMA ENGINES
DMC0
DMC1 ....
sMMR
LEGEND:
S1/S2-Ports: SHARC+ S1/S2 slave ports
D-Port: SHARC+ master data port
I-Port: SHARC+ master instruction port

Figure 9-1: ADSP-SC58x Complete L2 System Block Diagram

L2 System Memory Architectural Concepts

The following sections describe architecture features of the L2 system memory.

- Read/Write Latency and Throughput
- Arbitration and Priority

Access Characteristics

The L2 system memory interface converts all 8-bit, 16-bit, and 32-bit accesses to 64-bit accesses. Additionally, it converts 8-bit, 16-bit, and 32-bit bursts to an equivalent internal 64-bit access. For example, the L2 system memory interface converts a 64-bit address-aligned burst of 8-bit accesses of burst length 8 to a single 64-bit access.

Read/Write Latency and Throughput

The L2 memory design is optimized for burst accesses at the crossbar interface. The L2 system memory buffers and converts write data of 8/16/32-bit to an equivalent 64-bit access. This conversion creates modulo-32-bit writes if the starting addresses are 32-bit aligned. A single 8-bit or 16-bit access, or a non-32-bit address-aligned 8-bit or 16-bit burst access to an ECC-enabled bank creates an extra latency of two SYSCLK_0 cycles. No extra latency is seen if the ECC is disabled.

NOTE: Continuous 8/16-bit core access to an ECC-enabled L2 bank is not recommended from a throughput perspective.

L2 Memory Controller Block Diagram (Instance)

As shown in the following figure, the L2 controller has two ports that interface to system crossbars. Port 0 is a 64-bit interface that is dedicated to core traffic, and port 1 is a 32-bit interface that connects through DMA access. For L2 SRAM both ports (0/1) have a read channel and a write channel, for L2 ROM both ports (0/1) have read channels only. The SRAM/ROM are organized in multiple banks, and each bank has 32K Bytes of data. For ADSP-SC58x parts, the L2CTL0 has an additional bank for the boot ROM; for ADSP-2158x parts, the L2CTL1 has an additional bank for the boot ROM.

Within each bank, data is organized into 4096 words, with each word comprising 64 bits of data and 14 bits of ECC checksum. ROM memory is not protected by the ECC scheme. When the L2 controller accesses RAM and ROM cells, it always reads and writes whole 64-bit words. Despite this, the L2 controller supports 8-, 16-, and 32-bit reads and writes from cores and system by applying respective data masks.

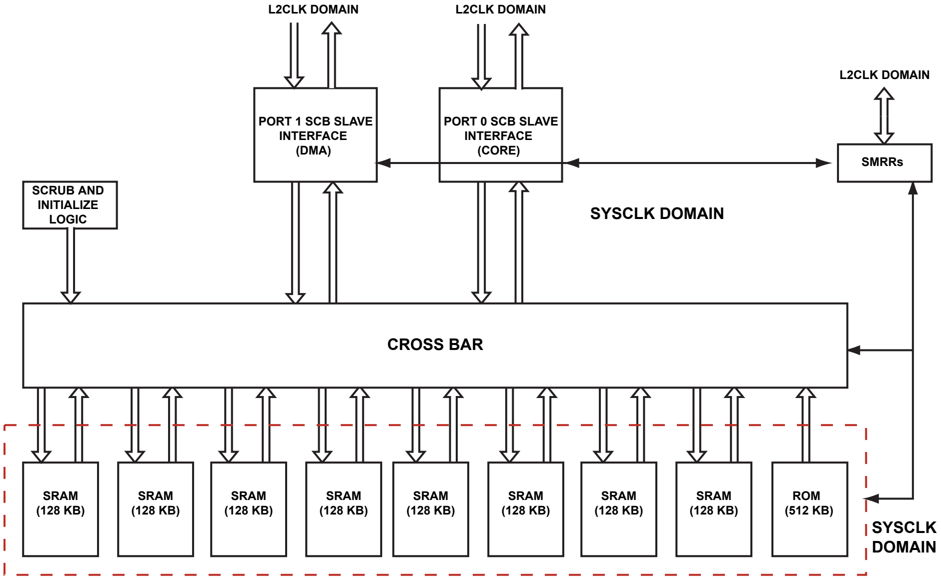

L2CLK DOMAIN
L2CLK DOMAIN
L2CLK DOMAIN
PORT 1 SCB SLAVE INTERFACE (DMA)
PORT 0 SCB SLAVE INTERFACE (CORE)
SMRRs
SCRUB AND INITIALIZE LOGIC
SYSCLK DOMAIN
CROSS BAR
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
SRAM (128 KB)
ROM (512 KB)
SYSCLK DOMAIN

Figure 9-2: L2 System Memory Block Diagram

Arbitration and Priority

Each bank of L2 RAM or ROM has an arbiter which receives requests from the two crossbar ports.

Each arbiter follows a fixed priority scheme for giving grants when more than one channel requests the same bank. The arbiter also supports priority elevation through urgent priority requests.

NOTE: Attempting a write access to both L2 ROM spaces returns an error.

The *Fixed Priority* table shows the priority for fixed priority mode (with urgent priority disabled) for each SCB channel.

If two cores (or the 64-bit Max BW DMA) simultaneously try to access L2 for the same instance (both read or both write), even to different banks, software allows only one master access at a time. One access port can support one read and write at the same time. However, if one core issues a write and the other issues a read, then access can proceed simultaneously. There is no extra latency inside L2, as long as the accesses are to different banks (assuming pending DMA traffic is also to a non-conflicting bank).

When a core and DMA both access the same bank via the same port (both read or both writes), the best access rate that DMA can achieve is one 64-bit access in every three SYSCLK_0 cycles during the conflict period. This access rate is achieved by programming the read priority count register (`L2CTL_RPCR.RPC0`) bit and the write priority count register (`L2CTL_WPCR.WPC0`) bit to 0, while programming the `L2CTL_RPCR.RPC1` and the `L2CTL_WPCR.WPC1` bits to 1.

Table 9-3: Fixed Priority

Channel	Priority Level
L2 Refresh Request	5 (highest)
Port 0 Read Channel	4
Port 0 Write Channel	3
Port 1 Read Channel	2
Port 1 Write Channel	1 (lowest)

The arbiters also support priority elevation for a particular channel that has been starved of grants for many SYSCLK_0 cycles. If a channel does not get a grant for *N* cycles after its request, then that channel can elevate the priority of its request by issuing an urgent priority request. This request causes that particular channel to become the highest priority master for the next grant cycle (pipelined arbitration for urgent priority). The number of cycles *N*, after which the priority is elevated, can be programmed for each channel separately using the `L2CTL_RPCR` and `L2CTL_WPCR` registers.

Programming the bits in the `L2CTL_RPCR` and `L2CTL_WPCR` registers appropriately achieves the best grant rate for DMA. This grant rate of one in three SYSCLK_0 cycles during the conflict period is achievable under the following conflict conditions:

- An access conflict between the core and DMA to the same memory bank in the fixed priority arbitration scheme with core activity always prioritized over DMA activity
- An access conflict within the pipelined implementation of urgent priority

To disable urgent priority requests, set the `L2CTL_CTL.DISURP` bit. This bit disables the urgent priority requests for all port channels. Each channel can also be prevented from raising the urgent priority request through the priority count register for the specific channel. However, there is no support for disabling urgent priority for a specific memory bank arbiter.

The *Fixed Priority With Priority Elevation* table provides the various priority levels for the L2 system memory.

Table 9-4: Fixed Priority With Priority Elevation

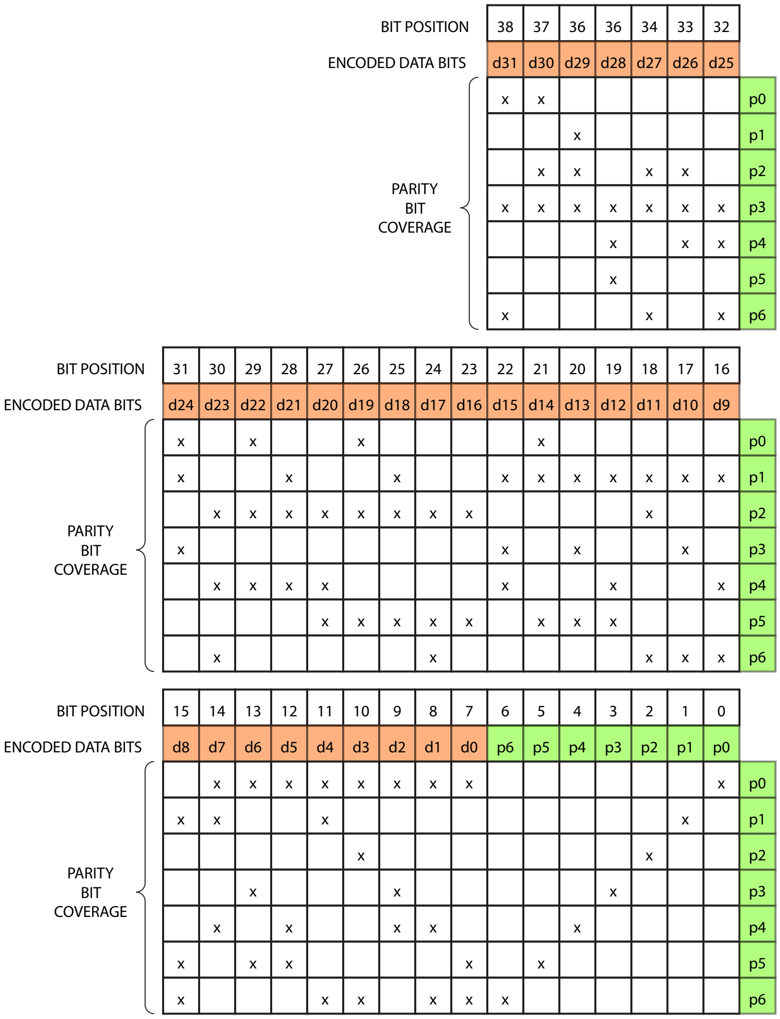
Channel	Priority Level
L2 Refresh Request	9 (highest)
Port 0 Read Channel Urgent Request	8
Port 0 Write Channel Urgent Request	7
Port 1 Read Channel Urgent Request	6
Port 1 Write Channel Urgent Request	5
Port 0 Read Channel Normal Request	4
Port 0 Write Channel Normal Request	3
Port 1 Read Channel Normal Request	2
Port 1 Write Channel Normal Request	1 (lowest)

Data Integrity

The following sections provide information on how the L2 system memory ensures data integrity.

ECC Algorithm

Hsaio encoding calculates the ECC syndrome. A 7-bit syndrome is generated during write operation and stored as a 7-bit parity field along with the 32 data bits. Each data bit contributes to three parity bits according. Each parity bit represents the XOR value of 13 or 14 data bits according to the following mapping:



BIT POSITION	38	37	36	36	34	33	32	
ENCODED DATA BITS	d31	d30	d29	d28	d27	d26	d25	
PARITY BIT COVERAGE	x	x						p0
			x					p1
		x	x		x	x		p2
	x	x	x	x	x	x	x	p3
				x		x	x	p4
				x				p5
	x				x		x	p6

BIT POSITION	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
ENCODED DATA BITS	d24	d23	d22	d21	d20	d19	d18	d17	d16	d15	d14	d13	d12	d11	d10	d9	
PARITY BIT COVERAGE	x		x			x					x						p0
	x			x			x			x	x	x	x	x	x	x	p1
		x	x	x	x	x	x	x	x					x			p2
	x									x		x			x		p3
		x	x	x	x					x			x			x	p4
					x	x	x	x	x		x	x	x				p5
		x						x						x	x	x	p6

BIT POSITION	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ENCODED DATA BITS	d8	d7	d6	d5	d4	d3	d2	d1	d0	p6	p5	p4	p3	p2	p1	p0	
PARITY BIT COVERAGE		x	x	x	x	x	x	x	x							x	p0
	x	x			x										x		p1
						x								x			p2
			x				x						x				p3
		x		x			x	x				x					p4
	x		x	x					x		x						p5
	x				x	x		x	x	x							p6

Figure 9-3: Hsaio Parity Bit Mapping

During read operation, the parity bits become part of the syndrome equation. The new syndrome bits are now the XOR values of the 13 or 14 data bits plus the respective stored parity bit. If any of the seven syndrome bits is set, an error situation is detected. An OR gate cross of the 7 bits reports the error, without specifying the type of the error.

If a single parity bit failed, the new 7-bit syndrome has 1 bit that is set. If a single data bit failed, the new syndrome has 3 bits that are set, because all three related parity bits fail. So, an XOR gate cross of all seven syndrome bits detects a single-bit error, indicating that an odd number of syndrome bits is set.

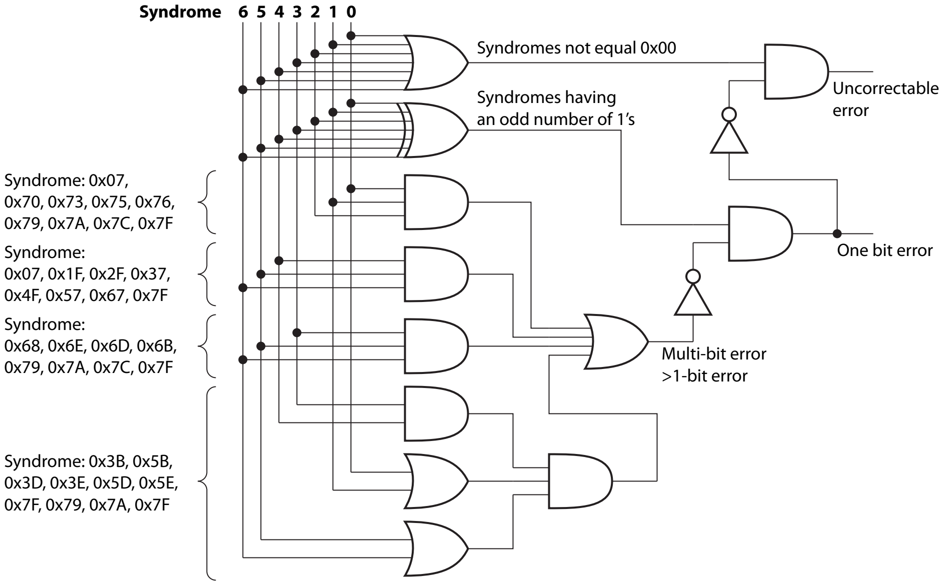

Syndrome 6 5 4 3 2 1 0
Syndromes not equal 0x00
Syndromes having an odd number of 1's
Uncorrectable error
One bit error
Multi-bit error >1-bit error
Syndrome: 0x07, 0x70, 0x73, 0x75, 0x76, 0x79, 0x7A, 0x7C, 0x7F
Syndrome: 0x07, 0x1F, 0x2F, 0x37, 0x4F, 0x57, 0x67, 0x7F
Syndrome: 0x68, 0x6E, 0x6D, 0x6B, 0x79, 0x7A, 0x7C, 0x7F
Syndrome: 0x3B, 0x5B, 0x3D, 0x3E, 0x5D, 0x5E, 0x7F, 0x79, 0x7A, 0x7F

Figure 9-4: Hsiao Error Reports

The XOR gate detects single-bit errors and does not flag any dual-bit error. But, the gate does flag 50% of the other multi-bit errors undesirably. Extra logic is implemented to increase the detection rate of multi-bit errors to 68.7% as shown in the figure.

If a single-bit error is detected, the failing bit can be localized and corrected. If all three syndrome bits corresponding to a specific data bit are 1, a data error is assumed. The respective data bit is toggled on its way to the system bus.

ECC Hardware Control

After reset, ECC protection is enabled. The boot code initializes all L2 SRAM data and checksum cells. ECC protection adds some cycle penalty when 8-bit and 16-bit values write L2 memory. Disable ECC protection for individual SRAM banks by setting the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits. Due to caching mechanisms of the processor cores and data bursting of the DMA channels, 8-bit and 16-bit write accesses are rather uncommon. Typically, only two-dimensional DMA operations or uncached 8-bit and 16-bit store instruction can trigger these writes.

For system integrity testing, the L2 system memory also provides a method for accessing the ECC checksum area directly. The `L2CTL_CTL.ECCMAP0` through `L2CTL_CTL.ECCMAP7` bits map the ECC checksum values into the address space of the data bits. This feature can be activated per SRAM bank. In this mode, only 32-bit accesses are allowed. 32-bit reads return the checksum value in the lower 7 bits while the upper bits read zero. Any 32-bit write overwrites the checksum. The upper bits are ignored.

Using this checksum mapping feature, safety critical applications can verify the ECC hardware during boot up sequence or even at run time. It is not required to set the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits explicitly. To test the ECC hardware, use the following steps:

1. Write data values to L2 SRAM destination (preferable an even number of 32-bit words).
2. If data cache enabled, make sure that it flushes out data.

3. Execute SYNC instruction.
4. Set `L2CTL_CTL.ECCMAP7-L2CTL_CTL.ECCMAP0` bits of interest.
5. Execute SYNC instruction.
6. Write checksum values using 32-bit store instructions.
7. If data cache enabled, make sure that it flushes out checksum values.
8. Execute SYNC instruction.
9. Clear `L2CTL_CTL.ECCMAP7-L2CTL_CTL.ECCMAP0` bits.
10. Execute SYNC instruction.
11. Read data values back.

ECC Error Management

The L2 system memory flags 2-bit and multi-bit errors to the system by:

- Raising the ECC_ERR interrupt
- Reporting a read error to the system bus
- Setting the sticky `L2CTL_STAT.ECCERR7-L2CTL_STAT.ECCERR0` status flag
- Latching the address of the failing operation into the respective `L2CTL_ERRADDR7-L2CTL_ERRADDR0` register.

There is one error status bit and one error address register per L2 SRAM bank.

Typically, ECC_ERR events are declared as system faults in the system event controller (SEC). Whether these faults are reported, the interrupt service routine can consult the `L2CTL_STAT` register and the `L2CTL_ERRADDR0` through `L2CTL_ERRADDR7` registers to determine whether:

- The data at the failing L2 address was critical enough to require an immediate reboot of the system
- The data at the failing L2 address was less critical or can be restored

The `L2CTL_STAT.ECCERR0` through `L2CTL_STAT.ECCERR7` flags are cleared with a W1C operation.

Memory Refresh

If data in L2 SRAM contains single-bit errors, the data is corrected on its way to the system buses. The corrected value is not written back to the SRAM location. To prevent any risk of accumulation of single-bit errors over time and to minimize likelihood of multi-bit errors, the L2 system memory provides a special memory refresh mechanism.

Software can initiate a memory refresh cycle of a 64-bit SRAM entity by writing the address of interest into the refresh address register, `L2CTL_RFA`. The write triggers an atomic operation. In this operation, the L2 system memory:

- Reads a 64-bit entity from the targeted memory

- Applies an ECC algorithm to the two 32-bit words
- Writes the corrected data back to memory

While the atomic refresh operation is ongoing, other accesses to the same SRAM bank are locked-out. The `L2CTL_STAT.RFRS` status bit signals an ongoing refresh operation. Hardware clears the bit after the operation has finished. The content of the `L2CTL_RFA` register must not change while the refresh operation is ongoing.

In safety-critical applications, software can refresh all L2 SRAM by periodically writing to the `L2CTL_RFA` register with values. It increments with a value of eight until all SRAM locations are refreshed.

Memory refresh operation is meaningless when the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits are set.

L2 System Memory Event Control

The following sections describe event control features of the L2 system memory, such as error response.

ECC Error Interrupt

A bus error is signaled under any of the following conditions.

- A write access to ROM address space
- A read/write access to reserved address space
- An ECC multi-bit error in an ECC-enabled bank. A non-modulo, 32-bit write to an ECC-enabled bank can also potentially create a bus error response due to an ECC multi-bit error. This response is because the L2 system memory implements a 32-bit ECC, and therefore a non-modulo, 32-bit write results in a read. This read can create multi-bit errors even if the memory was initialized.

Bus error notifications are stored in the `L2CTL_STAT` register, and the addresses that generated the error on a given port are stored in the `L2CTL_EADDR0`/`L2CTL_EADDR1` register of that port. The details of the error are stored in the `L2CTL_ET0`/`L2CTL_ET1` register of the port.

ADSP-SC58x L2CTL Register Descriptions

L2 Memory Controller (L2CTL) contains the following registers.

Table 9-5: ADSP-SC58x L2CTL Register List

Name	Description
`L2CTL_CTL`	Control Register
`L2CTL_EADDR0`	Error Type 0 Address Register
`L2CTL_EADDR1`	Error Type 1 Address Register
`L2CTL_ERRADDR0`	ECC Error Address 0 Register

Table 9-5: ADSP-SC58x L2CTL Register List (Continued)

Name	Description
`L2CTL_ERRADDR1`	ECC Error Address 1 Register
`L2CTL_ERRADDR2`	ECC Error Address 2 Register
`L2CTL_ERRADDR3`	ECC Error Address 3 Register
`L2CTL_ERRADDR4`	ECC Error Address 4 Register
`L2CTL_ERRADDR5`	ECC Error Address 5 Register
`L2CTL_ERRADDR6`	ECC Error Address 6 Register
`L2CTL_ERRADDR7`	ECC Error Address 7 Register
`L2CTL_ET0`	Error Type 0 Register
`L2CTL_ET1`	Error Type 1 Register
`L2CTL_RFA`	Refresh Address Register
`L2CTL_RPCR`	Read Priority Count Register
`L2CTL_STAT`	Status Register
`L2CTL_WPCR`	Write Priority Count Register

Control Register

The `L2CTL_CTL` register includes a write protection bit, enables L2 banks, and selects mapping of banks (as ECC RAM or data RAM).

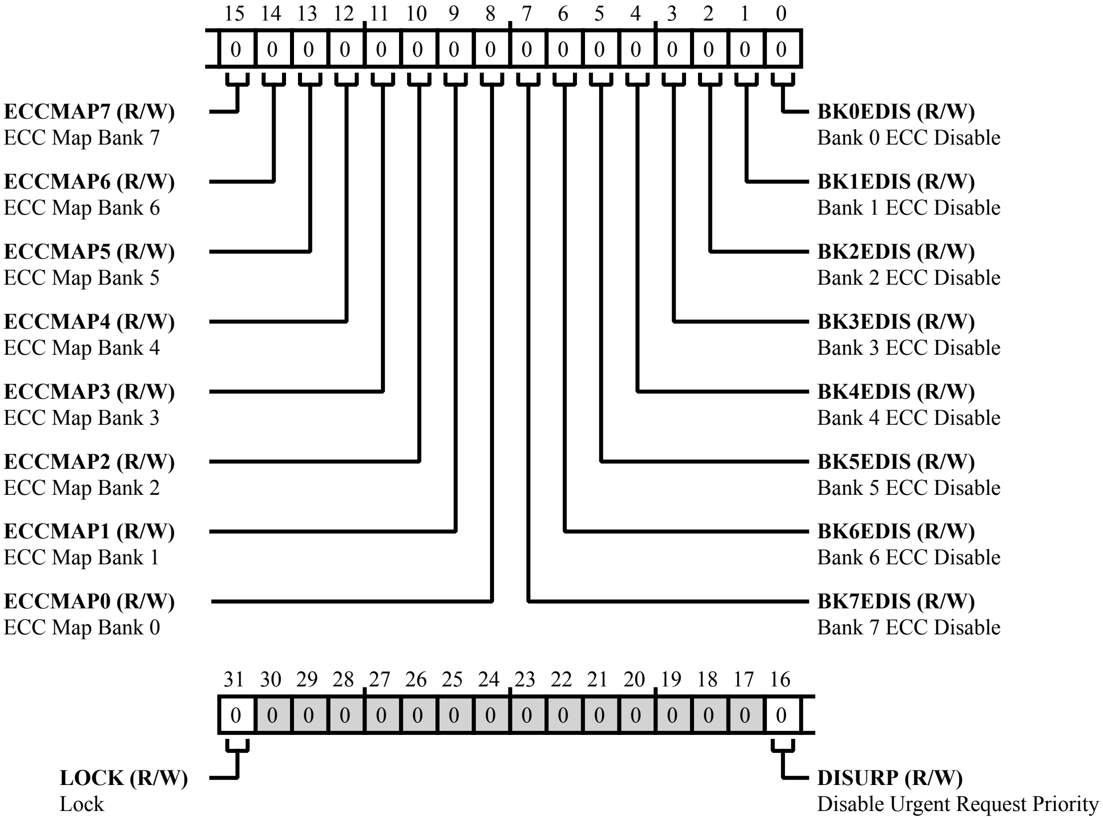

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ECCMAP7 (R/W)
ECC Map Bank 7
ECCMAP6 (R/W)
ECC Map Bank 6
ECCMAP5 (R/W)
ECC Map Bank 5
ECCMAP4 (R/W)
ECC Map Bank 4
ECCMAP3 (R/W)
ECC Map Bank 3
ECCMAP2 (R/W)
ECC Map Bank 2
ECCMAP1 (R/W)
ECC Map Bank 1
ECCMAP0 (R/W)
ECC Map Bank 0
BK0EDIS (R/W)
Bank 0 ECC Disable
BK1EDIS (R/W)
Bank 1 ECC Disable
BK2EDIS (R/W)
Bank 2 ECC Disable
BK3EDIS (R/W)
Bank 3 ECC Disable
BK4EDIS (R/W)
Bank 4 ECC Disable
BK5EDIS (R/W)
Bank 5 ECC Disable
BK6EDIS (R/W)
Bank 6 ECC Disable
BK7EDIS (R/W)
Bank 7 ECC Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock
DISURP (R/W)
Disable Urgent Request Priority

Figure 9-5: L2CTL_CTL Register Diagram

Table 9-6: L2CTL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (`SPU_CTL.GLCK` bit =1) and the `L2CTL_CTL.LOCK` bit is set, the `L2CTL_CTL` register is read only (locked).	
		0	Unlock
		1	Lock
16 (R/W)	DISURP	Disable Urgent Request Priority. The `L2CTL_CTL.DISURP` disables urgent request priority mode for all L2 banks.	
		0	Enable URP
		1	Disable URP

Table 9-6: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	ECCMAP7	ECC Map Bank 7. The `L2CTL_CTL.ECCMAP7` bit selects whether L2 bank 7 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
14 (R/W)	ECCMAP6	ECC Map Bank 6. The `L2CTL_CTL.ECCMAP6` bit selects whether L2 bank 6 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
13 (R/W)	ECCMAP5	ECC Map Bank 5. The `L2CTL_CTL.ECCMAP5` bit selects whether L2 bank 5 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
12 (R/W)	ECCMAP4	ECC Map Bank 4. The `L2CTL_CTL.ECCMAP4` bit selects whether L2 bank 4 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
11 (R/W)	ECCMAP3	ECC Map Bank 3. The `L2CTL_CTL.ECCMAP3` bit selects whether L2 bank 3 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
10 (R/W)	ECCMAP2	ECC Map Bank 2. The `L2CTL_CTL.ECCMAP2` bit selects whether L2 bank 2 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM

Table 9-6: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	ECCMAP1	ECC Map Bank 1. The `L2CTL_CTL.ECCMAP1` bit selects whether L2 bank 1 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
8 (R/W)	ECCMAP0	ECC Map Bank 0. The `L2CTL_CTL.ECCMAP0` bit selects whether L2 bank 0 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
7 (R/W)	BK7EDIS	Bank 7 ECC Disable. The `L2CTL_CTL.BK7EDIS` bit disables L2 bank 7 ECC operation.	
		0	Enable ECC
		1	Disable ECC
6 (R/W)	BK6EDIS	Bank 6 ECC Disable. The `L2CTL_CTL.BK6EDIS` bit disables L2 bank 6 ECC operation.	
		0	Enable ECC
		1	Disable ECC
5 (R/W)	BK5EDIS	Bank 5 ECC Disable. The `L2CTL_CTL.BK5EDIS` bit disables L2 bank 5 ECC operation.	
		0	Enable ECC
		1	Disable ECC
4 (R/W)	BK4EDIS	Bank 4 ECC Disable. The `L2CTL_CTL.BK4EDIS` bit disables L2 bank 4 ECC operation.	
		0	Enable ECC
		1	Disable ECC
3 (R/W)	BK3EDIS	Bank 3 ECC Disable. The `L2CTL_CTL.BK3EDIS` bit disables L2 bank 3 ECC operation.	
		0	Enable ECC
		1	Disable ECC

Table 9-6: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	BK2EDIS	Bank 2 ECC Disable. The `L2CTL_CTL.BK2EDIS` bit disables L2 bank 2 ECC operation.	
		0	Enable ECC
		1	Disable ECC
1 (R/W)	BK1EDIS	Bank 1 ECC Disable. The `L2CTL_CTL.BK1EDIS` bit disables L2 bank 1 ECC operation.	
		0	Enable ECC
		1	Disable ECC
0 (R/W)	BK0EDIS	Bank 0 ECC Disable. The `L2CTL_CTL.BK0EDIS` bit disables L2 bank 0 ECC operation.	
		0	Enable ECC
		1	Disable ECC

Error Type 0 Address Register

The `L2CTL_EADDR0` register holds the address that created an access error on the L2 port 0 bus interface (cores). This register is be updated only if the corresponding error status bit (`L2CTL_STAT.ERR0`) is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_EADDR0` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

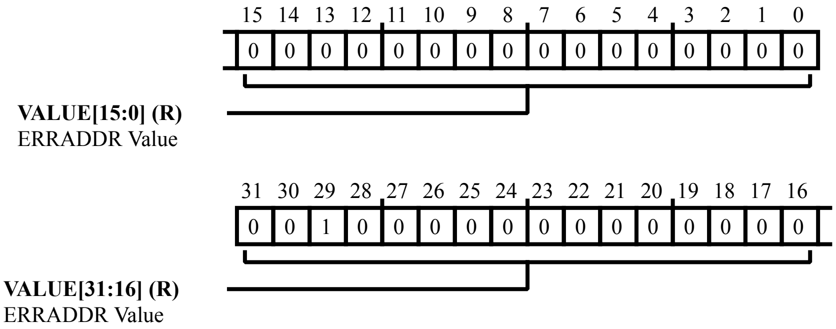

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-6: L2CTL_EADDR0 Register Diagram

Table 9-7: L2CTL_EADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_EADDR0.VALUE` bits hold the address causing the bus error.

Error Type 1 Address Register

The `L2CTL_EADDR1` register holds the address that created an access error on the L2 port 1 bus interface (DMA). This register is be updated only if the corresponding error status bit (`L2CTL_STAT.ERR1`) is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_EADDR1` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

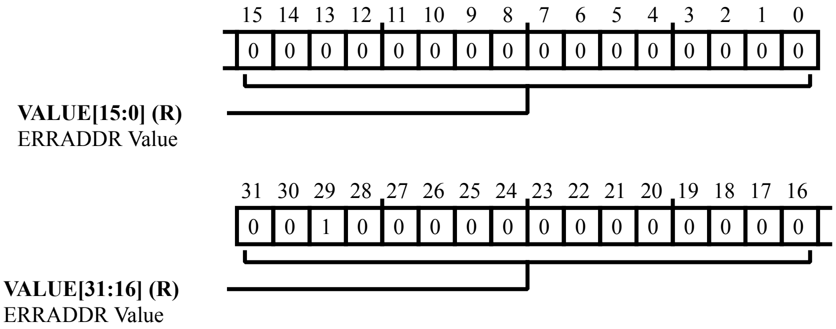

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-7: L2CTL_EADDR1 Register Diagram

Table 9-8: L2CTL_EADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_EADDR1.VALUE` bits hold the address causing the bus error.

ECC Error Address 0 Register

The `L2CTL_ERRADDR0` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR0`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

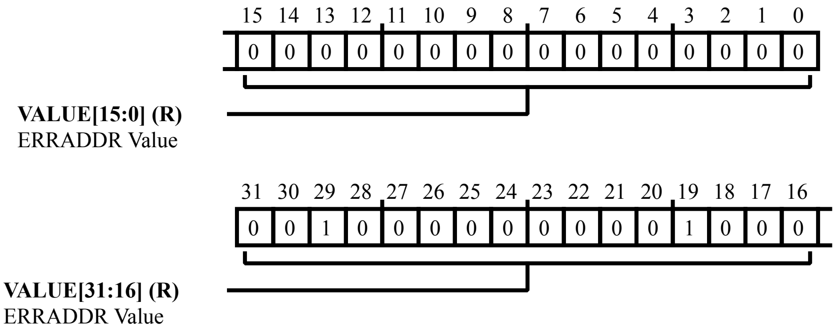

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-8: L2CTL_ERRADDR0 Register Diagram

Table 9-9: L2CTL_ERRADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR0.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 1 Register

The `L2CTL_ERRADDR1` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR1`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

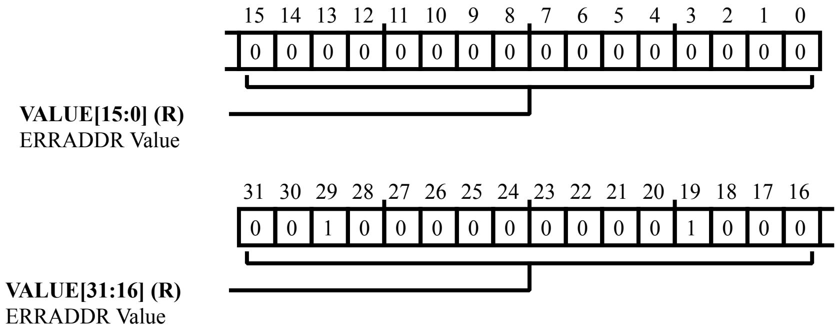

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-9: L2CTL_ERRADDR1 Register Diagram

Table 9-10: L2CTL_ERRADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR1.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 2 Register

The `L2CTL_ERRADDR2` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR2`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

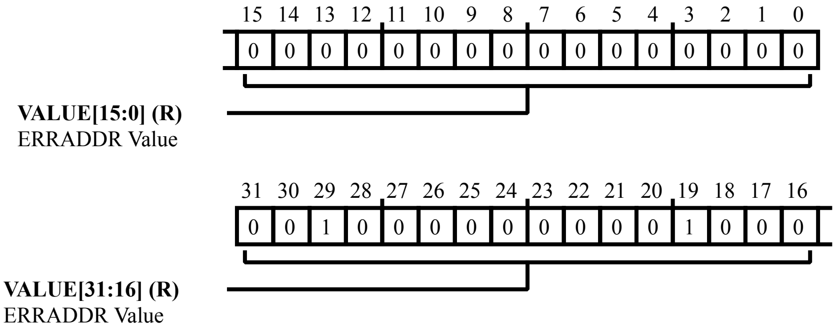

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-10: L2CTL_ERRADDR2 Register Diagram

Table 9-11: L2CTL_ERRADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR2.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 3 Register

The `L2CTL_ERRADDR3` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR3`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

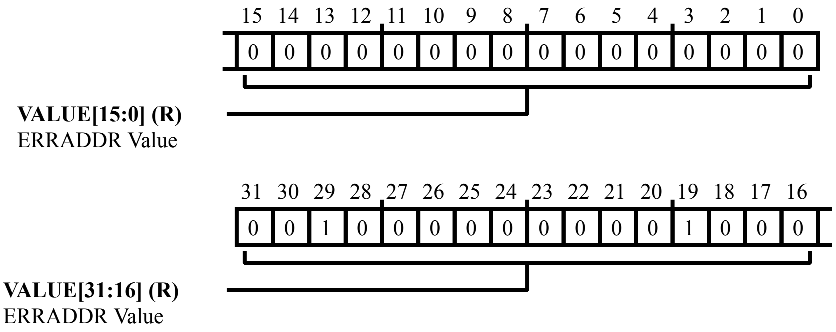

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-11: L2CTL_ERRADDR3 Register Diagram

Table 9-12: L2CTL_ERRADDR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR3.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 4 Register

The `L2CTL_ERRADDR4` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR4`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

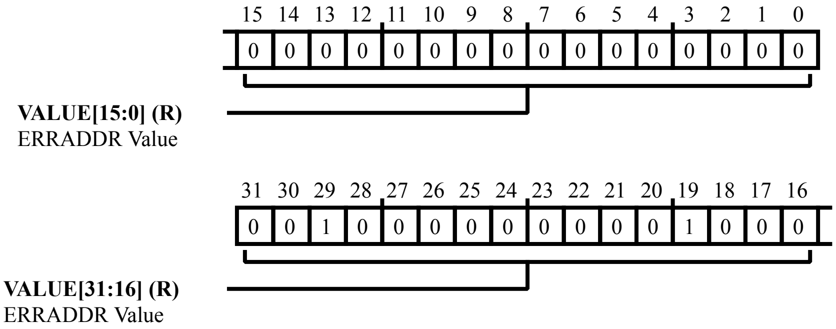

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-12: L2CTL_ERRADDR4 Register Diagram

Table 9-13: L2CTL_ERRADDR4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR4.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 5 Register

The `L2CTL_ERRADDR5` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR5`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

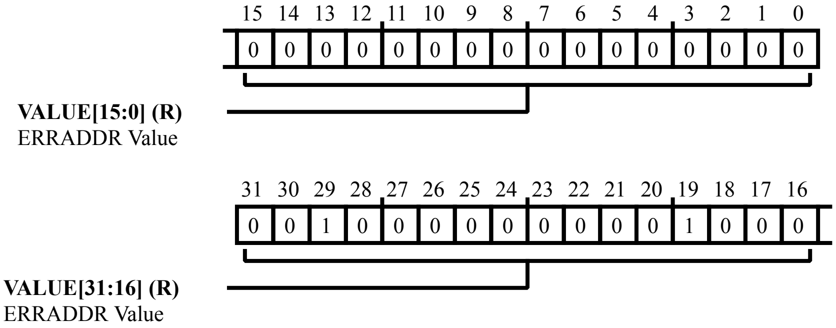

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-13: L2CTL_ERRADDR5 Register Diagram

Table 9-14: L2CTL_ERRADDR5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR5.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 6 Register

The `L2CTL_ERRADDR6` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR6`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

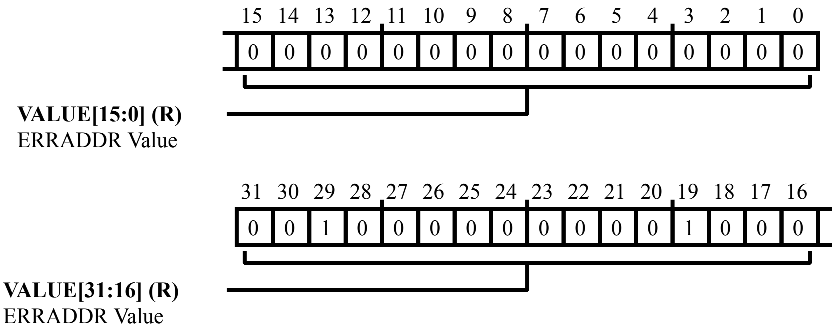

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-14: L2CTL_ERRADDR6 Register Diagram

Table 9-15: L2CTL_ERRADDR6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR6.VALUE` bits hold the address containing the ECC double-bit error.

ECC Error Address 7 Register

The `L2CTL_ERRADDR7` register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (`L2CTL_STAT.ECCERR7`) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

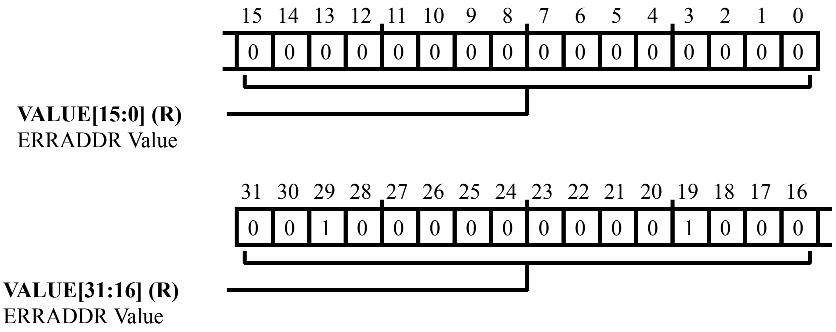

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
ERRADDR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
VALUE[31:16] (R)
ERRADDR Value

Figure 9-15: L2CTL_ERRADDR7 Register Diagram

Table 9-16: L2CTL_ERRADDR7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The `L2CTL_ERRADDR7.VALUE` bits hold the address containing the ECC double-bit error.

Error Type 0 Register

The `L2CTL_ET0` register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port 0 (cores). This register is updated only if the corresponding error status bit `L2CTL_STAT.ERR0` is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_ET0` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the `L2CTL_ET0` captures the write access error, keeping in sync with the error address register (`L2CTL_EADDR0`).

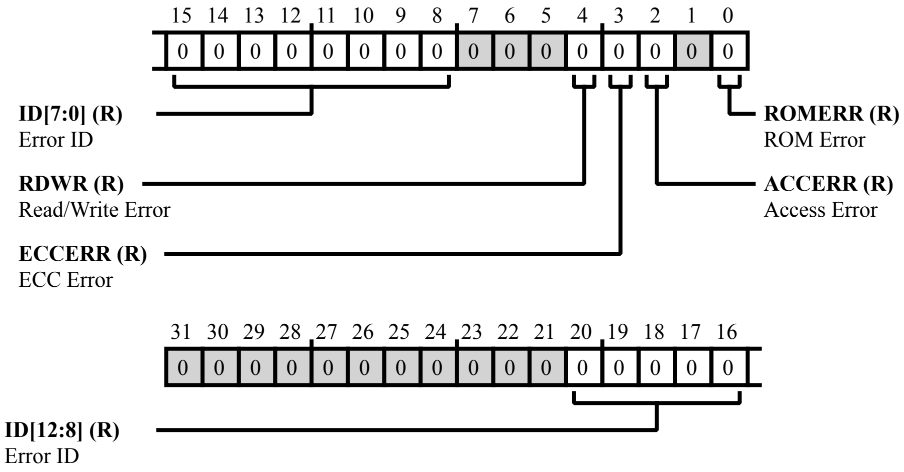

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[7:0] (R)
Error ID
RDWR (R)
Read/Write Error
ECCERR (R)
ECC Error
ROMERR (R)
ROM Error
ACCERR (R)
Access Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[12:8] (R)
Error ID

Figure 9-16: L2CTL_ET0 Register Diagram

Table 9-17: L2CTL_ET0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20:8 (R/NW)	ID	Error ID. The `L2CTL_ET0.ID` bits hold the bus master ID of the access that caused an error.
4 (R/NW)	RDWR	Read/Write Error. The `L2CTL_ET0.RDWR` bit indicates whether a read or write access caused an error. 0: Read Access created Error 1: Write Access created Error
3 (R/NW)	ECCERR	ECC Error. The `L2CTL_ET0.ECCERR` bit indicates whether the access had an ECC double-bit error.
2 (R/NW)	ACCERR	Access Error. The `L2CTL_ET0.ACCERR` bit indicates whether the access went to a restricted bank.

Table 9-17: L2CTL_ET0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	ROMERR	ROM Error. The `L2CTL_ET0.ROMERR` bit indicates whether a write access went to a ROM area.

Error Type 1 Register

The `L2CTL_ET1` register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port 1 (DMA). This register is updated only if the corresponding error status bit `L2CTL_STAT.ERR1` is cleared. After the status bit is set for an error, further errors do not update the `L2CTL_ET1` register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the `L2CTL_ET1` captures the write access error, keeping in sync with the error address register (`L2CTL_EADDR1`).

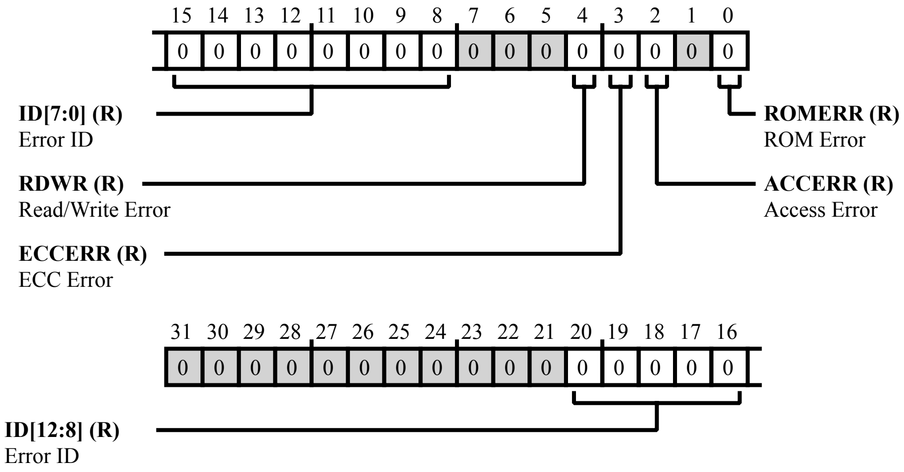

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[7:0] (R)
Error ID
RDWR (R)
Read/Write Error
ECCERR (R)
ECC Error
ROMERR (R)
ROM Error
ACCERR (R)
Access Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[12:8] (R)
Error ID

Figure 9-17: L2CTL_ET1 Register Diagram

Table 9-18: L2CTL_ET1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:8 (R/NW)	ID	Error ID. The `L2CTL_ET1.ID` bits hold the bus master ID of the access that caused an error.	
4 (R/NW)	RDWR	Read/Write Error. The `L2CTL_ET1.RDWR` bit indicates whether a read or write access caused an error.	
		0	Read access created error
		1	Write access created error
3 (R/NW)	ECCERR	ECC Error. If the `L2CTL_ET1.ECCERR` bit =1, the access had an ECC double-bit error.	
2 (R/NW)	ACCERR	Access Error. If the `L2CTL_ET1.ACCERR` bit =1, the access went to a restricted bank.	
0 (R/NW)	ROMERR	ROM Error. If the `L2CTL_ET1.ROMERR` bit =1, a write access went to a ROM area.	

Refresh Address Register

The `L2CTL_RFA` register stores the refresh address value. When this register is written, L2 initiates an atomic read-write operation to the address value written into the register. This is a read/write register, but a new value in the corresponding field has to be written only when there are no outstanding refresh request pending (`L2CTL_STAT.RFRS` =0). If a write occurs while a request is pending, the L2CTL generates a bus error, and the write does not take effect.

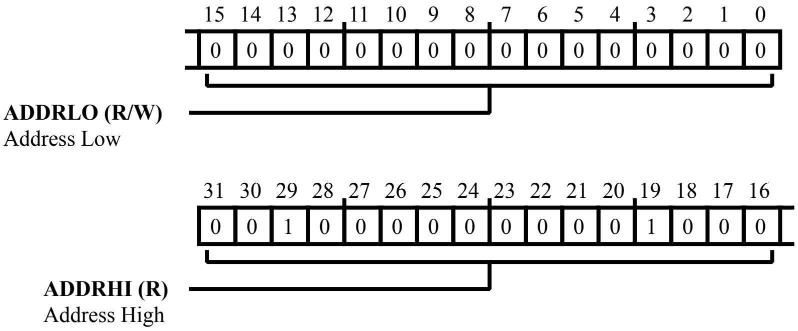

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRLO (R/W)
Address Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
ADDRHI (R)
Address High

Figure 9-18: L2CTL_RFA Register Diagram

Table 9-19: L2CTL_RFA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	ADDRHI	Address High. The `L2CTL_RFA.ADDRHI` bits hold the high 16-bits of the L2 refresh address. Note that the upper 14 bits are hard-coded to the upper bits of the L2 address map.
15:0 (R/W)	ADDRLO	Address Low. The `L2CTL_RFA.ADDRLO` bits hold the low 16-bits of the L2 refresh address. Note that the lowest three bits are do-not-care.

Read Priority Count Register

The `L2CTL_RPCR` register stores the count value to be used for priority elevation for bus read channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SYSCLK_0 cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus read channel. A best practice is to program this register before initiating an L2 access.

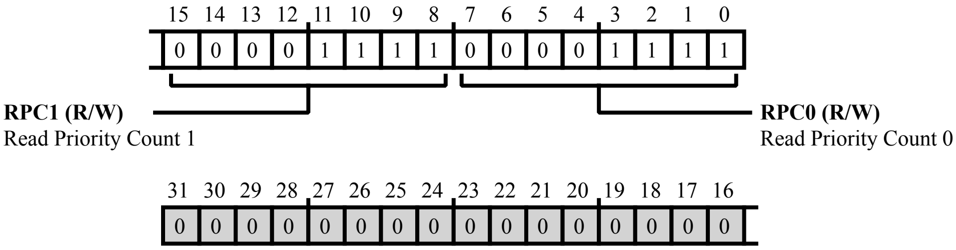

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
RPC1 (R/W)
Read Priority Count 1
RPC0 (R/W)
Read Priority Count 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 9-19: L2CTL_RPCR Register Diagram

Table 9-20: L2CTL_RPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	RPC1	Read Priority Count 1. The `L2CTL_RPCR.RPC1` bits hold the priority count for L2 bus read channel 1.
7:0 (R/W)	RPC0	Read Priority Count 0. The `L2CTL_RPCR.RPC0` bits hold the priority count for L2 bus read channel 0.

Status Register

The `L2CTL_STAT` register indicates ECC error status, refresh register status, and bus error status.

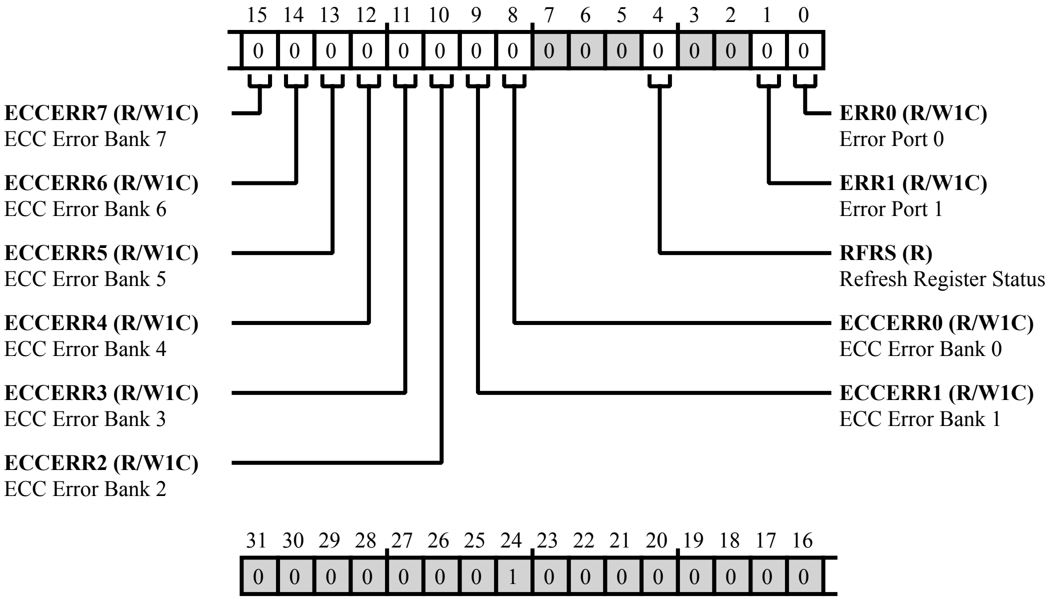

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ECCERR7 (R/W1C)
ECC Error Bank 7
ECCERR6 (R/W1C)
ECC Error Bank 6
ECCERR5 (R/W1C)
ECC Error Bank 5
ECCERR4 (R/W1C)
ECC Error Bank 4
ECCERR3 (R/W1C)
ECC Error Bank 3
ECCERR2 (R/W1C)
ECC Error Bank 2
ERR0 (R/W1C)
Error Port 0
ERR1 (R/W1C)
Error Port 1
RFRS (R)
Refresh Register Status
ECCERR0 (R/W1C)
ECC Error Bank 0
ECCERR1 (R/W1C)
ECC Error Bank 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Figure 9-20: L2CTL_STAT Register Diagram

Table 9-21: L2CTL_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	ECCERR7	ECC Error Bank 7. The `L2CTL_STAT.ECCERR7` bit indicates that an ECC double-bit error occurred inside L2 bank 7.	
		0	No Status
		1	ECC Double Bit Error
14 (R/W1C)	ECCERR6	ECC Error Bank 6. The `L2CTL_STAT.ECCERR6` bit indicates that an ECC double-bit error occurred inside L2 bank 6.	
		0	No Status
		1	ECC Double Bit Error
13 (R/W1C)	ECCERR5	ECC Error Bank 5. The `L2CTL_STAT.ECCERR5` bit indicates that an ECC double-bit error occurred inside L2 bank 5.	
		0	No Status
		1	ECC Double Bit Error

Table 9-21: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	ECCERR4	ECC Error Bank 4. The `L2CTL_STAT.ECCERR4` bit indicates that an ECC double-bit error occurred inside L2 bank 4.	
		0	No Status
		1	ECC Double Bit Error
11 (R/W1C)	ECCERR3	ECC Error Bank 3. The `L2CTL_STAT.ECCERR3` bit indicates that an ECC double-bit error occurred inside L2 bank 3.	
		0	No Status
		1	ECC Double Bit Error
10 (R/W1C)	ECCERR2	ECC Error Bank 2. The `L2CTL_STAT.ECCERR2` bit indicates that an ECC double-bit error occurred inside L2 bank 2.	
		0	No Status
		1	ECC Double Bit Error
9 (R/W1C)	ECCERR1	ECC Error Bank 1. The `L2CTL_STAT.ECCERR1` bit indicates that an ECC double-bit error occurred inside L2 bank 1.	
		0	No Status
		1	ECC Double Bit Error
8 (R/W1C)	ECCERR0	ECC Error Bank 0. The `L2CTL_STAT.ECCERR0` bit indicates that an ECC double-bit error occurred inside L2 bank 0.	
		0	No Status
		1	ECC Double Bit Error
4 (R/NW)	RFRS	Refresh Register Status. The `L2CTL_STAT.RFRS` bit indicates whether a refresh request is pending (in progress) or that there are no pending requests.	
		0	No Pending Requests
		1	Request Pending (Refresh in Progress)

Table 9-21: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ERR1	Error Port 1. The `L2CTL_STAT.ERR1` indicates whether the L2CTL has detected a bus access error on L2s bus port 1.	
		0	No Error
		1	Bus Access Error
0 (R/W1C)	ERR0	Error Port 0. The `L2CTL_STAT.ERR0` indicates whether the L2CTL has detected a bus access error on L2s bus port 0.	
		0	No Error
		1	Bus Access Error

Write Priority Count Register

The `L2CTL_WPCR` register stores the count value to be used for priority elevation for bus write channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SYSCLK_0 cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus write channel. A best practice is to program this register before initiating an L2 access.

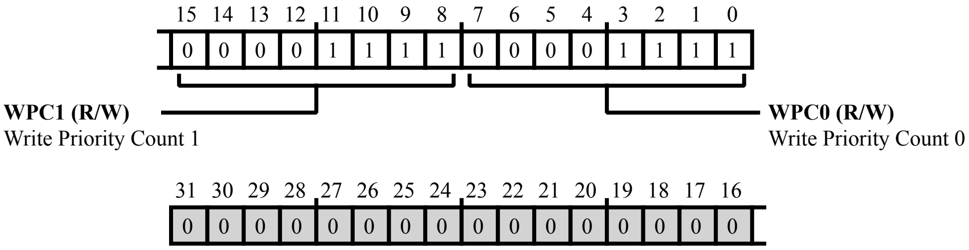

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
WPC1 (R/W)
Write Priority Count 1
WPC0 (R/W)
Write Priority Count 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 9-21: L2CTL_WPCR Register Diagram

Table 9-22: L2CTL_WPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	WPC1	Write Priority Count 1. The `L2CTL_WPCR.WPC1` bits hold the priority count for L2 bus write channel 0.
7:0 (R/W)	WPC0	Write Priority Count 0. The `L2CTL_WPCR.WPC0` bits hold the priority count for L2 bus write channel 1.

10 Dynamic Memory Controller (DMC)

The dynamic memory controller (DMC) provides a glueless interface between DDR3/DDR2/LPDDR SDRAMs and the system crossbar interface (SCB). The DMC enables execution of instructions from, as well as transfer of data to and from, DDR3, DDR2 SDRAM or LPDDR SDRAM respectively.

NOTE: The terms DDR2, DDR3, and LPDDR SDRAM are referred to generically as DDR SDRAM in the rest of this chapter unless otherwise noted.

NOTE: According to JEDEC STD specification (JESD79-3-1A.01) DDR3L marked devices (1.35V) can also be connected to ADSP-SC589 processors as these devices also meet 1.5V voltage level specifications of DDR3 marked devices.

The DMC is partitioned in a manner that allows reconfiguration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the *protocol controller*. An access and operation reordering mechanism is incorporated as an *efficiency controller*. An SCB slave interface is provided to interface with the on-chip interconnect. This interface results in an efficient slave implementation owing to its out-of-order transaction capabilities. The control and status registers present in the DMC can be accessed using the MMR access bus.

The DMC supports access to the external memory by core and DMA accesses.

DMC Features

The DMC includes a protocol controller that supports:

- JESD79-3E compatible double data rate DDR3 SDRAM devices
- JESD79-2E compatible DDR2 SDRAM devices
- JESD209A low-power DDR (LPDDR) SDRAM devices

The features of the dynamic memory controller are:

- Provides 16-bit data only interface to SDRAM devices
- Supports a single external rank (one chip select)
- Supports various burst lengths

- Provides page hit detection that supports multiple column accesses to the same row
- User-specified active, precharge, and refresh commands.
- Programmable SDRAM access timing parameters
- Enables automatic refresh generation with programmable refresh intervals
- Self-refresh mode to reduce system power consumption
- Efficient transaction processing to improve throughput and bandwidth using:
    - Software programmable SCB IDs to allow SCB ID-based priority
    - The ability to postpone up to eight auto-refresh commands
    - Software selectable closed page scheme on a per bank basis
    - Simple transaction scheduling mechanism to reduce read write turnaround frequency on the memory bus
    - Accesses with the same SCB ID are scheduled back-to-back to take advantage of same page access in SDRAM
    - Caching of SDRAM read data burst for specific masters to reduce the latency for same burst accesses.

The DDR2 features are:

- 256M bit to 4G-bit device sizes
- Burst length BL = 4 or 8
- Support for additive latency
- Support for programmable ODT and drive impedance from memory end
- Support for programmable (and ZQ calibration) ODT and drive impedance from the processor end

The DDR 3 features are:

- 512 Mb to 8 Gb device sizes
- Burst length BL = 8
- Support for additive latency
- Support for programmable (and ZQ calibration) ODT and drive impedance
- Support for programmable (and ZQ calibration) ODT and drive impedance from the processor end

The LPDDR features are:

- 64M bit to 2G-bit device sizes
- Burst length BL = 4 or 8
- Support for deep power-down mode

Feature Exclusions

The DMC exclusions are as follows:

For DDR2:

- 4-bit and 8-bit wide DDR2 DRAM memories are not supported
- OCD is not supported
- Burst interleaved accesses are not supported
- Single-ended signaling mode not supported

For DDR3:

- 4-bit and 8-bit wide DDR2 DRAM memories are not supported
- Burst interleaved accesses are not supported
- Both burst chop-BC4 and BC4-on-the-fly are not supported
- Auto refresh pull-in is not supported
- Write leveling is not supported
- DLL off mode is not supported

For LPDDR:

- 32-bit wide LPDDR memory devices are not supported
- Status register read (SRR) is not supported
- Sampling the optional temperature output (TQ) signal is not supported
- Clock stop mode is not supported
- Bursts of 2 and 16 words are not supported
- No support for BURST_TERMINATE command
- Dual-die, two CS# and two CKE packages are not supported

DMC Functional Description

The dynamic memory controller consists of master and slave interfaces, a protocol controller, and an efficiency controller. The following sections describe the function of these interfaces and controllers.

ADSP-SC58x DMC Register List

The Dynamic Memory Controller module (DMC) provides an interface to external double-data-rate SDRAM. This interface supports various DDR standards (see chapter descriptions). A set of registers governs DMC controller operations. For more information on DMC controller functionality, see the DMC Controller Register Descriptions.

Table 10-1: ADSP-SC58x DMC Register List

Name	Description
DMC_CFG	Configuration Register
DMC_CPHY_CTL	Controller to PHY Interface Register
DMC_CTL	Control Register
DMC_DLLCTL	DLL Control Register
DMC_DT_CALIB_ADDR	Data Calibration Address Register
DMC_DT_DATA_CALIB_DATA0	Data Calibration Data 0 Register
DMC_DT_DATA_CALIB_DATA1	Data Calibration Data 1 Register
DMC_EFFCTL	Efficiency Control Register
DMC_EMR1	Shadow EMR1 DDR2 Register
DMC_EMR2	Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)
DMC_MR	Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3)
DMC_MR1	Shadow MR1 Register (DDR3)
DMC_MR2	Shadow MR2 Register (DDR3)
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

ADSP-SC58x DMC Register List

A set of registers governs DMCPHY operations. For more information on functionality, see the register descriptions.

Table 10-2: ADSP-SC58x DMC Register List

Name	Description
DMC_CAL_PADCTL0	Calibration PAD Control 0 Register
DMC_CAL_PADCTL2	Calibration PAD Control 2 Register
DMC_PHY_CTL0	PHY Control 0 Register
DMC_PHY_CTL1	PHY Control 1 Register
DMC_PHY_CTL2	PHY Control 2 Register
DMC_PHY_CTL3	PHY Control 3 Register
DMC_PHY_CTL4	PHY Control 4 Register

Protocol Controller

The DDR SDRAM protocol controller translates memory access requests from the SCB (system crossbar) interface to JEDEC protocol-specific transactions used by DDR SDRAM devices.

The protocol controller ensures that the various timing parameters are met before reading and writing the SDRAM. The controller also performs the SDRAM initialization sequence as mandated by the standard. The protocol controller can:

- Issue reads and writes
- Precharge a row in a bank
- Activate a row in a bank
- Put the SDRAM devices in self-refresh and power-down modes

The protocol controller takes mode register writes from the MMR interface and translates them into mode register writes to SDRAM. Writing into the mode register is restricted through a mask register.

Efficiency Controller

The efficiency controller controls the ordering of transfers buffered in the read and write command buffers. It attempts to order transfers to optimize the available memory bandwidth. The DMC uses a number of schemes, described in the following sections, to increase the throughput.

Page-Based Scheduling

The DMC parses each write and read transaction that it buffered and gets the information of the row (page) and bank address. The protocol controller maintains the information about the pages that are opened in each bank. The efficiency controller uses the information about the opened pages while scheduling the buffered transactions. The transactions to the opened pages are given higher priority than the other outstanding transactions.

Same Master Transaction Scheduling

The DMC also stores the ID of each transaction that it buffered. In most of the cases, the transactions related to a master result in page hits from the locality of reference rule. The efficiency controller uses the ID information of the transactions while scheduling. When the page-based scheduling of the buffered transactions is complete, same master transaction scheduling is triggered. If multiple transactions from a master are received, the efficiency controller schedules the transactions back-to-back.

DMC Read Data Buffer

The DMC read data buffer contains a data buffer and an address buffer. The depth of the data buffer is equal to the burst length that is programmed in SDRAM. The address buffer holds the corresponding SDRAM burst address. When an SDRAM write address from any master matches an address in the DMC read data buffer, the DMC invalidates the related data in the read buffer. When the `DMC_RDDATABUFMSK1` or `DMC_RDDATABUFMSK2` register is programmed with a value other than zero, the DMC read data buffer operation is enabled. The set of masters whose data is buffered and retrieved are programmed in the `DMC_RDDATABUFID1` or `DMC_RDDATABUFID2` registers. The DMC can use the `DMC_RDDATABUFMSK1` and `DMC_RDDATABUFMSK2` ID registers to select a set of masters similar to the programming of the `DMC_PRIOMSK` and `DMC_PRIOMSK2` registers.

See the SCB ID-Based Priority section for details.

Closed Page Per Bank

The `DMC_EFFCTL` register provides per-bank granularity for closing pages. The software can determine that most accesses to a given bank in memory always result in a missed page. In this case, set the `PREC_BANK` bit corresponding to the required bank to close the row after every transfer. This proactive step can result in reduced thrashing and increases memory throughput.

SCB ID-Based Priority

The primary goal of the dynamic memory controller is to improve sustainable memory system bandwidth so that the service time for the average request can be reduced. However, to service critical requests from any master in the system, the DMC provides a mechanism to elevate priority of a given access. The DMC priority ID registers (`DMC_PRIO` and `DMC_PRIO2`) can be programmed with up to two SCB IDs with elevated priority.

After every access in a snapshot, the command buffers are searched to determine whether a commands ID matches with the ID programmed in the `DMC_PRIO` and `DMC_PRIO2` registers. The priority SCB ID access is sent before the subsequent access in the snapshot if:

- A match occurs, and
- The direction of the access (for example write) is the same as the direction of the snapshot (write)

There is an alternative to providing priority to a specific SCB ID. If a number of IDs from the same master require priority, program the DMC priority mask ID registers (`DMC_PRIOMSK` and `DMC_PRIOMSK2`) so that the corresponding bits are 0. The DMC uses a combination of the `DMC_PRIO` and `DMC_PRIO2` registers and the `DMC_PRIOMSK`/`DMC_PRIOMSK2` registers to elevate the priority of a select few or all IDs that belong to a master. By default, none of the IDs are prioritized. The following are a few possibilities.

- The `DMC_PRIOMSK` field is set to 0x00000000. If a single ID (7234) needs priority, set the `DMC_PRIOMSK` field to 0xFFFFFFFF and set the `DMC_PRIO` field to 7234.
- If the `DMC_PRIOMSK` field is set to 0xFFFFFFFE, the SCB IDs 7234 and 7235 are given priority.

- If the `DMC_PRIOMSK` field is set to 0xFFFFFFFC, the SCB IDs 7234, 7235, 7236 and 7237 are given priority.
- If two transactions with priority, one read and the other a write, are outstanding, the priority transaction that does not change the direction of the DMC access gets priority.
- The other priority transaction is handled at the beginning of the next snapshot. For example, if a write snapshot is in progress, the write priority transaction is sent. The read priority transaction is sent at the beginning of the next read snapshot.

NOTE: Use SCB-ID-based priority judiciously because it can significantly reduce the throughput of the DMC.

Delaying up to Eight Auto-Refresh Commands

The DMC uses this method to ensure that auto-refresh does not interfere with any critical data transfers. Up to eight auto-refresh commands can accumulate in the DMC. The exact number of auto-refresh commands can be programmed using the `DMC_EFFCTL.NUMREF` bit.

After the first refresh command is accumulated, the DMC constantly looks for an opportunity to schedule a refresh command. When the SCB read and write command buffers become empty for the programmed number of clock cycles (`DMC_EFFCTL.IDLECYC` bit field), the accumulated number of refresh commands are sent back-to-back to the DRAM. (The empty state of the SCB command buffers implies that no access is outstanding.)

After every refresh, the SCB command buffers are checked to ensure that they stay empty. If the SCB command buffers are always full, once the programmed number of refresh commands accumulates, the refresh operation is elevated to urgent priority. One refresh command is sent immediately. After this process, the DMC continues to wait for an opportunity to send out refresh commands. If self-refresh mode is enabled, all pending refresh commands are given out only after that DMC enters into self-refresh mode.

Page and Bank Interleaving

Page and bank interleaving allow consecutive row accesses to fall into the same bank (bank interleaving) or into a different bank (page interleaving). The DMC uses bank interleaving by default (`DMC_CTL.ADDRMODE` bit =0). If the `DMC_CTL.ADDRMODE` bit =1, the DMC uses page interleaving. Page misses in one addressing mode result in hits in the other addressing mode.

System Crossbar Slave Interface

The DMC uses the system crossbar slave interface to move all data. The system crossbar interface accepts interleaved write transactions and sends out-of-order responses. The read and write interfaces consist of buffers for address, data, and control information transferred to or from the system crossbar bus.

The system crossbar interface transactions are sent to the SDRAM only after the SDRAM has been initialized. However, if transactions arrive before or during initialization, they accumulate in the system crossbar interface and are sent out to the protocol controller once the initialization completes.

To increase throughput, the system crossbar write-response is sent out as soon as the final DDR burst is scheduled for transfer into the SDRAM. However, if an auto-refresh is needed, the scheduled write data is sent only after the auto-refresh. A delay can occur. The delay is a maximum of 64 clock cycles from the moment the write response is sent on the SCB to the write operation of the data into SDRAM.

The system crossbar interface performs the following operations:

- Buffers read and write command requests from the system crossbar bus
- Processes the requests by converting them to protocol controller user-interface transfers
- Sends and receives data to or from the protocol controller
- Creates a suitable read/write response and sends read data back to the system crossbar bus

The system crossbar slave interface supports the following:

- All burst lengths (1–16)
- Incremental and wrap bursts
- Data transfer sizes of 8-bit, 16-bit, or 32-bit
- Arrival of write data before write address
- Generation of error responses which include:
  - Any access to an unimplemented region of the external memory space
  - Any access when the SDRAM is in self-refresh, power-down, or deep power-down mode (in case of LPDDR)
  - Any access when the direct command interface is in operation

Read/Write Command and Data Buffers

The system crossbar interface consists of a four-deep read command buffer and a four-deep write command buffer. Up to four write commands and four read commands can be waiting for access to the SDRAM. The system crossbar write buffer is 32 deep. It can support write data interleaving of two. The system crossbar read buffer is 32 deep.

Peripheral Bus Slave Interface

The peripheral bus slave interface connects the dynamic memory controller to the peripheral bus and provides a host controller with access to the registers. The peripheral bus slave interface supports the following features:

- Read and write word accesses
- 32-bit data bus

Architectural Concepts

The following sections provide information on the architecture of the interface.

Controller On Die Termination (ODT)

The controller ODT is enabled with the granularity of a byte lane. The description of this feature can be obtained in the description of the corresponding PHY registers. Controller ODT involves extra overhead in terms of power consumption during reads.

The DMC implements dynamic on die termination at processor pads. When controller ODT is enabled, the termination resistors in the pads are turned on when the controller reads data from the DRAM. These resistors are turned off when the controller writes to the DRAM.

Mode Register Set and Extended Mode Register Set Command

The load mode register command initializes the SDRAM operation parameters. The DMC supports the mode register set and extended mode register set commands. The controller automatically issues the mode register set command during power-on initialization and also when the `DMC_MR` register is written with the `DMC_MSK.MR` bit. The mode register set command is sent after the ongoing data transfer completes.

The DMC automatically issues the mode register set command when the shadow EMR1/EMR2/EMR3 registers are written. The corresponding `DMC_MSK.EMR3DMC_MSK.EMR2`/`DMC_MSK.EMR1` bits must be enabled.

DDR3 Reset Functionality

DDR3 contains an additional pin corresponding to reset functionality. Reset is part of the initialization sequence but it can be performed asynchronously when needed. The reset procedure is similar to the steps involved in the initialization except the initial part of power-up.

To perform reset on the DDR3 module:

1. Check to ensure the module is in the idle state by polling the `DMC_STAT.IDLE` bit (0x0008).
2. Set the `DMC_CTL.RESET` bit (0x0004).
3. Monitor the `DMC_STAT.RESETDONE` bit for the completion of the reset function.

Do not perform any transactions during a module reset. Wait for the `DMC_STAT.RESETDONE` signal.

DDR3 SDRAM Organization

The DMC supports DDR3 SDRAM memory modules ranging from 512 Mb to 8 Gb. The following tables list the address translation mechanism from the user interface to DDR3 memory interface. The controller also supports two types of addressing modes: bank interleaving (`DMC_CTL.ADDRMODE` =1) and page interleaving (`DMC_CTL.ADDRMODE` =0).

Bank Interleaving

The ***DDR3 Bank Interleaving*** table shows DDR3 bank interleaving.

Table 10-3: DDR3 Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
512 Mb	25:24	23:11	10:1

Table 10-3: DDR3 Bank Interleaving (Continued)

SDRAM size	Bank address bits	Row address bits	Column address bits
1 Gb	26:24	23:11	10:1
2 Gb	27:25	24:11	10:1
4 Gb	28:26	25:11	10:1
8 Gb	29:27	26:11	10:1

Page Interleaving

The ***DDR3 Page Interleaving*** table shows DDR3 page interleaving.

Table 10-4: DDR3 Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1
2 Gb	27:14	13:11	10:1
4 Gb	28:14	13:11	10.1
8 Gb	29:14	13:11	10.1

DDR2 SDRAM Organization

The DMC supports DDR2 SDRAM memory modules ranging from 256M bit to 4G bit. The following tables list the address translation mechanism from the user interface to DDR2 memory interface. The controller also supports two types of addressing modes: bank interleaving (`DMC_CTL.ADDRMODE` =1) and page interleaving (`DMC_CTL.ADDRMODE` =0).

Bank Interleaving

The ***DDR2 Bank Interleaving*** table shows DDR2 bank interleaving.

Table 10-5: DDR2 Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
256 Mb	24:23	22:10	9:1
512 Mb	25:24	23:11	10:1
1 Gb	26:24	23:11	10:1
2 Gb	27:25	24:11	10:1
4 Gb	28:26	25:11	10:1

Page Interleaving

The ***DDR2 Page Interleaving*** table shows DDR2 page interleaving.

Table 10-6: DDR2 Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
256 Mb	24:12	11:10	9.1
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1
2 Gb	27:14	13:11	10:1
4 Gb	28:14	13:11	10.1

LPDDR SDRAM Organization

The DMC supports LPDDR SDRAM memory modules ranging from 64M bit to 2G bit. The following tables list the address translation mechanism from the user interface to LPDDR memory interface.

The controller also supports two types of addressing modes: bank interleaving (`DMC_CTL.ADDRMODE`=1) and page interleaving (`DMC_CTL.ADDRMODE`=0).

Bank Interleaving

The ***LPDDR Bank Interleaving*** table shows LPDDR bank interleaving.

Table 10-7: LPDDR Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
64 Mb	22:21	20:9	8:1
128 Mb	23:22	21:10	9:1
256 Mb	24:23	22:10	9:1
512 Mb	25:24	23:11	10:1
1 Gb	26:24	23:11	10:1
2 Gb	27:26	25:12	11:1

Page Interleaving

The ***LPDDR Page Interleaving*** table shows LPDDR page interleaving.

Table 10-8: LPDDR Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
64 Mb	22:11	10:9	8.1
128 Mb	23:12	11:10	9:1
256 Mb	24:12	11:10	9:1
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1

Table 10-8: LPDDR Page Interleaving (Continued)

SDRAM size	Row address bits	Bank address bits	Column address bits
2 Gb	27:14	13:11	10.1

DMC Clocking

The DMC uses a divided-down version of the *PLLCLK* (PLL clock) to generate an internal clock for clocking the DMC block and interface. The specific value of the *DCLK* frequency is programmed in the `CGU_DIV` register. The section Initializing the DMC (ADSP-SC58x) describes the procedure.

For information on the maximum clock frequency supported for specific modes, refer to the processor data sheet.

NOTE: For the processor variants that have two DMC blocks, both blocks run on the same DCLK frequency.

NOTE: In some cases, it might be required to generate a DCLK frequency asynchronous to CCLK (for example, CCLK=450 MHz and DCLK=400 MHz). For these cases, one CGU can be used to generate CCLK and another can be used to generate DCLK. For more details, refer to the CGU chapter.

DMC DMA

The DMC supports DMA-based transfers to and from external DDR SDRAM memory and internal memory.

The DMC DMA controller, part of the distributed DMA engines (DDE) that are dispersed through the infrastructure, connects to the system crossbar fabric.

The DMC uses two DDEs for memory-to-memory DMA (MDMA). One channel is the source channel, and the second, the destination channel.

DMA transfers on the processor are descriptor-based or register-based. Register-based DMA allows the processor to program DMA control registers directly to initiate a DMA transfer. On completion, the control registers can be automatically updated with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. This transfer allows the chaining together of multiple DMA sequences. In descriptor-based DMA operations, a DMA channel can be programmed to set up and start another DMA transfer automatically after the current sequence completes.

Refer to the DMA chapter for further details.

Enhanced DMA operations (such as delay line DMA, scatter or gather DMA) are also supported to or from the DMC module. For more details, refer to the Extended Memory DMA (EMDMA) chapter.

DMC Operating Modes

By default, the DMC is in DDR2 mode. To enable DDR3 or LPDDR mode, the corresponding bits in the `DMC_CTL` and `DMC_PHY_CTL4` register must be configured.

DDR2 Mode

This mode is the default mode of the DMC module and supports JESD79-2E compatible DDR2 SDRAM. In this mode, the `DMC_CTL.DDR3EN` bit =0, the `DMC_CTL.LPDDR` bit =0, and the `DMC_PHY_CTL4.DDRMODE` bit field is 0b'01.

DDR3 Mode

The DMC module supports JESD79-3E compatible double data rate DDR3 SDRAM. To configure this mode of operation, first set (=1) the `DMC_CTL.DDR3EN` bit and set the `DMC_PHY_CTL4.DDRMODE` bit field to 0b'00.

LPDDR Mode

The DMC module supports JESD209A low-power DDR (LPDDR) SDRAM. To configure this mode of operation, set (=1) the `DMC_CTL.LPDDR` bit and set the `DMC_PHY_CTL4.DDRMODE` bit field to 0b'11.

Deep Power-Down Mode

The DMC module supports JESD209A low-power DDR (LPDDR) SDRAM. To configure this mode of operation, set (=1) the `DMC_CTL.LPDDR` bit and set the `DMC_PHY_CTL4.DDRMODE` bit field to 0b'11.

When the processor does not require the data stored in SDRAM (assume reset state of SDRAM), the DMC can put the SDRAM in deep power-down mode. When the DMC is in deep power-down, any data accesses cause the DMC to generate a bus error. To configure this mode, set (=1) the `DMC_CTL.DPDREQ` bit when low-power DMC operation is enabled (`DMC_CTL.LPDDR` =1).

The `DMC_STAT.IDLE` bit indicates the activity in the DMC. If this bit is set, there is no activity all through the DMC. Deep power can be entered by setting the `DMC_CTL.DPDREQ` bit. The `DMC_STAT.DPDACK` bit is asserted when the SDRAM enters deep power-down mode. The DMC stays in deep power-down mode as long as the `DMC_CTL.DPDREQ` bit is asserted.

Clearing (=0) the `DMC_CTL.DPDREQ` bit causes the DMC to exit deep power-down mode. Also, when exiting deep power-down mode, the controller clears the `DMC_STAT.DPDACK` bit. The user must re-initialize the DMC after it comes out of deep power-down mode.

Self-Refresh Mode

For low-power consumption, the SDRAM can be put in self-refresh mode. When no data activity occurs, the DMC can put the SDRAM in self refresh to save power. The `DMC_STAT.IDLE` bit indicates the activity on the DMC. If this bit is set, there is no activity in the DMC.

Enable self-refresh mode by writing the `DMC_CTL.SRREQ` bit. The DMC stays in a self-refresh state as long as this bit is asserted. The `DMC_STAT.SRACK` bit indicates when the SDRAM enters self-refresh mode.

When the DMC is in self-refresh mode, the DMC generates an SCB error when any data accesses (read or write requests) is requested.

The DMC can be brought out of self-refresh mode by clearing the `DMC_CTL.SRREQ` bit again. The controller clears the `DMC_STAT.SRACK` bit after the self-refresh operation completes.

DMC Event Control

The DMC has no related interrupt or trigger event information.

DMC Programming Model

The dynamic memory controller contains five groups of memory-mapped registers. The DMC uses the MMR access bus to connect to these registers.

- Control and status registers. These registers control the various operation modes of the dynamic memory controller and provide status.
- Timing parameter registers. The value programmed in these registers depends on the speed grade of the SDRAM device used.
- Mode register mirror registers. These shadow registers are copies of the mode registers residing in the SDRAM device.
- PHY control and status registers. The DMC uses these registers to control the operation of the PHY.
- PAD control registers. The DMC uses these registers to control the various aspects of the I/O pads.

The DMC control registers contain sensitive timing parameters and settings for the DDR SDRAM. These registers are programmed with values that are in the operating range of the DDR used.

Writing to reserved fields or writing any reserved values in register bits can cause the dynamic memory controller to function erroneously.

Programming Considerations for DDR2, DDR3, and LPDDR Memory

The ***DDR2, DDR3, and LPDDR Programming*** table shows important programming considerations and differences across DDR2, DDR3, and LPDDR memory technologies. The table serves as a quick reference when configuring the DMC and PHY registers.

Table 10-9: DDR2, DDR3, and LPDDR Programming

PHY/Controller	Description	Registers and Bit Fields Involved	DDR3	DDR2	LPDDR
PHY	Enabling DDR3/DDR2/LPDDR modes	`DMC_PHY_CTL4`	Select DDR3 mode by setting the `DMC_PHY_CTL4.DDRMODE` bit field to 00.	Select DDR2 mode by setting the`DMC_PHY_CTL4.DDRMODE` bit field to 01.	Select LPDDR mode by setting the `DMC_PHY_CTL4.DDRMODE` bit field to 11.

Table 10-9: DDR2, DDR3, and LPDDR Programming (Continued)

PHY/Controller	Description	Registers and Bit Fields Involved	DDR3	DDR2	LPDDR
PHY	ODT and drive impedance calibration	DMC_CAL_PADCTL0, DMC_CAL_PADCTL2	The DMC supports ODT and drive impedance calibration. Configure the DMC_CAL_PADCTL0 and DMC_CAL_PADCTL2 registers accordingly. For details, refer PAD Calibration for Driver Impedance and On Die Termination (ODT) section. Programming driver impedance is required. Programming ODT is optional. To disable ODT, set the DMC_PHY_CTL1.BYPODTEN bit.	The DMC supports OCD calibration. Configure the DMC_CAL_PADCTL0 register accordingly.	The DMC does not support ODT and drive impedance calibration. No need to program the DMC_CAL_PADCTL0 and DMC_CAL_PADCTL2 registers. Must make sure that the DMC_PHY_CTL1.BYPODTEN bit is set to bypass the processor ODT settings.
Controller	Enabling DDR3/DDR2/LPDDR modes	DMC_CTL.DDR3EN, DMC_CTL.LPDDR	Select DDR3 mode by setting the DMC_CTL.DDR3EN bit. Make sure that the bit DMC_CTL.LPDDR is cleared.	Default is DDR2 mode. Make sure that both the bits DMC_CTL.LPDDR and DMC_CTL.DDR3EN are cleared.	Select LPDDR mode by setting the DMC_CTL.LPDDR bit. Make sure that the bit DMC_CTL.DDR3EN is cleared.
Controller	Configuring DMC_CTL.RDTOWR bit field	DMC_CTL.RDTOWR	Make sure that the DMC_CTL.RDTOWR bit field is always set to 010 (=2 in decimal).		
Controller	Configuring DMC_CFG register fields	DMC_CFG	Make sure that the DMC_CFG.IFWID and DMC_CFG.SDRWID bit fields are always set to 0010 (=2 in decimal) as the DMC only supports 16-bit wide interface and SDRAM widths. Make sure that the bit field DMC_CFG.EXTBANK is always set to 0000 as the DMC only supports one external bank. Select the DMC_CFG.SDRSIZE as per the SDRAM size. Supported sizes are 64 Mb to 2 Gb for LPDDR, 256 Mb to 4 Gb for DDR2, and 512 Mb to 8 Gb for DDR3.		
Controller	Configuring controller timing parameter registers	DMC_TR0, DMC_TR1, DMC_TR2	Configure the parameters t_{RCD}, t_{WTR}, t_{RP}, t_{RAS}, t_{MRD}, t_{REF}, t_{RFC}, t_{RRD}, t_{WR}, t_{XP}, t_{CKE} (DDR3/DDR2/LPDDR), and t_{FAW}, t_{RTP} (DDR3/DDR2) in terms of DCLK cycles.		

Table 10-9: DDR2, DDR3, and LPDDR Programming (Continued)

PHY/Controller	Description	Registers and Bit Fields Involved	DDR3	DDR2	LPDDR
Controller	Configuring burst length	`DMC_MR.BLEN` (for DDR2 and LPDDR), DMC_MR0.BLEN for DDR3	The DMC only supports burst length of 8. Configure the DMC_MR0.BLEN field to 00 only.	The DMC supports both burst length of 4 and 8. Configure `DMC_MR.BLEN` field to 10 for burst length of 4 and to 11 for burst length of 8 words.	The DMC supports both burst length of 4 and 8. Configure `DMC_MR.BLEN` field to 10 for burst length of 4 and to 11 for burst length of 8 words.
Controller	Configuring CAS latency	`DMC_MR.CL` (for DDR2 and LPDDR), DMC_MR0.CL0, and DMC_MR0.CL (for DDR3)	The DMC supports CAS latencies of 5 to14. Refer to the `DMC_MR.CL` register description for more details.	The DMC supports CAS latencies of 3 to 6. Refer to the `DMC_MR` register description for more details.	The DMC supports CAS latency of 3. Refer to the `DMC_MR` register description for more details.
Controller	Configuring the `DMC_MR.DLLRST` bit (DDR2/LPDDR) or DMC_MR0(DDR3)	`DMC_MR.DLLRST`/ DMC_MR0.DLLRST	Set this bit while performing the initialization	Setting of this bit is optional for DDR2.	Reserved
Controller	Write recovery timing	`DMC_MR.WRRECOV` and DMC_MR0.WRRECOV	The DMC supports WR values of 5 to 16. Refer to the DMC_MR0 register description for more details.	The DMC supports WR values of 2 to 8. Refer to the `DMC_MR` register description for more details.	Reserved
Controller	Configuring DMC_MR1(DDR3) or `DMC_EMR1` (DDR2) register	DMC_MR1(DDR3) or `DMC_EMR1` (DDR2)	The DMC can use this register to configure memory drive impedance, ODT, and additive latency parameters. Refer to the corresponding register descriptions for more details.		This register is not used for LPDDR programming.
Controller	Configuring DMC_MR2(DDR3) or `DMC_EMR2` (DDR2) or DMC_EMR (LPDDR) register	DMC_MR2(DDR3) or `DMC_EMR2`(DDR2) or DMC_EMR (LPDDR)	Configure the write latency field (CWL) to the required value. The DMC can also use this register to enable Auto Self-Refresh (ASR) and to select Self-Refresh Temperature (SRT) functionalities in the memory device. For more details on these functionalities, refer to the	The DMC can use this register to configure the Partial Array Self-Refresh (PASR) and High Temperature Self-Refresh Rate Enable (SRF) functionalities of DDR2 memory device. For more details on these functionalities, refer to the DDR2 memory device data sheet.	The DMC can use this register to configure the Partial Array Self-Refresh (PASR), Temperature compensated self-refresh (TCSR), and drive strength (DS) functionalities of the memory device. For more details on these functionalities, refer to the LPDDR memory device data sheet.

Table 10-9: DDR2, DDR3, and LPDDR Programming (Continued)

PHY/Controller	Description	Registers and Bit Fields Involved	DDR3	DDR2	LPDDR
			DDR3 memory device data sheets.		
Controller	Configuring the `DMC_DLLCTL` register	`DMC_DLLCTL`	Always configure the `DMC_DLLCTL.DLLCALRDCNT` field to 0x48 and `DMC_DLLCTL.DATACYC` field to 0x9.		

PHY DLL Calibration

The PHY DLL calibration is performed as part of the SDRAM power-up initialization. It calibrates data against the DQS and CLK signal. However, running DLL calibration after self-refresh or at an arbitrary time is required in certain cases.

The DMC allows PHY DLL calibration to start by setting the `DMC_CTL.DLLCAL` bit. The `DMC_STAT.DLLCALDONE` bit can be used to monitor the progress of the calibration. Once calibration is over, this bit is set. Once the calibration procedure is started by writing to the `DMC_CAL_PADCTL0.CALSTRT` bit, the full calibration takes 300 DCLK cycles to complete.

NOTE: DLL calibration can be initiated only when the DMC is idle (`DMC_STAT.IDLE`= 1).

DDR2 OCD Calibration

OCD calibration is not supported by the ADSP-SC58x processors.

DDR3 ZQ Calibration Short CMD

The ZQ calibration short command is generally used to correct small variations in ZQ (approximately 0.5%). To perform ZQ calibration, the controller is first checked for its idle state. Once the idle bit is obtained, a ZQCS command can be issued by setting the `DMC_STAT.ZQCSDONE` bit (0x0004). The `DMC_STAT.ZQCSDONE` bit (0x0008) can be used to monitor the calibration sequence. When this bit is “0”, it indicates that the calibration is ongoing. When it is “1”, it indicates that the calibration is done. As an example, a GP timer can be used to periodically trigger a ZQCS command to address tiny variations.

NOTE: The ZQ calibration function is essential for normal operation of DDR3. With the reference of the external resistance (240Ω ± 1%) connected to the DMC_RZQ pin, DDR3 calibrates the Ron and Rtt values of the ZQ pin against temperature and voltage variations.

DDR3 ZQ Calibration Long CMD

Several DDR3 impedance calibrations are implemented for optimal signal integrity. The long ZQ calibration is used after power-up and the short ZQ calibration is used periodically during normal operation to compensate for voltage and temperature drift. These calibration sequences improve connectivity between the SDRAM pads and the PCB trace. The DMC_RZQ pin on the SDRAM is connected to an external precision resistor that adjusts the output driver impedance Rtt and ODT values to match the trace impedance. The connection reduces impedance discontinuity and minimizes signal reflections.

The command has two variants named as ZQ calibration long (ZQCL) and ZQ calibration short (ZQCS). The ZQCL command is issued during initialization and after self-refresh exit command. It can be issued later depending on the system environment.

The DMC pads can be autocalibrated to the required driver impedance Rtt using an external resistance RZQ and the On Die Termination (ODT) value using the corresponding bits (`DMC_CAL_PADCTL2`). The autocalibration logic translates these values into a corresponding drive strength control inside the PHY and then routed to the PADS. Autocalibration starts as soon as the `DMC_CAL_PADCTL0.CALSTRT` bit is programmed (set the DCLK at the required frequency before setting this bit). Autocalibration expects the program to select two different member sets of pads (address/command pads versus CLK/Data/DQS/DM pads).

On Die Termination (DDR2/DDR3)

The DMC supports dynamic On Die Termination (ODT) at the pads. When the controller ODT is set, the termination resistors in the pads are turned on when the controller reads data from the DRAM. These resistors are turned off when the controller is writing to the DRAM. Controller ODT is enabled with the granularity of a byte lane. The description of this feature can be obtained in the description of the corresponding PHY registers.

ODT resistance Rtt is selectable in the same way as DDR2 SDRAM ([A9, A6, A2] in MR1, [A10, A9] in MR2)

DDR3 SDRAM inherits the ODT function provided for DDR2 SDRAM, and provides extended ODT mode.

Synchronous ODT: ODT timing is the same as that of DDR2 SDRAM

Asynchronous ODT: ODT timing in the slow exit power-down mode

Dynamic ODT: Function that can dynamically switch the ODT resistance during a write operation without an MRS command. It improves signal quality during a write operation.

Output Driver Impedance

Output driver impedance (Rtt) of DQ, DQS, /DQS/DM is selectable in the same way as DDR2 SDRAM ([A5, A1] in MR1). Rtt can fluctuate with the process, voltage, and temperature (PVT). DDR2 SDRAM can calibrate Rtt fluctuation due to PVT using the optional OCD (off-chip driver calibration) function. DDR3 SDRAM uses the ZQ calibration function instead of the OCD function.

In addition to the driver impedance, the bidirectional pads (Data and DQS) also require the initialization sequence to program the termination impedance by writing to the field `DMC_CAL_PADCTL2.IMPRTT`. The DMC pads use parallel termination, one branch goes from the pad to the I/O supply. The other branch goes to the I/O ground. The value programmed to this 8-bit field is the value to be used for each branch. There is a correction factor involved while programming this register. The value of this correction factor is 0.8. For example, suppose that a termination of 50 Ω is required on the data pads to match with the board trace. The value is programmed to 100 × 0.8 = 80, as the two parallel paths lead to an effective impedance of 50 ohms.

Initializing the DMC (ADSP-SC58x)

To initialize the DMC, use the following steps. If it is not the first time that the DMC initializes, check to first ensure that the DMC is idle and not in the midst of any activity.

If DMC initialization occurs for the first time after power-up, PHY and PAD initialization is a requirement. The initialization occurs with the following steps:

For LPDDR mode, set the `DMC_PHY_CTL4.DDRMODE` bits to 0b'11 (3 in decimal) and set the `DMC_PHY_CTL1.BYPODTEN` bit.

For DDR2/DDR3 modes, follow these steps to perform pad impedance calibration:

1. Set the device mode in the `DMC_PHY_CTL4.DDRMODE` to DDR2 or DD3.
2. Configure the required values in the `DMC_CAL_PADCTL0` and `DMC_CAL_PADCTL2` registers without setting the `DMC_CAL_PADCTL0.CALSTRT` bit.
3. Set the `DMC_CAL_PADCTL0.CALSTRT` bit.
4. Wait for 300 DCLK cycles for the PAD calibration to complete.

   Bits 0, 1, 2, 3 of the `DMC_PHY_CTL0` register and the bits 31 through 26 of the `DMC_PHY_CTL2` register are timing trim bits. Always set these bits for DDR2 and DDR3 modes. For example, set these bits during first-time DMC initialization. Then, software does not need to touch or clear these bits.

   Bits 0, 1, 2, 3 of the `DMC_PHY_CTL0` register and bits 26, 27, 28, 29, 30, and 31 of the `DMC_PHY_CTL2` register are timing trim bits. Always set these bits for DDR2 and DDR3 modes. For example, set these bits during first-time DMC initialization. Then, software does not need to touch or clear these bits.

   Use the following C code to set these bits for the first time DMC initialization:

```
*pREG_DMC0_PHY_CTL0 |= 0x0000000F;
*pREG_DMC0_PHY_CTL2 |= 0xFC000000;
```

NOTE: Bits 6, 7, 25 and 27 of the `DMC_PHY_CTL3` register should always be set for all the DDR modes (DDR3/DDR2/LPDDR). The program can set these bits during first-time DMC initialization. Then, software need not touch or clear these bits. Use the following C code to set these bits for the first time DMC initialization:

```
*pREG_DMC0_PHY_CTL3=0xA0000C0;
```

NOTE: For DDR3 mode, set bit 1 and configure bits [5:2] of the `DMC_CPHY_CTL` register with WL = CWL + AL in DCLK cycles. For example, in case of DMC0, if CWL =6 and AL =0, program the `DMC_CPHY_CTL` register with the value 0x0000001A.

Use the following steps for first-time DMC initialization and reinitialization:

1. Perform first-time DMC initialization, as needed.

   ADDITIONAL INFORMATION: Perform this step only for the first time DMC initialization after power-up or reset. Skip this step if reinitializing the DMC.

   a. Set the `DMC_PHY_CTL0.RESETDLL` bit of the `DMC_PHY_CTL0` register.

   b. Initialize the CGU to change the DCLK frequency.

   a. Clear the `DMC_PHY_CTL0.RESETDLL` bit of the `DMC_PHY_CTL0` register.

2. Reinitialize the DMC with a DCLK change

   a. Place the DMC in self-refresh mode.

   b. Set the `DMC_PHY_CTL0.RESETDLL` bit of the `DMC_PHY_CTL0` register.

   c. Initialize the CGU to change the DCLK frequency.

   d. Clear the `DMC_PHY_CTL0.RESETDLL` bit of the `DMC_PHY_CTL0` register.

   e. Bring the DMC out of self-refresh mode.

3. If not already done, wait 9000 DCLK cycles to ensure that the DLL locked.

4. Program the `DMC_CFG`, `DMC_CTL`, `DMC_TR0`, `DMC_TR1`, and `DMC_TR2` registers to the appropriate values to set proper SDRAM cycle timing options.

   ADDITIONAL INFORMATION: For example, t_{RAS}, t_{RC}, t_{RP}, t_{RCD}, t_{WR}, t_{FAW} are some of the parameters.

5. Program the shadow registers `DMC_MR` (DDR2/DDR3/LPDDR), `DMC_EMR1` (DDR2)/`DMC_EMR1`(DDR3), `DMC_EMR1` (DDR2)/`DMC_EMR1` (LPDDR) `DMC_EMR1` (DDR3), with the needed burst length, CAS latency, additive latency, and other parameters.

6. Finally, after programming these registers, write the `DMC_CTL.INIT` bit to the DMC control register to begin the power-up initialization sequence.

7. Wait for the SDRAM initialization sequence to complete by making sure that the `DMC_STAT.INITDONE` bit is set.

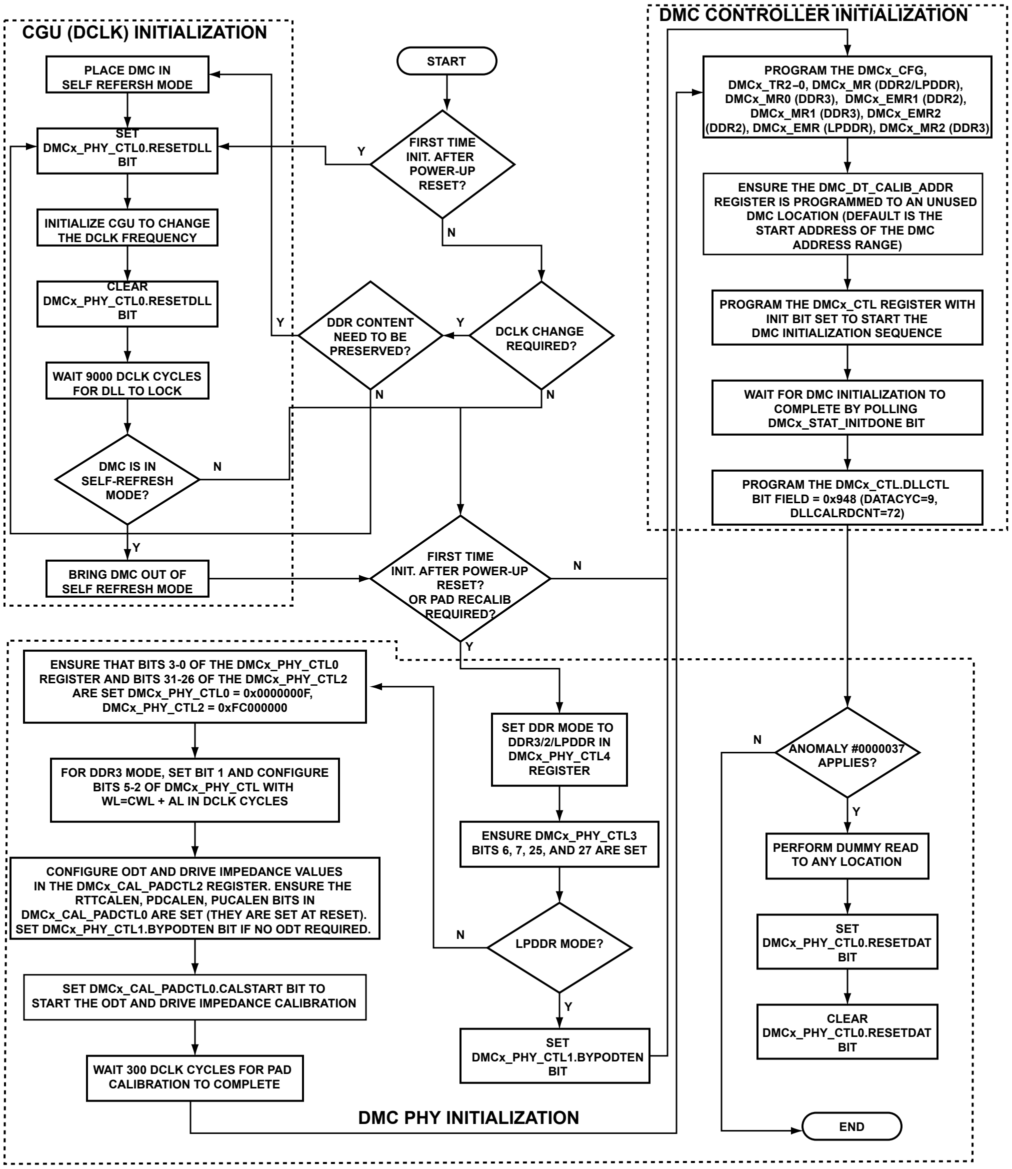

CGU (DCLK) INITIALIZATION
PLACE DMC IN SELF REFERSH MODE
SET DMCx_PHY_CTL0.RESETDLL BIT
INITIALIZE CGU TO CHANGE THE DCLK FREQUENCY
CLEAR DMCx_PHY_CTL0.RESETDLL BIT
WAIT 9000 DCLK CYCLES FOR DLL TO LOCK
DMC IS IN SELF-REFRESH MODE?
BRING DMC OUT OF SELF REFRESH MODE
START
FIRST TIME INIT. AFTER POWER-UP RESET?
DDR CONTENT NEED TO BE PRESERVED?
DCLK CHANGE REQUIRED?
FIRST TIME INIT. AFTER POWER-UP RESET? OR PAD RECALIB REQUIRED?
Y
N
DMC CONTROLLER INITIALIZATION
PROGRAM THE DMCx_CFG, DMCx_TR2–0, DMCx_MR (DDR2/LPDDR), DMCx_MR0 (DDR3), DMCx_EMR1 (DDR2), DMCx_MR1 (DDR3), DMCx_EMR2 (DDR2), DMCx_EMR (LPDDR), DMCx_MR2 (DDR3)
ENSURE THE DMC_DT_CALIB_ADDR REGISTER IS PROGRAMMED TO AN UNUSED DMC LOCATION (DEFAULT IS THE START ADDRESS OF THE DMC ADDRESS RANGE)
PROGRAM THE DMCx_CTL REGISTER WITH INIT BIT SET TO START THE DMC INITIALIZATION SEQUENCE
WAIT FOR DMC INITIALIZATION TO COMPLETE BY POLLING DMCx_STAT_INITDONE BIT
PROGRAM THE DMCx_CTL.DLLCTL BIT FIELD = 0x948 (DATACYC=9, DLLCALRDCNT=72)
ENSURE THAT BITS 3-0 OF THE DMCx_PHY_CTL0 REGISTER AND BITS 31-26 OF THE DMCx_PHY_CTL2 ARE SET DMCx_PHY_CTL0 = 0x0000000F, DMCx_PHY_CTL2 = 0xFC000000
FOR DDR3 MODE, SET BIT 1 AND CONFIGURE BITS 5-2 OF DMCx_PHY_CTL WITH WL=CWL + AL IN DCLK CYCLES
CONFIGURE ODT AND DRIVE IMPEDANCE VALUES IN THE DMCx_CAL_PADCTL2 REGISTER. ENSURE THE RTTCALEN, PDCALEN, PUCALEN BITS IN DMCx_CAL_PADCTL0 ARE SET (THEY ARE SET AT RESET). SET DMCx_PHY_CTL1.BYPODTEN BIT IF NO ODT REQUIRED.
SET DMCx_CAL_PADCTL0.CALSTART BIT TO START THE ODT AND DRIVE IMPEDANCE CALIBRATION
WAIT 300 DCLK CYCLES FOR PAD CALIBRATION TO COMPLETE
SET DDR MODE TO DDR3/2/LPDDR IN DMCx_PHY_CTL4 REGISTER
ENSURE DMCx_PHY_CTL3 BITS 6, 7, 25, AND 27 ARE SET
LPDDR MODE?
SET DMCx_PHY_CTL1.BYPODTEN BIT
DMC PHY INITIALIZATION
ANOMALY #0000037 APPLIES?
PERFORM DUMMY READ TO ANY LOCATION
SET DMCx_PHY_CTL0.RESETDAT BIT
CLEAR DMCx_PHY_CTL0.RESETDAT BIT
END

Figure 10-1: DMC Initialization Flow

The DMC accumulates system crossbar transactions that occur during or before initialization and sends them to SDRAM once the SDRAM initialization or DLL calibration is complete.

NOTE: During the DMC PHY DLL calibration, a particular set of locations in the DRAM is written followed immediately by a series of reads. The DMC PHY needs information about the data to be read during the PHY DLL calibration prior to the operation. The controller performs one burst write operation to the address programmed in the `DMC_DT_CALIB_ADDR` register. The exact address chosen does not matter during memory initialization.

If calibration of the PHY is performed when the DRAM contains valid data, ensure that this address points to an unused address. Otherwise, this operation modifies application data stored at the address selected. The DLL calibration modifies all 16 bytes corresponding to the 16-byte aligned address, even if the address programmed is not 16-byte aligned. For example, it updates all the locations 0x80000000 to 0x8000000F regardless of whether the address is programmed as 0x80000000 or 0x80000004.

The program performs second-time initialization for cases where the DMC has already been initialized. The initialization can be through a preload during a debug session, or through code executed during the booting process.

ADSP-SC58x DMC Register Descriptions

Dynamic Memory Controller (DMC) contains the following registers.

Table 10-10: ADSP-SC58x DMC Register List

Name	Description
DMC_CFG	Configuration Register
DMC_CPHY_CTL	Controller to PHY Interface Register
DMC_CTL	Control Register
DMC_DLLCTL	DLL Control Register
DMC_DT_CALIB_ADDR	Data Calibration Address Register
DMC_DT_DATA_CALIB_DATA0	Data Calibration Data 0 Register
DMC_DT_DATA_CALIB_DATA1	Data Calibration Data 1 Register
DMC_EFFCTL	Efficiency Control Register
DMC_EMR1	Shadow EMR1 DDR2 Register
DMC_EMR2	Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)
DMC_MR	Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3)
DMC_MR1	Shadow MR1 Register (DDR3)
DMC_MR2	Shadow MR2 Register (DDR3)
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register

Table 10-10: ADSP-SC58x DMC Register List (Continued)

Name	Description
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

Configuration Register

The `DMC_CFG` register selects SDRAM device specific parameters and selects the SDRAM interface width.

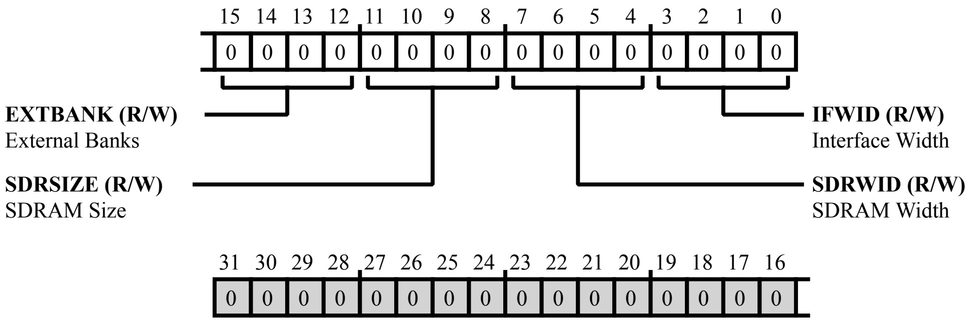

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EXTBANK (R/W)
External Banks
SDRSIZE (R/W)
SDRAM Size
IFWID (R/W)
Interface Width
SDRWID (R/W)
SDRAM Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-2: DMC_CFG Register Diagram

Table 10-11: DMC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EXTBANK	External Banks. The `DMC_CFG.EXTBANK` bits select the number of external banks connected to the DMC. Note that all values other than those shown are reserved.	
		0	1 External Bank
		1-15	Reserved
11:8 (R/W)	SDRSIZE	SDRAM Size. The `DMC_CFG.SDRSIZE` bits select the size of individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		0	64M Bit SDRAM (LPDDR Only)
		1	128M Bit SDRAM (LPDDR Only)
		2	256M Bit SDRAM
		3	512M Bit SDRAM
		4	1G Bit SDRAM
		5	2G Bit SDRAM
		6	4G Bit SDRAM
		7	8G Bit SDRAM

Table 10-11: DMC_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	SDRWID	SDRAM Width. The `DMC_CFG.SDRWID` bits select the width of the individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		0-1	Reserved
		2	16-Bit Wide SDRAM
		3-15	Reserved
3:0 (R/W)	IFWID	Interface Width. The `DMC_CFG.IFWID` bits select the width of the interface between the DMC and SDRAM. Note that all values other than those shown are reserved.	
		0-1	Reserved
		2	16-Bit Wide Interface. All other values are reserved. This field specifies the interface width between the controller and the SDRAM.
		3-15	Reserved

Controller to PHY Interface Register

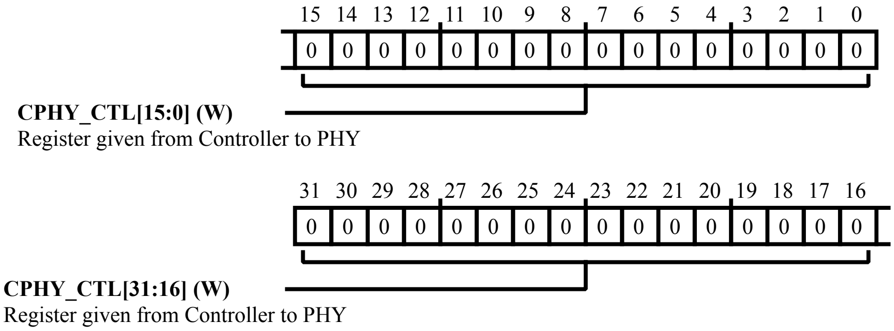

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPHY_CTL[15:0] (W)
Register given from Controller to PHY
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPHY_CTL[31:16] (W)
Register given from Controller to PHY

Figure 10-3: DMC_CPHY_CTL Register Diagram

Table 10-12: DMC_CPHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	CPHY_CTL	Register given from Controller to PHY.

Control Register

The `DMC_CTL` register controls DMC modes, DLL calibration, and DRAM initialization.

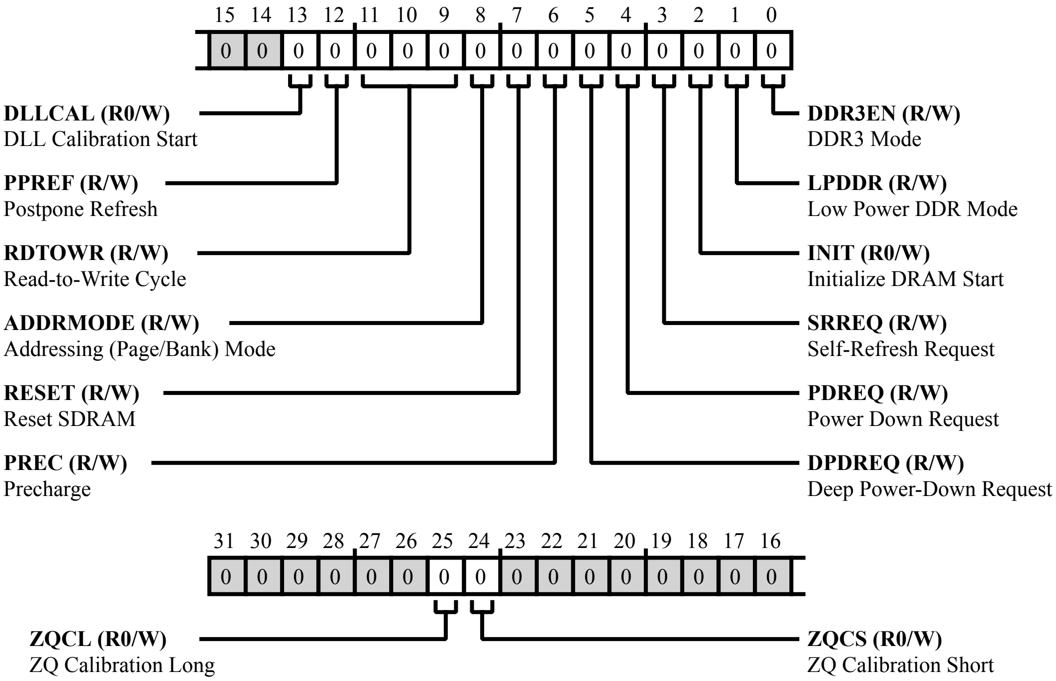

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DLLCAL (R0/W)
DLL Calibration Start
PPREF (R/W)
Postpone Refresh
RDTOWR (R/W)
Read-to-Write Cycle
ADDRMODE (R/W)
Addressing (Page/Bank) Mode
RESET (R/W)
Reset SDRAM
PREC (R/W)
Precharge
DDR3EN (R/W)
DDR3 Mode
LPDDR (R/W)
Low Power DDR Mode
INIT (R0/W)
Initialize DRAM Start
SRREQ (R/W)
Self-Refresh Request
PDREQ (R/W)
Power Down Request
DPDREQ (R/W)
Deep Power-Down Request
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ZQCL (R0/W)
ZQ Calibration Long
ZQCS (R0/W)
ZQ Calibration Short

Figure 10-4: DMC_CTL Register Diagram

Table 10-13: DMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R0/W)	ZQCL	ZQ Calibration Long. The `DMC_CTL.ZQCL` bit starts the ZQ calibration long sequence. Note that this bit always reads as 0.	
		0	No effect
		1	Triggers ZQ calibration long sequence
24 (R0/W)	ZQCS	ZQ Calibration Short. The `DMC_CTL.ZQCS` bit starts the ZQ calibration short sequence. Note that this bit always reads as 0.	
		0	No effect
		1	Triggers ZQ calibration short sequence

Table 10-13: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R0/W)	DLLCAL	DLL Calibration Start. The `DMC_CTL.DLLCAL` bit starts the PHY DLL calibration sequence. Note that this bit always reads as 0.	
		0	No effect
		1	Start PHY DLL calibration
12 (R/W)	PPREF	Postpone Refresh. The `DMC_CTL.PPREF` bit enables postponing the DMCs sending of auto-refresh commands. When enabled, the DMC accumulates refresh commands. The `DMC_EFFCTL.NUMREF` field selects the number of refresh commands that the DMC can accumulate. When disabled, the `DMC_TR1.TREF` field selects the interval for auto-refresh command distribution. A maximum of eight auto-refresh commands can be accumulated in DDR2 and DDR3 mode and a maximum of four auto-refresh commands in low power DDR mode.	
		0	Disable Postpone Refresh
		1	Enable Postpone Refresh
11:9 (R/W)	RDTOWR	Read-to-Write Cycle. The `DMC_CTL.RDTOWR` bits select the number of cycles that the DMC adds when a write operation follows a read operation. For proper operation, it should be programmed with the value of 010.	
		0	1 Cycle Added from JEDEC Spec Value
		1	2 Cycles Added from JEDEC Spec Value
		2	3 Cycles Added from JEDEC Spec Value
		3	4 Cycles Added from JEDEC Spec Value
		4	5 Cycles Added from JEDEC Spec Value
		5	6 Cycles Added from JEDEC Spec Value
		6	7 Cycles Added from JEDEC Spec Value
		7	8 Cycles Added from JEDEC Spec Value

Table 10-13: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	ADDRMODE	Addressing (Page/Bank) Mode. The `DMC_CTL.ADDRMODE` bit selects whether the DMC uses page or bank interleaving for addressing. When using page interleaving, the bank address bits follow the most significant column address bits. When using bank interleaving, the bank address bits follow the most significant row address bits.	
		0	Bank Interleaving
		1	Page Interleaving
7 (R/W)	RESET	Reset SDRAM. The `DMC_CTL.RESET` bit starts the reset sequence. Note that this bit always reads as 0.	
		0	No effect
		1	Starts reset sequence
6 (R/W)	PREC	Precharge. The `DMC_CTL.PREC` bit enables precharge, which closes DRAM rows immediately after access. When disabled, all accesses result in the respective DRAM rows remaining open, until the DMC needs to close them.	
		0	No Effect
		1	Enable Precharge
5 (R/W)	DPDREQ	Deep Power-Down Request. The `DMC_CTL.DPDREQ` bit enables deep power-down mode if low power DMC operation is enabled (`DMC_CTL.LPDDR` =1). When the processor does not require the data stored in SDRAM (assume reset state of SDRAM), the DMC may put the SDRAM in deep power-down mode. When the DMC is in deep power-down mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in deep power-down mode as along as this bit is 1.	
		0	Disable Deep Power-Down
		1	Enable Deep Power-Down
4 (R/W)	PDREQ	Power Down Request. The `DMC_CTL.PDREQ` bit enables power-down mode. When the DMC is in power-down mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in power-down mode as along as this bit is 1.	
		0	Disable Power-Down
		1	Enable Power-Down

Table 10-13: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SRREQ	Self-Refresh Request. The `DMC_CTL.SRREQ` bit enables self-refresh mode. When the DMC is in self-refresh mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in self-refresh mode as along as this bit is 1.	
		0	Disable Self-Refresh
		1	Enable Self-Refresh
2 (R0/W)	INIT	Initialize DRAM Start. The `DMC_CTL.INIT` bit starts the power up DRAM initialization sequence and DLL calibration sequence. Note that this bit always reads as 0.	
		0	No Effect
		1	Start DRAM Initialization
1 (R/W)	LPDDR	Low Power DDR Mode. The `DMC_CTL.LPDDR` bit selects whether the DMC operates in low power DDR mode or DDR2 mode.	
		0	DDR2 mode
		1	LPDDR mode
0 (R/W)	DDR3EN	DDR3 Mode. The `DMC_CTL.DDR3EN` bit selects whether the DMC operates in DDR3 mode or DDR2 mode.	
		0	Enable DDR2 mode
		1	Enable DDR3 mode

DLL Control Register

The `DMC_DLLCTL` register holds the programmable parameters associated with the DLLs within the DMC PHY.

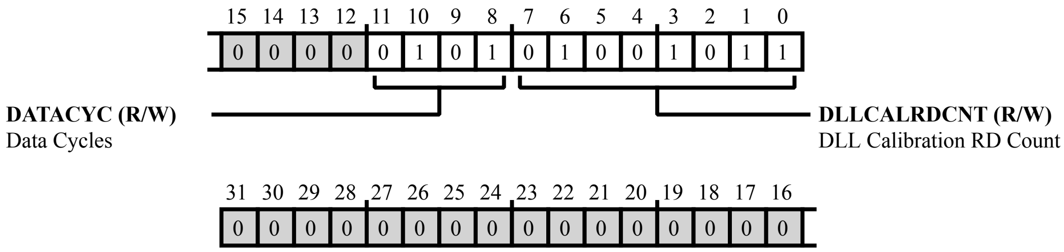

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1
DATACYC (R/W)
Data Cycles
DLLCALRDCNT (R/W)
DLL Calibration RD Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-5: DMC_DLLCTL Register Diagram

Table 10-14: DMC_DLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	DATACYC	Data Cycles. The `DMC_DLLCTL.DATACYC` bits select the latency after which the DMC reads data from the PHY. This field must be written with the value (9). All other values are reserved. Taking round trip delay into account, the DLL indicates whether a latency of 2 cycles is supported by means of status bits.
7:0 (R/W)	DLLCALRDCNT	DLL Calibration RD Count. The `DMC_DLLCTL.DLLCALRDCNT` field selects the number of read operations that the PHY uses for DLL calibration.

Data Calibration Address Register

The DMC_DT_CALIB_ADDR register provides the address used for the data calibration for read and write. During the DMC PHY DLL calibration, a particular set of locations in the DRAM is written and a series of reads are performed back to back to calibrate the PHY. The DMC PHY needs prior information about the data that would be read during the PHY DLL calibration. The controller performs one burst write operation to the address programmed in DMC_DT_CALIB_ADDR (0x0090).

Note: While the exact address chosen does not matter much during memory initialization, if calibration of the PHY is performed when the DRAM contains valid data, care needs to be taken to ensure that this address points to an unused address. Else, this operation will modify application data stored at the address selected.

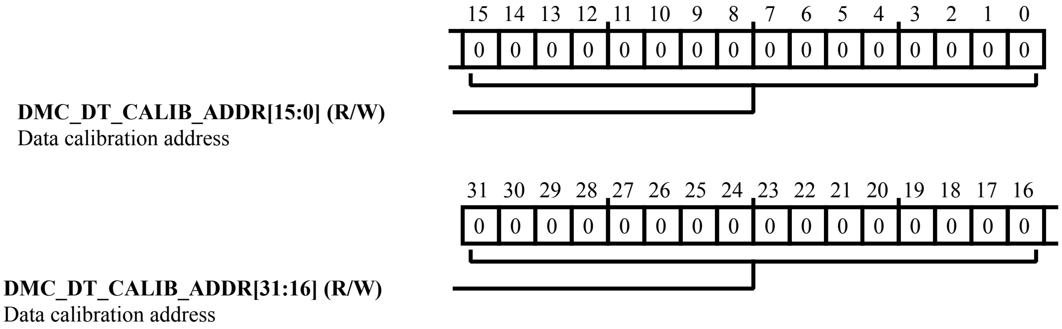

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_CALIB_ADDR[15:0] (R/W)
Data calibration address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_CALIB_ADDR[31:16] (R/W)
Data calibration address

Figure 10-6: DMC_DT_CALIB_ADDR Register Diagram

Table 10-15: DMC_DT_CALIB_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DMC_DT_CALIB_ADDR	Data calibration address. The `DMC_DT_CALIB_ADDR.DMC_DT_CALIB_ADDR` bit field contains the address to be programmed for the data calibration for read and write.

Data Calibration Data 0 Register

The `DMC_DT_DATA_CALIB_DATA0` register contains the first 32-bit data used for the write during the data calibration.

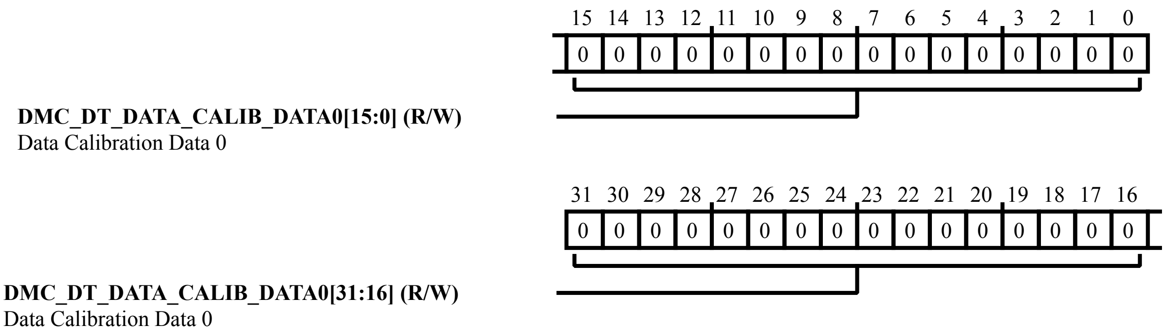

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_DATA_CALIB_DATA0[15:0] (R/W)
Data Calibration Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_DATA_CALIB_DATA0[31:16] (R/W)
Data Calibration Data 0

Figure 10-7: DMC_DT_DATA_CALIB_DATA0 Register Diagram

Table 10-16: DMC_DT_DATA_CALIB_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DMC_DT_DATA_CALIB_DATA0	Data Calibration Data 0. The `DMC_DT_DATA_CALIB_DATA0.DMC_DT_DATA_CALIB_DATA0` bit field contains the first 32 bit data used for the write during the data calibration.

Data Calibration Data 1 Register

The `DMC_DT_DATA_CALIB_DATA1` register contains the second 32-bit data used for the write during the data calibration.

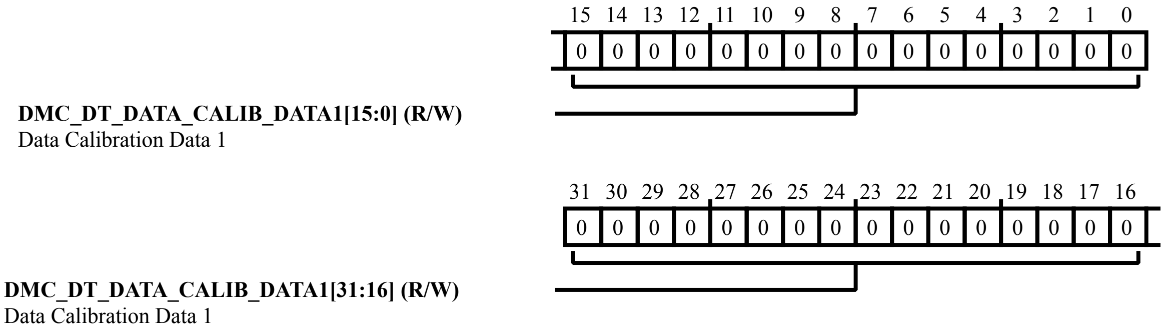

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_DATA_CALIB_DATA1[15:0] (R/W)
Data Calibration Data 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMC_DT_DATA_CALIB_DATA1[31:16] (R/W)
Data Calibration Data 1

Figure 10-8: DMC_DT_DATA_CALIB_DATA1 Register Diagram

Table 10-17: DMC_DT_DATA_CALIB_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DMC_DT_DATA_CALIB_DATA1	Data Calibration Data 1. The `DMC_DT_DATA_CALIB_DATA1.DMC_DT_DATA_CALIB_DATA1` bit field contains the second 32 bit data used for the write during the data calibration.

Efficiency Control Register

The `DMC_EFFCTL` register control DMC features that improve throughput efficiency. These include features such as auto-refresh management, precharge options, and write data options.

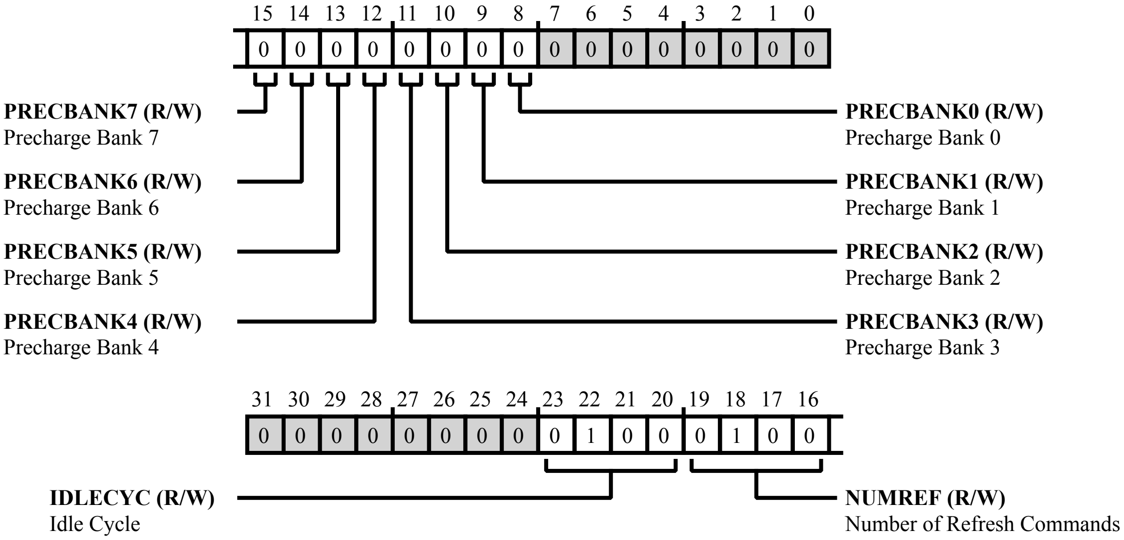

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PRECBANK7 (R/W)
Precharge Bank 7
PRECBANK6 (R/W)
Precharge Bank 6
PRECBANK5 (R/W)
Precharge Bank 5
PRECBANK4 (R/W)
Precharge Bank 4
PRECBANK0 (R/W)
Precharge Bank 0
PRECBANK1 (R/W)
Precharge Bank 1
PRECBANK2 (R/W)
Precharge Bank 2
PRECBANK3 (R/W)
Precharge Bank 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
IDLECYC (R/W)
Idle Cycle
NUMREF (R/W)
Number of Refresh Commands

Figure 10-9: DMC_EFFCTL Register Diagram

Table 10-18: DMC_EFFCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/W)	IDLECYC	Idle Cycle. The `DMC_EFFCTL.IDLECYC` bits select the number of cycles after which the DMC issues any accumulated auto-refresh commands if postpone refresh is enabled (`DMC_CTL.PPREF` =1). When `DMC_EFFCTL.IDLECYC` is set to 0, the DMC ignores the `DMC_CTL.PPREF` selection and does not accumulate/postpone periodic auto-refresh commands. Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles. Note 2: This value is ignored if `DMC_CTL.PPREF` is not set. Note 3: Setting this value to 0000 overrides the "postpone refresh" feature and does not accumulate/postpone periodic auto refreshes.	
		0-15	0 to 15 Idle Cycles to Postpone Refresh Commands

Table 10-18: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	NUMREF	Number of Refresh Commands. The `DMC_EFFCTL.NUMREF` bits select the number of auto-refresh commands that the DMC can accumulate if postpone refresh is enabled (`DMC_CTL.PPREF` =1). The number of auto-refresh commands that can accumulate depends on whether the DMC is in DDR2 or LPDDR mode as selected by the `DMC_CTL.LPDDR` bit. In LPDDR mode, the DMC can accumulate up to four auto-refresh commands. In DDR2 mode, the DMC may accumulate up to eight auto-refresh commands. Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles. Note 2: This value is ignored if `DMC_CTL.PPREF` is not set.	
		0	No Refresh Commands Accumulate
		1	1 Refresh Command May Accumulate
		2	2 Refresh Commands May Accumulate
		3	3 Refresh Commands May Accumulate
		4	4 Refresh Commands May Accumulate
		5	5 Refresh Commands May Accumulate
		6	6 Refresh Commands May Accumulate
		7	7 Refresh Commands May Accumulate
		8	8 Refresh Commands May Accumulate
15 (R/W)	PRECBANK7	Precharge Bank 7. The `DMC_EFFCTL.PRECBANK7` bit enables precharge (closes the page) of bank 7 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 7
		1	Enable Precharge Bank 7
14 (R/W)	PRECBANK6	Precharge Bank 6. The `DMC_EFFCTL.PRECBANK6` bit enables precharge (closes the page) of bank 6 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 6
		1	Enable Precharge Bank 6

Table 10-18: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PRECBANK5	Precharge Bank 5. The `DMC_EFFCTL.PRECBANK5` bit enables precharge (closes the page) of bank 5 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 5
		1	Enable Precharge Bank 5
12 (R/W)	PRECBANK4	Precharge Bank 4. The `DMC_EFFCTL.PRECBANK4` bit enables precharge (closes the page) of bank 4 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 4
		1	Enable Precharge Bank 4
11 (R/W)	PRECBANK3	Precharge Bank 3. The `DMC_EFFCTL.PRECBANK3` bit enables precharge (closes the page) of bank 3 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 3
		1	Enable Precharge Bank 3
10 (R/W)	PRECBANK2	Precharge Bank 2. The `DMC_EFFCTL.PRECBANK2` bit enables precharge (closes the page) of bank 2 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 2
		1	Enable Precharge Bank 2
9 (R/W)	PRECBANK1	Precharge Bank 1. The `DMC_EFFCTL.PRECBANK1` bit enables precharge (closes the page) of bank 1 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 1
		1	Enable Precharge Bank 1

Table 10-18: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PRECBANK0	Precharge Bank 0. The `DMC_EFFCTL.PRECBANK0` bit enables precharge (closes the page) of bank 0 after each transfer if the DMC precharge feature is enabled (`DMC_CTL.PREC` =1). Note: The (`DMC_CTL.PREC`) takes precedence over value in this register. If (`DMC_CTL.PREC` =1) then all banks are precharged.	
		0	Disable Precharge Bank 0
		1	Enable Precharge Bank 0

Shadow EMR1 DDR2 Register

The `DMC_EMR1` register in the DMC shadows the EMR1 register in the SDRAM when the DMC is in DDR2 mode (`DMC_CTL.LPDDR` =0). This register is used only when the DMC is operating in DDR2 mode.

If unmasked by the corresponding bit in the shadow mask register (`DMC_MSK.EMR1` =1), a write to `DMC_EMR1` triggers an extended "mode register set" command on the memory interface. If masked, a write to `DMC_EMR1` only updates the register in the DMC, not the register in the SDRAM.

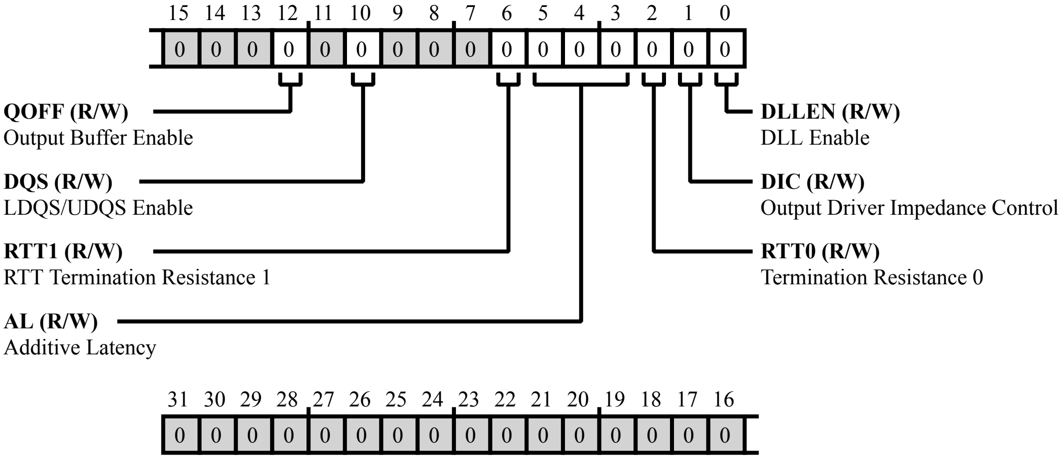

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
QOFF (R/W)
Output Buffer Enable
DQS (R/W)
LDQS/UDQS Enable
RTT1 (R/W)
RTT Termination Resistance 1
AL (R/W)
Additive Latency
DLLEN (R/W)
DLL Enable
DIC (R/W)
Output Driver Impedance Control
RTT0 (R/W)
Termination Resistance 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-10: DMC_EMR1 Register Diagram

Table 10-19: DMC_EMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	QOFF	Output Buffer Enable. The `DMC_EMR1.QOFF` bit enables the SDRAM output pins. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable
		1	Disable
10 (R/W)	DQS	LDQS/UDQS Enable. The `DMC_EMR1.DQS` bit enables the single ended operation of the `DMC_LDQS`/`DMC_LDQS` or `DMC_UDQS`/`DMC_UDQS` pin. For more information about this operation, see the data sheet for the SDRAM being used in your system. The DDR Controller enables differential signaling mode (EMR1) by default. Because the controller supports differential signaling mode only, setting this bit is not allowed.	
6 (R/W)	RTT1	RTT Termination Resistance 1. The `DMC_EMR1.RTT1` bit combines with the `DMC_EMR1.RTT0` bit to set the termination resistance. See the `DMC_EMR1.RTT0` bit description for more information.	

Table 10-19: DMC_EMR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:3 (R/W)	AL	Additive Latency. The `DMC_EMR1.AL` bits select a number of added latency time for CAS operations in terms of clock cycles (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	0 Clock Cycles Added
		1	1 Clock Cycle Added
		2	2 Clock Cycles Added
		3	3 Clock Cycles Added
		4	4 Clock Cycles Added
		5	5 Clock Cycles Added
2 (R/W)	RTT0	Termination Resistance 0. The `DMC_EMR1.RTT0` bit and the `DMC_EMR1.RTT1` bits select the SDRAM termination resistance. RTT1=0, RTT0=0: No ODT at memory device RTT1=0, RTT0=1: 75 Ohm ODT at memory device RTT1=1, RTT0=0: 150 Ohm ODT at memory device RTT1=1, RTT0=1: 50 Ohm ODT at memory device For more information about this operation, see the data sheet for the SDRAM being used in your system.	
1 (R/W)	DIC	Output Driver Impedance Control. The `DMC_EMR1.DIC` bit selects the drive strength mode for the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system. It must be kept at 0 if the SDRAM does not support this bit.	
		0	Full Strength
		1	Reduced Strength
0 (R/W)	DLLEN	DLL Enable. The `DMC_EMR1.DLLEN` bit enables the DLL in the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable DLL (Normal Operation)
		1	Disable DLL (Test/Debug Operation)

Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)

The `DMC_EMR2` register in the DMC shadows the EMR2 register in the SDRAM when the DMC is in DDR2 mode (`DMC_CTL.LPDDR` =0) and shadows the EMR register in the SDRAM when the DMC is in LPDDR mode (`DMC_CTL.LPDDR` =1). If unmasked by the corresponding bit in the shadow mask register (`DMC_MSK.EMR2` =1), a write to `DMC_EMR2` triggers an extended "mode register set" command on the memory interface. If masked, a write to `DMC_EMR2` only updates the register in the DMC, not the register in the SDRAM.

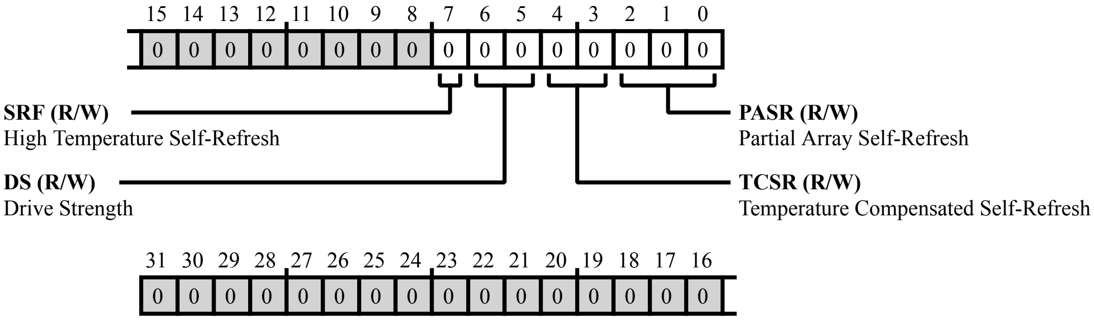

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SRF (R/W)
High Temperature Self-Refresh
DS (R/W)
Drive Strength
PASR (R/W)
Partial Array Self-Refresh
TCSR (R/W)
Temperature Compensated Self-Refresh
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-11: DMC_EMR2 Register Diagram

Table 10-20: DMC_EMR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SRF	High Temperature Self-Refresh. The `DMC_EMR2.SRF` bit enables the SDRAM's high temperature self-refresh rate feature when the DMC is in DDR2 mode. (This bit is reserved in LPDDR mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Disable
		1	Enable
6:5 (R/W)	DS	Drive Strength. The `DMC_EMR2.DS` bits select the drive strength value when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) Note that all values other than those shown are reserved. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		4	Octant Drive strength
		0	Full Drive Strength
		1	1/2 Drive Strength
		2	3/4 Drive Strength
		3	1/4 Drive Strength

Table 10-20: DMC_EMR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	TCSR	Temperature Compensated Self-Refresh. The `DMC_EMR2.TCSR` bits select the temperature for applying temperature compensated self-refresh when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	70 degree C (in LPDDR Mode)
		1	45 degree C
		2	15 degree C
		3	85 degree C
2:0 (R/W)	PASR	Partial Array Self-Refresh. The `DMC_EMR2.PASR` bits select the amount of memory to be refreshed during self-refresh. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Full Array for DDR2 , DDR3, and LPDDR Modes. Applies to both 4 and 8 bank devices.
		1	1/2 Array for DDR2 , DDR3, and LPDDR Modes. For 4 bank devices, BA[1:0] =00 and 01. For 8 bank devices, BA[2:0] = 000, 001, 010, 011
		2	1/4 Array for DDR2 , DDR3, and LPDDR Modes. For 4 bank devices, BA[1:0] = 00. For 8 bank devices, BA[2:0] = 000 and 001.
		3	1/8 Array for 8 DDR2 or DDR3 Banks Only. Reserved for LPDDR. For 4 bank devices, not defined. For 8 bank devices, BA[2:0] = 000.
		4	3/4 Array for DDR2 or DDR3. Reserved for LPDDR. For 4 bank devices, BA[1:0]=01, 10 and 11. For 8 bank devices, BA[2:0] = 010, 011, 100, 101, 110, and 111.
		5	1/2 Array for DDR2 or DDR3. 1/8 Array for LPDDR. For 4 bank devices, BA[1:0]=10 and 11. For 8 bank devices, BA[2:0] = 100, 101, 110, and 111.
		6	1/4 Array for DDR2 or DDR3. 1/16 Array for LPDDR. For 4 bank devices, BA[1:0]=11. For 8 bank devices, BA[2:0] =110 and 111.
		7	1/8 array (for DDR2 or DDR3 Banks only); Reserved (LPDDR)

Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3)

The `DMC_MR` register in the DMC shadows the MR register in the SDRAM when the DMC is in DDR2 mode or LPDDR mode (`DMC_CTL.LPDDR` =0 or =1 and `DMC_CTL.DDR3EN` =0) or DDR3 mode (`DMC_CTL.DDR3EN` =0 or =1 and `DMC_CTL.LPDDR` =0). If unmasked by the corresponding bit in the shadow mask register (`DMC_MSK.MR` =1), a write to `DMC_MR` triggers a "mode register set" command on the memory interface. If masked, a write to `DMC_MR` only updates the register in the DMC, not the register in the SDRAM.

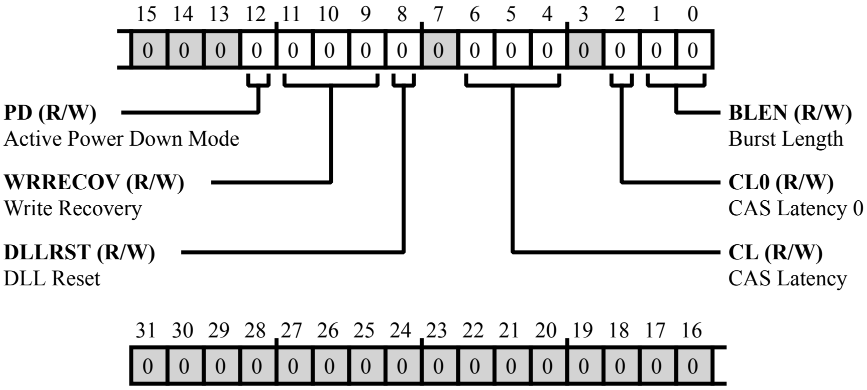

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PD (R/W)
Active Power Down Mode
WRRECOV (R/W)
Write Recovery
DLLRST (R/W)
DLL Reset
BLEN (R/W)
Burst Length
CL0 (R/W)
CAS Latency 0
CL (R/W)
CAS Latency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-12: DMC_MR Register Diagram

Table 10-21: DMC_MR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PD	Active Power Down Mode. The `DMC_MR.PD` bit selects the active power-down mode. Note that this parameter applies only for DDR2/DDR3 mode and is reserved for LPDDR mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.	
		0	Fast Exit (normal)
		1	Slow Exit (low power)
11:9 (R/W)	WRRECOV	Write Recovery. The `DMC_MR.WRRECOV` bit selects the write recovery time in terms of clock cycles (t_{CK}). Note that this parameter applies only for DDR2/DDR3 mode and is reserved for LPDDR mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.	
		0	16 clock cycles for DDR3 only
		1	2 Clock Cycles for DDR2 and 5 clock cycles for DDR3
		2	3 Clock Cycles for DDR2 and 6 clock cycles for DDR3
		3	4 Clock Cycles for DDR2 and 7 clock cycles for DDR3
		4	5 Clock Cycles for DDR2 and 8 clock cycles for DDR3

Table 10-21: DMC_MR Register Fields (Continued)

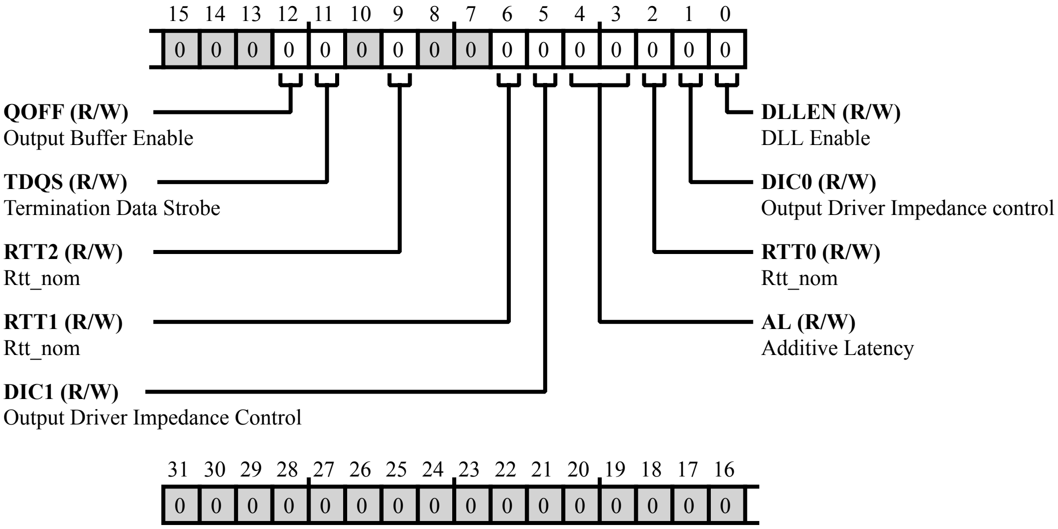
Bit No. (Access)	Bit Name	Description/Enumeration	
		5	6 Clock Cycles for DDR2 and 10 clock cycles for DDR3
		6	7 Clock Cycles for DDR2 and 12 clock cycles for DDR3
		7	8 Clock Cycles for DDR2 and 14 clock cycles for DDR3
8 (R/W)	DLLRST	DLL Reset. The `DMC_MR.DLLRST` bit initiates a DLL reset on the SDRAM. Note that this parameter applies only for DDR2/DDR3 mode and is reserved for LPDDR mode. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Normal Operation
		1	Reset DLL

Table 10-21: DMC_MR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:4 (R/W)	CL	CAS Latency. The `DMC_MR.CL` bit field selects latency from the assertion of a read/write signal to the SDRAM until the first valid data on the output from the SDRAM in terms of clock cycles. For more information about this operation, see the data sheet for the SDRAM being used in your system. The valid numbers for DDR2 and LPDDR are: 010 = Reserved 011 = 3 100 = 4 (DDR2 only) 101 = 5 (DDR2 only) 110 = 6 (DDR2 only) For DDR3 only bit [2] `DMC_MR.CL0` should be used along with bits [6:4]: 0010 = 5 0100 = 6 0110 = 7 1000 = 8 1010 = 9 1100 = 10 1110 = 11 0001 = 12 0011 = 13 0101 = 14 All other combinations are reserved.
2 (R/W)	CL0	CAS Latency 0. The `DMC_MR.CL0` bit is applicable for DDR3 only and is used in conjunction with the `DMC_MR.CL` bits.
1:0 (R/W)	BLEN	Burst Length. The `DMC_MR.BLEN` bits select burst length for transfers. For more information about this operation, see the data sheet for the SDRAM being used in your system. Note that values other than those shown are not supported. 0 | 8-Bit Burst Length - DDR3 only 2 | 4-Bit Burst Length -LPDDR/DDR2 only 3 | 8-Bit Burst Length - LPDDR/DDR2 only

Shadow MR1 Register (DDR3)

The DMC_MR1 register is a mirror of the DDR3 SDRAM Mode register 1. This register is used only when the DMC is operating in DDR3 mode. A write to this register triggers an extended "mode register 1 set" command on the memory interface provided the corresponding mask bit is set in the mask register. Else, only the mirror register is updated.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
QOFF (R/W)
Output Buffer Enable
TDQS (R/W)
Termination Data Strobe
RTT2 (R/W)
Rtt_nom
RTT1 (R/W)
Rtt_nom
DIC1 (R/W)
Output Driver Impedance Control
DLLEN (R/W)
DLL Enable
DIC0 (R/W)
Output Driver Impedance control
RTT0 (R/W)
Rtt_nom
AL (R/W)
Additive Latency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-13: DMC_MR1 Register Diagram

Table 10-22: DMC_MR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	QOFF	Output Buffer Enable. The `DMC_MR1.QOFF` bit enables the SDRAM output pins. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Output buffer enabled
		1	Output buffer disabled
11 (R/W)	TDQS	Termination Data Strobe. The `DMC_MR1.TDQS` bit provides additional termination resistance outputs that may be useful in some system configurations. The `DMC_MR1.TDQS` bit is not supported in x4 or x16 configurations. When enabled via the mode register, the same termination resistance function is applied to the TDQS/TDQS# pins that is applied to the DQS/ DQS# pins.	
		0	Enable
		1	Disable

Table 10-22: DMC_MR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	RTT2	Rtt_nom. The `DMC_MR1.RTT2` bit is used in conjunction with the `DMC_MR1.RTT0` and `DMC_MR1.RTT1` bits. (9 6 2) 0 0 0 Rtt_Nom disabled 0 0 1 RZQ/4 0 1 0 RZQ/2 0 1 1 RZQ/6 1 0 0 RZQ/12 (reserved if Rtt_Nom is used during writes) 1 0 1 RZQ/8 (reserved if Rtt_Nom is used during writes) 1 1 0 Reserved 1 1 1 Reserved
6 (R/W)	RTT1	Rtt_nom. The `DMC_MR1.RTT1` bit combines with the `DMC_MR1.RTT0` bit to set the termination resistance. See the `DMC_MR1.RTT2` and `DMC_MR1.RTT0` bit description for more information.
5 (R/W)	DIC1	Output Driver Impedance Control. The `DMC_MR1.DIC1` bit is used in conjunction with the `DMC_MR1.DIC0` bit. (5, 1) 0 0 RZQ/6 0 1 RZQ/7 1 0 Reserved 1 1 Reserved
4:3 (R/W)	AL	Additive Latency. The `DMC_MR1.AL` bits select a number of added latency time for CAS operations in terms of clock cycles (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system. 0 AL disabled 1 CL-1 2 CL-2 3 Reserved
2 (R/W)	RTT0	Rtt_nom. The `DMC_MR1.RTT0` bit combines with the `DMC_MR1.RTT1` and `DMC_MR1.RTT1` bits to set the termination resistance. See the `DMC_MR1.RTT1` and `DMC_MR1.RTT2` bit descriptions for more information.

Table 10-22: DMC_MR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	DIC0	Output Driver Impedance control. The `DMC_MR1.DIC0` bit is used with the `DMC_MR1.DIC1` bit.
0 (R/W)	DLLEN	DLL Enable. The `DMC_MR1.DLLEN` bit enables the DLL in the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system. 0 Enable 1 Disable

Shadow MR2 Register (DDR3)

The DMC_MR2 register mirrors DDR3 SDRAM device Mode register 2 when the controller is operating in DDR3 mode. A write to this register triggers an extended "mode register set" command on the memory interface provided the corresponding mask bit is set in the mask register. Else, only the mirror register is updated.

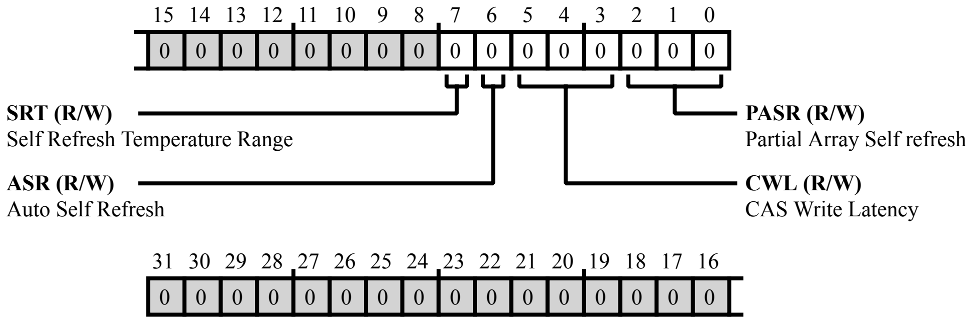

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SRT (R/W)
Self Refresh Temperature Range
ASR (R/W)
Auto Self Refresh
PASR (R/W)
Partial Array Self refresh
CWL (R/W)
CAS Write Latency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-14: DMC_MR2 Register Diagram

Table 10-23: DMC_MR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SRT	Self Refresh Temperature Range. The DMC_MR2.SRT bit enables high temperature self-refresh rate.	
		0	Disable
		1	Enable
6 (R/W)	ASR	Auto Self Refresh.	
		0	Manual SR Reference (SRT)
		1	ASR enable (Optional)
5:3 (R/W)	CWL	CAS Write Latency.	
		0	5 clock cycles
		1	6 clock cycles
		2	7 clock cycles
		3	8 clock cycles
		4	9 clock cycles
		5	10 clock cycles
		6	11 clock cycles
		7	12 clock cycles

Table 10-23: DMC_MR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	PASR	Partial Array Self refresh. The `DMC_MR2.PASR` bits select the amount of memory to be refreshed during self refresh. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	4 banks: full array, 8 banks: full array
		1	4 banks: Half Array (BA[1:0]=00&01), 8 banks: Half Array (BA[2:0] = 000, 001, 010, &011)
		2	4 banks: Quarter Array (BA[1:0]=00), 8 banks: Quarter Array (BA[2:0] = 000&001)
		3	4 banks: not defined, 8 banks: 1/8th array (BA[2:0] = 000)
		4	4 banks: 3/4 Array (BA[1:0]=01, 10&11), 8 banks: 3/4 Array (BA[2:0] = 010, 011, 100, 101, 110, &111)
		5	4 banks: Half Array (BA[1:0]=10&11), 8 banks: Half Array (BA[2:0] = 100, 101, 110, &111)
		6	4 banks: Quarter Array (BA[1:0]=11), 8 banks: Quarter Array (BA[2:0] =110 & 111)
		7	4 banks: not defined, 8 banks: 1/8th array (BA[2:0] = 111)

Mask (Mode Register Shadow) Register

The `DMC_MSK` register permits masking (disabling) writes to the MR and EMRn registers in the SDRAM. When masked, writes to these registers go instead to shadow copies of these registers (`DMC_MR`, `DMC_EMR1`, `DMC_EMR2`), which are maintained within the DMC. When a shadow register's corresponding bit is unmasked (enabled), the DMC generates the MRS or EMRS command to transfer the contents of the shadow register (in the DMC) to the actual register (in the SDRAM).

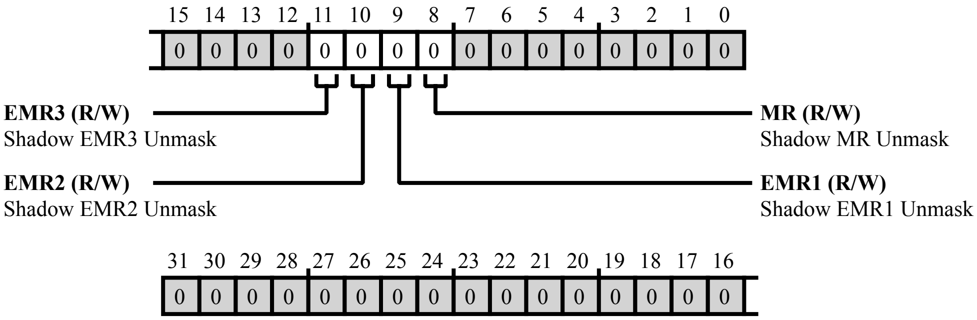

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EMR3 (R/W)
Shadow EMR3 Unmask
EMR2 (R/W)
Shadow EMR2 Unmask
MR (R/W)
Shadow MR Unmask
EMR1 (R/W)
Shadow EMR1 Unmask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-15: DMC_MSK Register Diagram

Table 10-24: DMC_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EMR3	Shadow EMR3 Unmask. The `DMC_MSK.EMR3` bit masks or unmasks writes to the EMR3 register (in DDR2) in the SDRAM. When masked, writes to this register instead go to the EMR3 register. When unmasked, the DMC writes the EMR3 value to the EMR3 register (in DDR2) in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (`DMC_CTL.LPDDR` =1).	
		0	Mask (Disable) Write to EMR3
		1	Unmask (Enable) Write to EMR3
10 (R/W)	EMR2	Shadow EMR2 Unmask. The `DMC_MSK.EMR2` bit masks or unmasks writes to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. When masked, writes to this register instead go to the `DMC_EMR2` register. When unmasked, the DMC writes the `DMC_EMR2` value to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. After completing the write, the DMC clears this bit.	
		0	Mask (Disable) Write to EMR2
		1	Unmask (Enable) Write to EMR2

Table 10-24: DMC_MSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	EMR1	Shadow EMR1 Unmask. The `DMC_MSK.EMR1` bit masks or unmasks writes to the EMR1 register in the SDRAM. When masked, writes to this register instead go to the `DMC_EMR1` register. When unmasked, the DMC writes the `DMC_EMR1` value to the EMR1 register in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (`DMC_CTL.LPDDR` =1).
		0 Mask (Disable) Write to EMR1
		1 Unmask (Enable) Write to EMR1
8 (R/W)	MR	Shadow MR Unmask. The `DMC_MSK.MR` bit masks or unmasks writes to the MR register in the SDRAM. When masked, writes to this register instead go to the `DMC_MR` register. When unmasked, the DMC writes the `DMC_MR` value to the MR register in the SDRAM. After completing the write, the DMC clears this bit.
		0 Mask (Disable) Write to MR
		1 Unmask (Enable) Write to MR

Priority ID Register 1

The `DMC_PRIO` register allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the `DMC_PRIOMSK` register to obtain a single SCB ID or a range of IDs that get elevated priority.

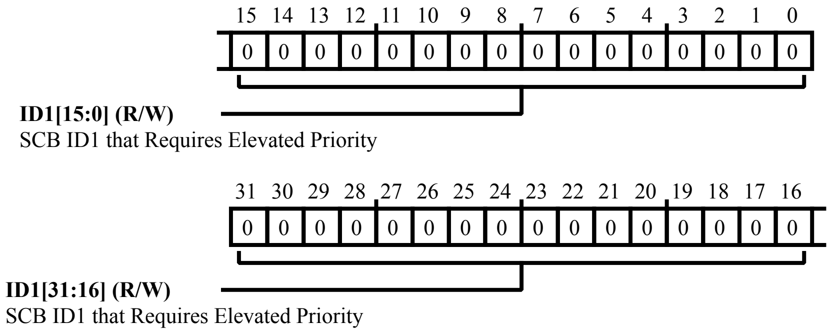

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID1[15:0] (R/W)
SCB ID1 that Requires Elevated Priority
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID1[31:16] (R/W)
SCB ID1 that Requires Elevated Priority

Figure 10-16: DMC_PRIO Register Diagram

Table 10-25: DMC_PRIO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1	SCB ID1 that Requires Elevated Priority.

Priority ID Register 2

The `DMC_PRIO2` register is another register which allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the `DMC_PRIOMSK2` register to obtain a single SCB ID or a range of IDs that get elevated priority.

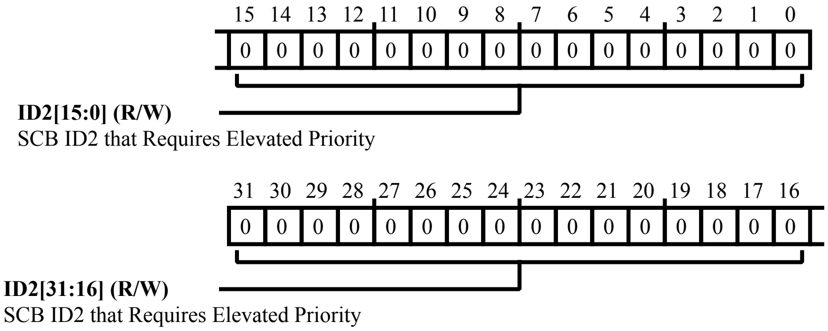

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID2[15:0] (R/W)
SCB ID2 that Requires Elevated Priority
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID2[31:16] (R/W)
SCB ID2 that Requires Elevated Priority

Figure 10-17: DMC_PRIO2 Register Diagram

Table 10-26: DMC_PRIO2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2	SCB ID2 that Requires Elevated Priority.

Priority ID Mask Register 1

The `DMC_PRIOMSK` register masks the respective ID bits in the `DMC_PRIOMSK` register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

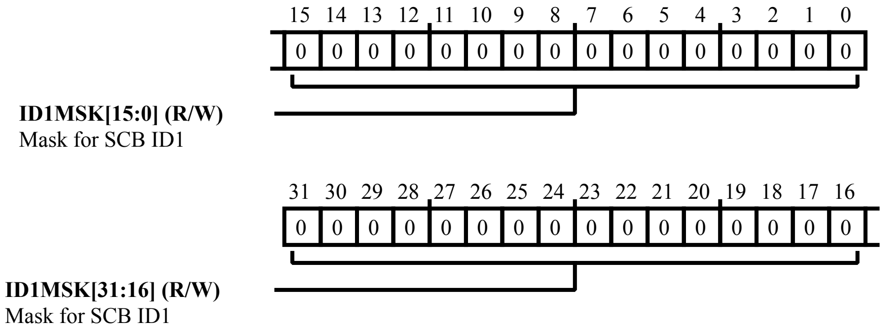

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID1MSK[15:0] (R/W)
Mask for SCB ID1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID1MSK[31:16] (R/W)
Mask for SCB ID1

Figure 10-18: DMC_PRIOMSK Register Diagram

Table 10-27: DMC_PRIOMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1MSK	Mask for SCB ID1.

Priority ID Mask Register 2

The `DMC_PRIOMSK2` register bits mask the respective ID bits in the `DMC_PRIO2` register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

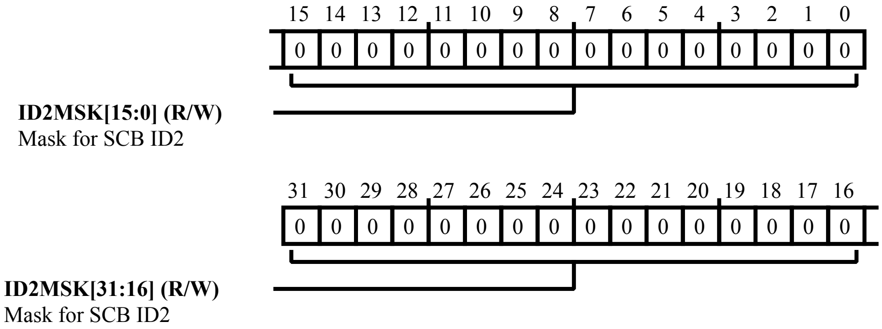

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID2MSK[15:0] (R/W)
Mask for SCB ID2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID2MSK[31:16] (R/W)
Mask for SCB ID2

Figure 10-19: DMC_PRIOMSK2 Register Diagram

Table 10-28: DMC_PRIOMSK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2MSK	Mask for SCB ID2.

DMC Read Data Buffer ID Register 1

The `DMC_RDDATABUFID1` register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the `DMC_RDDATABUFMSK1` register to obtain a single SCB ID or a range of IDs that get elevated priority.

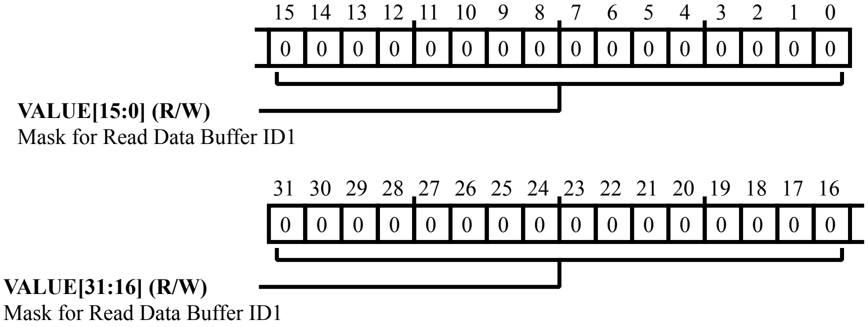

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Mask for Read Data Buffer ID1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Mask for Read Data Buffer ID1

Figure 10-20: DMC_RDDATABUFID1 Register Diagram

Table 10-29: DMC_RDDATABUFID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer ID Register 2

The `DMC_RDDATABUFID2` register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the `DMC_RDDATABUFMSK2` register to obtain a single SCB ID or a range of IDs that get elevated priority.

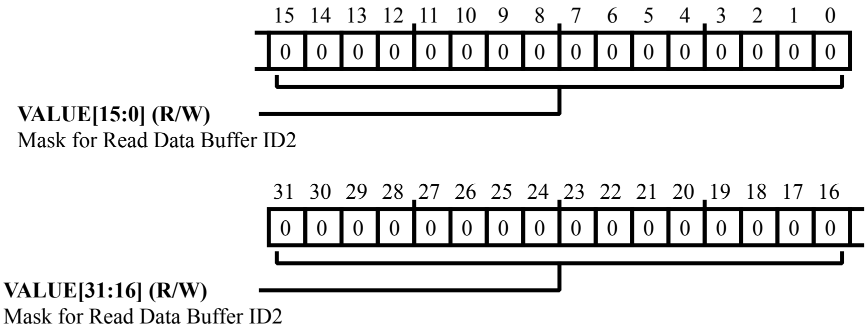

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Mask for Read Data Buffer ID2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Mask for Read Data Buffer ID2

Figure 10-21: DMC_RDDATABUFID2 Register Diagram

Table 10-30: DMC_RDDATABUFID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

DMC Read Data Buffer Mask Register 1

The `DMC_RDDATABUFMSK1` register bits mask the respective ID bits in the DMC Priority Mask ID register.

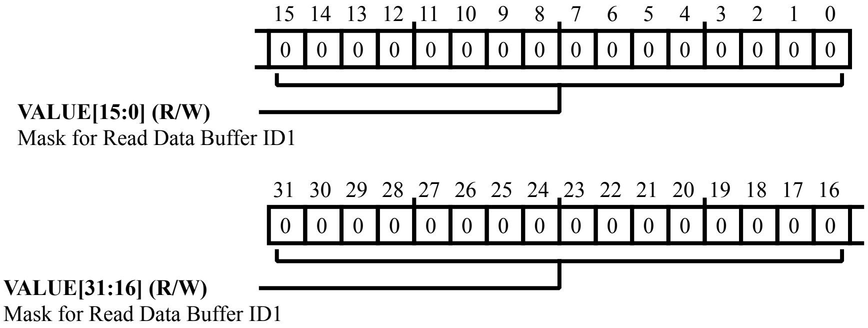

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Mask for Read Data Buffer ID1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Mask for Read Data Buffer ID1

Figure 10-22: DMC_RDDATABUFMSK1 Register Diagram

Table 10-31: DMC_RDDATABUFMSK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer Mask Register 2

The `DMC_RDDATABUFMSK2` register bits mask the respective ID bits in the DMC Priority Mask ID register.

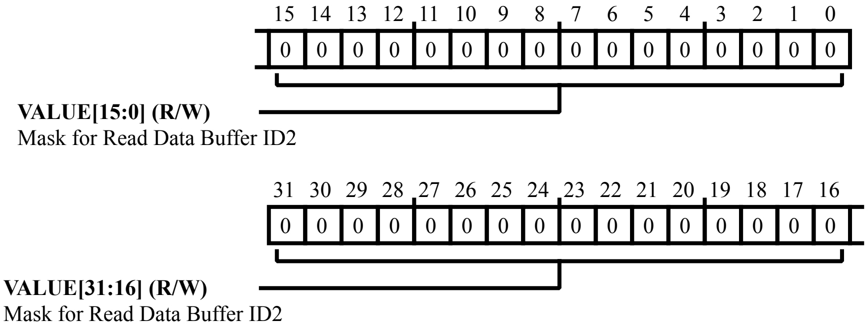

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Mask for Read Data Buffer ID2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Mask for Read Data Buffer ID2

Figure 10-23: DMC_RDDATABUFMSK2 Register Diagram

Table 10-32: DMC_RDDATABUFMSK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

Status Register

The DMC_STAT register indicates status for modes selected with the DMC_CTL register and indicates status DMC operations.

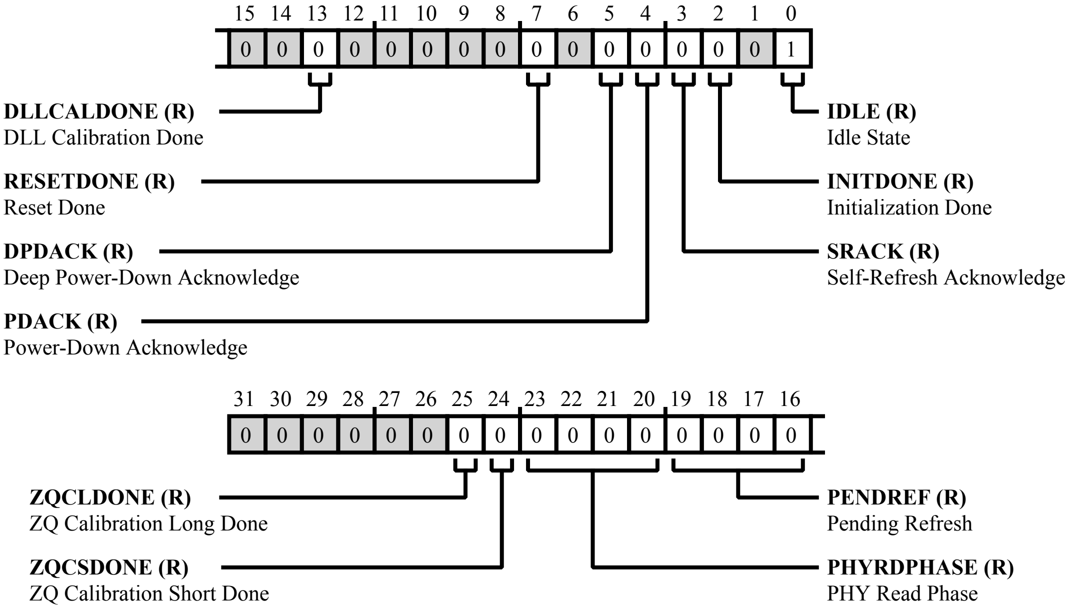

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
DLLCALDONE (R)
DLL Calibration Done
RESETDONE (R)
Reset Done
DPDACK (R)
Deep Power-Down Acknowledge
PDACK (R)
Power-Down Acknowledge
IDLE (R)
Idle State
INITDONE (R)
Initialization Done
SRACK (R)
Self-Refresh Acknowledge
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ZQCLDONE (R)
ZQ Calibration Long Done
ZQCSDONE (R)
ZQ Calibration Short Done
PENDREF (R)
Pending Refresh
PHYRDPHASE (R)
PHY Read Phase

Figure 10-24: DMC_STAT Register Diagram

Table 10-33: DMC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/NW)	ZQCLDONE	ZQ Calibration Long Done. The DMC_STAT.ZQCLDONE bit checks if the ZQ calibration long sub routine is done.	
		0	ZQ Calibration long is ongoing
		1	ZQ Calibration long is done
24 (R/NW)	ZQCSDONE	ZQ Calibration Short Done. The DMC_STAT.ZQCSDONE bit checks if the ZQ calibration short sub routine is done.	
		0	ZQ Calibration short is ongoing
		1	ZQ Calibration Short is done
23:20 (R/NW)	PHYRDPHASE	PHY Read Phase. The DMC_STAT.PHYRDPHASE bits indicate the latency after which the DMC may read from the PHY. Taking round trip delay into account, the DLL indicates the exact number of clock cycles after which the controller needs to read data. Values other than those shown are reserved.	

Table 10-33: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	2 Clock Cycles Latency
		3	3 Clock Cycles Latency
		4	4 Clock Cycles Latency
		5	5 Clock Cycles Latency
		6	6 Clock Cycles Latency
		7	7Clock Cycles Latency
19:16 (R/NW)	PENDREF	Pending Refresh. The `DMC_STAT.PENDREF` bits indicate the number of pending auto-refresh commands whose value can be from "0000" to "0111". When the DMC is in low power DDR mode (`DMC_CTL.LPDDR` =1), the maximum value for `DMC_STAT.PENDREF` is 3.	
13 (R/NW)	DLLCALDONE	DLL Calibration Done. The `DMC_STAT.DLLCALDONE` indicates that the PHY DLL calibration sequence is complete.	
		0	No Status
		1	Completed PHY DLL Calibration
7 (R/NW)	RESETDONE	Reset Done. The `DMC_STAT.RESETDONE` bit indicates that the reset sequence is complete.	
		0	SDRAM Reset is ongoing
		1	SDRAM Reset is done
5 (R/NW)	DPDACK	Deep Power-Down Acknowledge. The `DMC_STAT.DPDACK` bit indicates that deep power-down mode is active. Note that this status is available in low power DDR mode (`DMC_CTL.LPDDR` =1) only.	
		0	Not in Deep Power-Down Mode
		1	Deep Power-Down Mode Active
4 (R/NW)	PDACK	Power-Down Acknowledge. The `DMC_STAT.PDACK` bit indicates that power-down mode is active.	
		0	Not in Power-Down Mode
		1	Power-Down Mode Active
3 (R/NW)	SRACK	Self-Refresh Acknowledge. The `DMC_STAT.SRACK` bit indicates that self-refresh mode is active.	
		0	Not in Self-Refresh Mode
		1	Self-Refresh Mode Active

Table 10-33: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	INITDONE	Initialization Done. The `DMC_STAT.INITDONE` bit indicates that the initialization sequence is complete.	
		0	No Status
		1	Initialize Done
0 (R/NW)	IDLE	Idle State. The `DMC_STAT.IDLE` bit indicates whether the DMC is idle or busy.	
		0	Busy
		1	Idle

Timing 0 Register

The `DMC_TR0` register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in `DMC_TR0` are in increments of clock cycle time (t_{CK}).

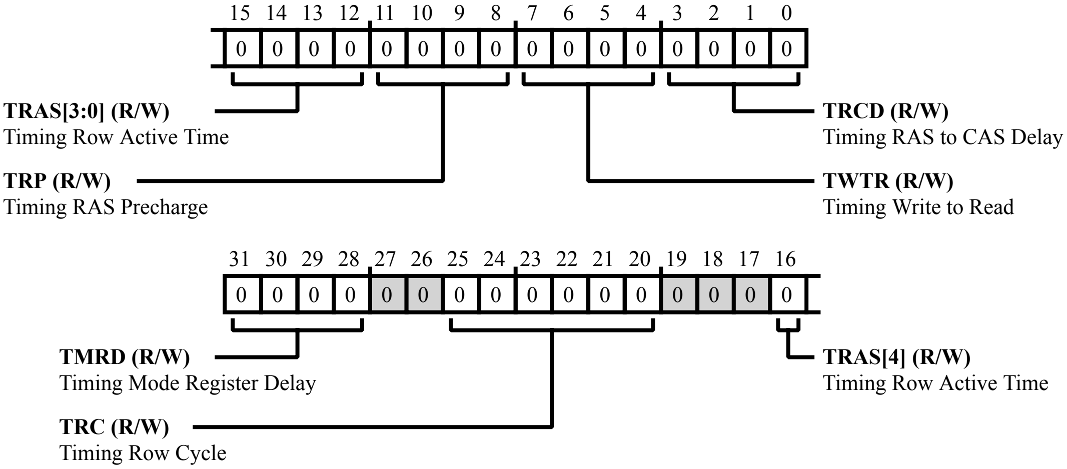

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRAS[3:0] (R/W)
Timing Row Active Time
TRP (R/W)
Timing RAS Precharge
TRCD (R/W)
Timing RAS to CAS Delay
TWTR (R/W)
Timing Write to Read
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRD (R/W)
Timing Mode Register Delay
TRC (R/W)
Timing Row Cycle
TRAS[4] (R/W)
Timing Row Active Time

Figure 10-25: DMC_TR0 Register Diagram

Table 10-34: DMC_TR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/W)	TMRD	Timing Mode Register Delay. The `DMC_TR0.TMRD` field selects the set-to-active timing parameter (t_{MRD}), which is the number of clock cycles that occur after the mode registers in the SDRAM are set and before the next command is issued.
25:20 (R/W)	TRC	Timing Row Cycle. The `DMC_TR0.TRC` field selects the active-to-active time (t_{RC}), which is the minimum number of clock cycles that occur from an active command to the next active command in the same bank.
16:12 (R/W)	TRAS	Timing Row Active Time. The `DMC_TR0.TRAS` field selects the active-to-precharge time (t_{RAS}), which is the number of clock cycles that occur from an active command until a precharge command is allowed.
11:8 (R/W)	TRP	Timing RAS Precharge. The `DMC_TR0.TRP` field selects the precharge-to-active time (t_{RP}), which is the number of clock cycles that occur while the SDRAM recovers from a precharge command and becomes ready to accept the next active command.

Table 10-34: DMC_TR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	TWTR	Timing Write to Read. The `DMC_TR0.TWTR` field selects the write-to-read delay time (t_{WTR}), which is the number of clock cycles that occur from the last write data to the next read command.
3:0 (R/W)	TRCD	Timing RAS to CAS Delay. The `DMC_TR0.TRCD` field selects the RAS to CAS delay time (t_{RCD}), which is the number of clock cycles that occur from an active command to a read/write assertion.

Timing 1 Register

The DMC_TR1 register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in DMC_TR1 are in increments of clock cycle time (t_{CK}).

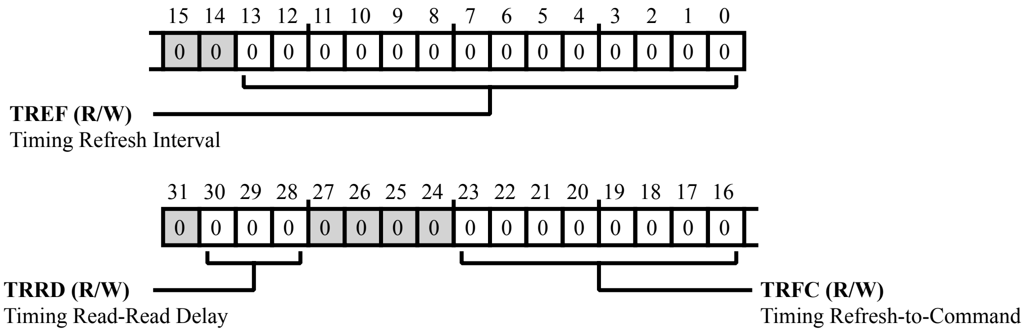

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TREF (R/W)
Timing Refresh Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRRD (R/W)
Timing Read-Read Delay
TRFC (R/W)
Timing Refresh-to-Command

Figure 10-26: DMC_TR1 Register Diagram

Table 10-35: DMC_TR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:28 (R/W)	TRRD	Timing Read-Read Delay. The `DMC_TR1.TRRD` field selects the active-to-active time (t_{RRD}), which is the minimum number of clock cycles occurring from a bank x active command to a bank y active command.
23:16 (R/W)	TRFC	Timing Refresh-to-Command. The `DMC_TR1.TRFC` field selects the refresh-to-active command delay (t_{RFC}), which is the number of clock cycles required for the SDRAM to recover from a refresh signal to be ready to take the next command. It is also the number of clock cycles needed for the SDRAM to recover from executing one active command and ready to accept the next active command.
13:0 (R/W)	TREF	Timing Refresh Interval. The `DMC_TR1.TREF` field selects the refresh interval time (t_{REF}), which is the number of clock cycles occurring from one refresh command to the next refresh command. The actual timing of issuing a precharge command may be delayed by if the SDRAM is processing a normal access. However, the delay is not accumulative so there is no need to shorten the refresh interval to account for the memory access time. The non-accumulative refresh delay typically increases memory bandwidth by a few percentage points.

Timing 2 Register

The `DMC_TR2` register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and before initializing the SDRAM.

Note that all values for bit fields in `DMC_TR2` are in increments of clock cycle time (t_{CK}).

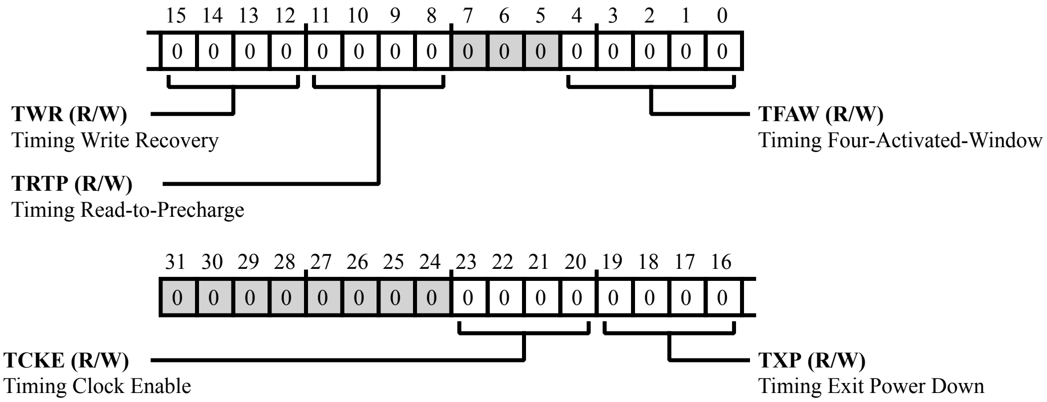

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TWR (R/W)
Timing Write Recovery
TRTP (R/W)
Timing Read-to-Precharge
TFAW (R/W)
Timing Four-Activated-Window
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TCKE (R/W)
Timing Clock Enable
TXP (R/W)
Timing Exit Power Down

Figure 10-27: DMC_TR2 Register Diagram

Table 10-36: DMC_TR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:20 (R/W)	TCKE	Timing Clock Enable. The `DMC_TR2.TCKE` field selects the CKE minimum pulsewidth (t_{CKE}).
19:16 (R/W)	TXP	Timing Exit Power Down. The `DMC_TR2.TXP` field selects the exit power down to next valid command time (t_{XP}).
15:12 (R/W)	TWR	Timing Write Recovery. The `DMC_TR2.TWR` field selects the write recovery time (t_{WR}). Note that this parameter applies to LPDDR only.
11:8 (R/W)	TRTP	Timing Read-to-Precharge. The `DMC_TR2.TRTP` bit field selects the internal read to precharge time (t_{RTP}) for DDR2/DDR3 modes. If the resulting value in DDR2/DDR3 mode is less than 2, program the bit field to 2. This value (2) is the minimum t_{RTP} time for DDR2/DDR3 mode. For LPDDR mode program this bit field to zero.
4:0 (R/W)	TFAW	Timing Four-Activated-Window. The `DMC_TR2.TFAW` field selects the four-banks-activated window time (t_{FAW}). No more than four SDRAM banks should be activated within this window.

ADSP-SC58x DMC Register Descriptions

DMCPHY (DMC) contains the following registers.

Table 10-37: ADSP-SC58x DMC Register List

Name	Description
`DMC_CAL_PADCTL0`	Calibration PAD Control 0 Register
`DMC_CAL_PADCTL2`	Calibration PAD Control 2 Register
`DMC_PHY_CTL0`	PHY Control 0 Register
`DMC_PHY_CTL1`	PHY Control 1 Register
`DMC_PHY_CTL2`	PHY Control 2 Register
`DMC_PHY_CTL3`	PHY Control 3 Register
`DMC_PHY_CTL4`	PHY Control 4 Register

Calibration PAD Control 0 Register

The `DMC_CAL_PADCTL0` register sets the pad calibration controls.

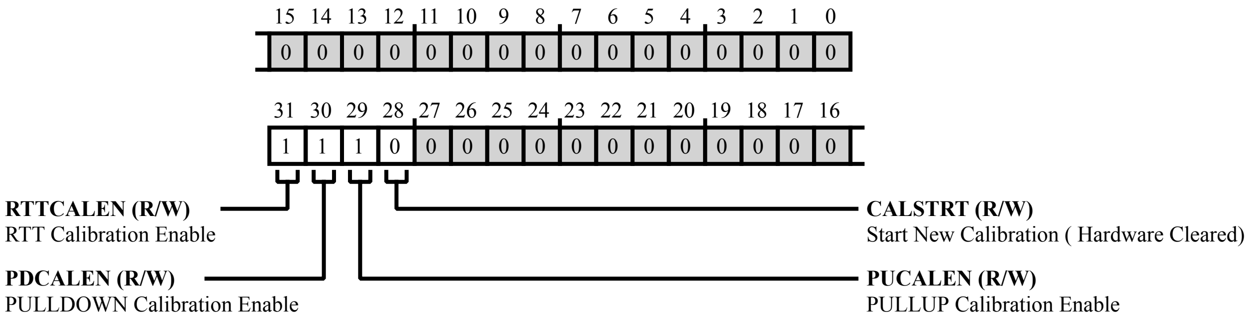

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
RTTCALEN (R/W)
RTT Calibration Enable
PDCALEN (R/W)
PULLDOWN Calibration Enable
CALSTRT (R/W)
Start New Calibration (Hardware Cleared)
PUCALEN (R/W)
PULLUP Calibration Enable

Figure 10-28: DMC_CAL_PADCTL0 Register Diagram

Table 10-38: DMC_CAL_PADCTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	RTTCALEN	RTT Calibration Enable. The `DMC_CAL_PADCTL0.RTTCALEN` bit is set to 1 at reset. Programming this bit to 0 is not allowed.
30 (R/W)	PDCALEN	PULLDOWN Calibration Enable. The `DMC_CAL_PADCTL0.PDCALEN` bit is set to 1 at reset. Programming this bit to 0 is not allowed.
29 (R/W)	PUCALEN	PULLUP Calibration Enable. The `DMC_CAL_PADCTL0.PUCALEN` bit is set to 1 at reset. Programming this bit to 0 is not allowed.
28 (R/W)	CALSTRT	Start New Calibration (Hardware Cleared).

Calibration PAD Control 2 Register

The `DMC_CAL_PADCTL2` register sets the pad calibration controls. The DMC pads can be auto-calibrated to the required driver impedance and the On Die Termination (ODT) value using the corresponding bits in this register. These values are translated by the auto calibration logic into a corresponding drive strength control inside the PHY and then routed to the PADS. Auto-calibration starts as soon as the `DMC_CAL_PADCTL0.CALSTRT` bit is programmed. The DCLK needs to be set at the required frequency before setting the `DMC_CAL_PADCTL0.CALSTRT` bit.

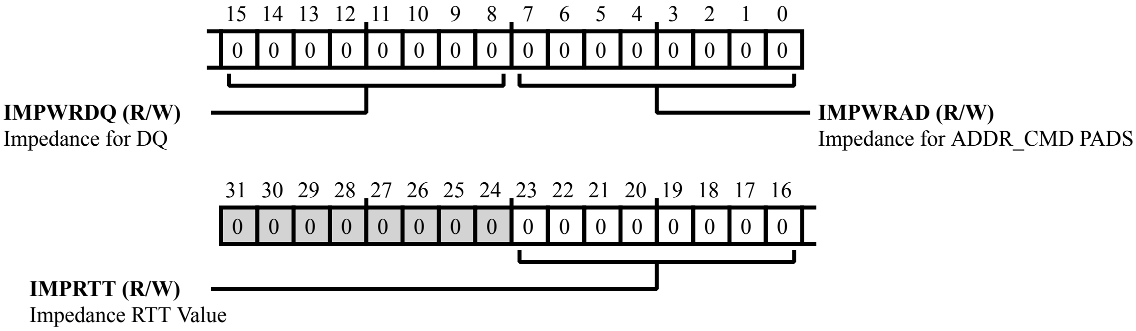

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IMPWRDQ (R/W)
Impedance for DQ
IMPWRAD (R/W)
Impedance for ADDR_CMD PADS
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IMPRTT (R/W)
Impedance RTT Value

Figure 10-29: DMC_CAL_PADCTL2 Register Diagram

Table 10-39: DMC_CAL_PADCTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	IMPRTT	Impedance RTT Value. Writing to the `DMC_CAL_PADCTL2.IMPRTT` bit field sets the required initialization sequence to program the termination impedance for the data PADS and the DQS PADS.
15:8 (R/W)	IMPWRDQ	Impedance for DQ. The `DMC_CAL_PADCTL2.IMPWRDQ` bit field sets the drive impedance for DQ DQS CLK and DM pads. Data pads (`DDR_DQ[NN]`), DQS pads (`DDR_LDQS`, /`DDR_LDQS`, `DDR_UDQS`, /`DDR_UDQS`), Clock pads (`DDR_CK`, /`DDR_CK`), DM pads (`DDR_UDM`, `DDR_LDM`)
7:0 (R/W)	IMPWRAD	Impedance for ADDR_CMD PADS. The `DMC_CAL_PADCTL2.IMPWRAD` bit field sets the desired drive for address pads (`DDR_A[NN]`), Command pads (`DDR_RAS`, `DDR_CAS`, `DDR_CKE`, `DDR_WE`, `DDR_CS[N]`, `DDR_ODT`).

PHY Control 0 Register

The `DMC_PHY_CTL0` register controls programmable PHY features.

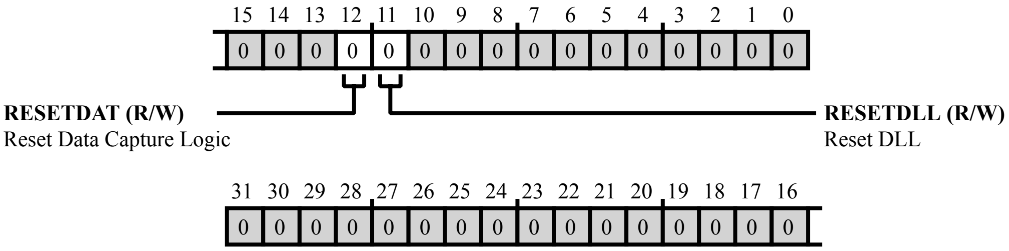

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RESETDAT (R/W)
Reset Data Capture Logic
RESETDLL (R/W)
Reset DLL
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

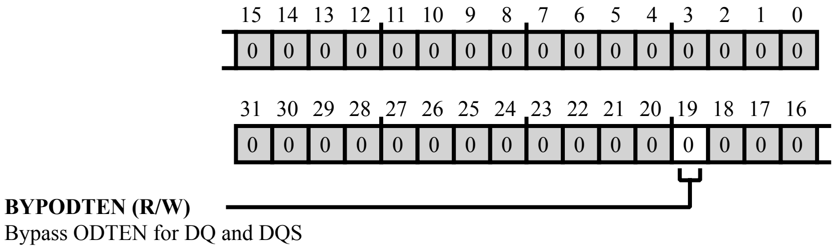
Figure 10-30: DMC_PHY_CTL0 Register Diagram

Table 10-40: DMC_PHY_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	RESETDAT	Reset Data Capture Logic. The `DMC_PHY_CTL0.RESETDAT` bit resets the data capture logic only, including P and N buffers. If Quickboot is used, this bit does not have any effect. The `DMC_PHY_CTL0.RESETDAT` bit is reset by the hardware. A read of this bit returns zero.
11 (R/W)	RESETDLL	Reset DLL. The `DMC_PHY_CTL0.RESETDLL` bit resets DLL control logic only, including the 90 degree DQS shifters. If Quickboot is used, this bit does not have any effect. The `DMC_PHY_CTL0.RESETDLL` bit is reset by the hardware a read of this bit returns zero.

PHY Control 1 Register

The `DMC_PHY_CTL1` register controls programmable PHY features.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYPODTEN (R/W)
Bypass ODTEN for DQ and DQS

Figure 10-31: DMC_PHY_CTL1 Register Diagram

Table 10-41: DMC_PHY_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	BYPODTEN	Bypass ODTEN for DQ and DQS.	
		0	Reserved
		1	Reserved

PHY Control 2 Register

The `DMC_PHY_CTL2` register controls programmable PHY features. Program this register as per the programming guidelines for proper operation of the DMC.

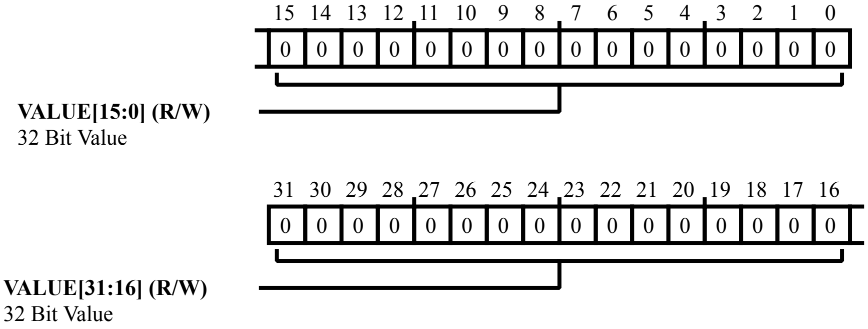

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
32 Bit Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
32 Bit Value

Figure 10-32: DMC_PHY_CTL2 Register Diagram

Table 10-42: DMC_PHY_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Control 3 Register

The DMC_PHY_CTL3 register controls programmable PHY features. Program this register as per the programming guidelines for proper operation of DMC.

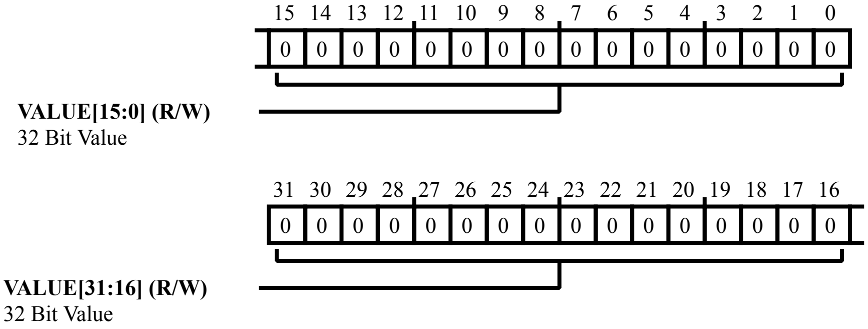

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
32 Bit Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
32 Bit Value

Figure 10-33: DMC_PHY_CTL3 Register Diagram

Table 10-43: DMC_PHY_CTL3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Control 4 Register

The `DMC_PHY_CTL4` register controls programmable PHY features.

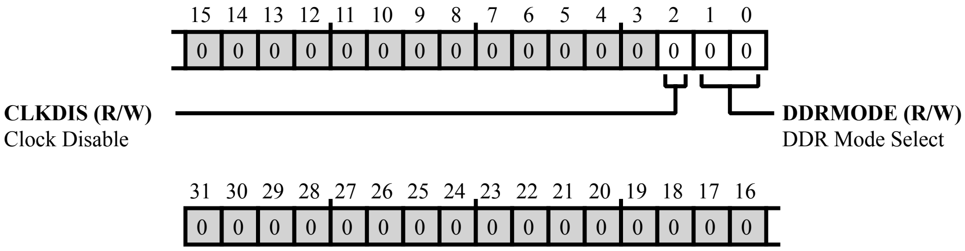

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKDIS (R/W)
Clock Disable
DDRMODE (R/W)
DDR Mode Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 10-34: DMC_PHY_CTL4 Register Diagram

Table 10-44: DMC_PHY_CTL4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CLKDIS	Clock Disable. The `DMC_PHY_CTL4.CLKDIS` bit enables and disables the DDR clock.	
		0	Enable Clock
		1	Disable Clock
1:0 (R/W)	DDRMODE	DDR Mode Select. The `DMC_PHY_CTL4.DDRMODE` bit field selects between the various DDR modes. Not all modes are available on all processors.	
		0	DDR3 Mode
		1	DDR2 Mode
		2	Reserved
		3	LPDDR Mode

11 Static Memory Controller (SMC)

The static memory controller is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to various external memories and peripheral devices, including:

- SRAM
- ROM
- EPROM
- NOR flash memory
- FPGA/ASIC devices

The SMC acts as an SCB slave. The processor SCB interconnect fabric arbitrates accesses to the SMC. On the chip boundary, the SMC connects to an address bus, a data bus, and signal pins for memory control (such as read, write, output enable, and memory select lines).

SMC Features

SMC features include:

- 16-bit I/O width
- Provides flexible timing control through extended timing parameters
- Supports asynchronous access extension (`SMC_ARDY` pin)
- Supports 8-bit data masking writes

SMC Definitions

The timing registers contain bits to program the setup time, hold time, and access time for read and write access to each bank separately. The SMC allows for different setup, hold, or access times for reads and writes. The `SMC_B0TIM` – `SMC_B3TIM` registers control the timing characteristics of the asynchronous memory interface using the following parameter definitions. Each of these parameters can be programmed in terms of SCLK0_0 clock cycles.

Read setup time

The time between the beginning of a memory cycle (`SMC_AMS0` signal low) and the read-enable assertion (`SMC_ARE` signal low).

Read hold time

The time between read-enable deassertion (`SMC_ARE` signal high) and the end of the memory cycle (`SMC_AMS0` signal high).

Read access

The time between read-enable assertion (`SMC_ARE` signal low) and deassertion (`SMC_ARE` signal high).

Write setup time

The time between the beginning of a memory cycle (`SMC_AMS0` signal low) and the write-enable assertion (`SMC_AWE` signal low).

Write hold time

The time between write-enable deassertion (`SMC_AWE` signal high) and the end of the memory cycle (`SMC_AMS0` signal high).

Write access

The time between write-enable assertion (`SMC_AWE` signal low) and deassertion (`SMC_AWE` signal high).

The SMC provides another register for defining more timing characteristics of control signals by programming the extended `SMC_B0TIM` – `SMC_B3TIM` timing registers. These registers contain bits to program following timing characteristics.

Pre-setup time

The number of cycles the `SMC_AMS0` signal is asserted before the `SMC_AOE` signal is asserted.

Pre-access time

The number of cycles inserted after the `SMC_AOE` signal is deasserted and before the `SMC_ARE` signal is asserted for the next access.

Memory idle time

The number of bus idle cycles between the `SMC_AMS0` deasserting edge and next asserting edge.

Memory transition time

The number of bus idle cycles extending the idle time cycles. These idle cycles occur in the case where a subsequent access has a different data direction or the access is to a different bank.

Bus contention

State of the bus in which more than one device on the bus attempts to place values on the bus at the same time. For more information, see Avoiding Bus Contention.

ARDY signal

The SMC uses the `SMC_ARDY` signal to insert wait states for slower asynchronous memories. There is no upper limit to how many wait states the `SMC_ARDY` signal can enter. As long as it is held, the processor waits for the access to the asynchronous memory. Once asserted, the processor accesses the memory according to the timing diagrams. For more information, see ARDY Input Control.

SMC Functional Description

The SMC contains memory-mapped registers that control the access characteristics for each asynchronous memory bank. Different banks can be programmed in various modes and independently-controlled using the functional and cycle time bit settings for each bank.

Independent bank control

The SMC provides separate sets of registers, `SMC_B0CTL` through `SMC_B3CTL` (control), `SMC_B0TIM` through `SMC_B3TIM` (timing) and `SMC_B0ETIM` through `SMC_B3ETIM` (extended timing) to control the mode and timing characteristic of each bank independently. The control registers contain bits for enabling the bank and bits for selecting mode of operation.

Bank select control signal control

The control registers also contain bits to control the type of bank select control signal. External FIFO devices often do not have a separate chip select pin. As a result, for a read, the FIFOs output enable (`SMC_AOE`) pin must be connected to the OR of the `SMC_AMS0` and the `SMC_ARE`. Similarly, the write case requires an OR between `SMC_AMS0` and `SMC_AWE`. The SMC provides this function so that an external OR gate is not required. The appropriate AMS function can be selected for each memory bank region using the `SMC_B0CTL.SELCTRL` bits.

ADSP-SC58x SMC Register List

The Static Memory Controller SMC is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. The SMC acts as a bus slave, and accesses to SMC are arbitrated by the module's system crossbar. On the chip boundary, the SMC is connected to an external memory address bus, a 16-bit data bus and memory control signal pins (read, write, select). This memory interface can support a sizable external memory connected to one or more banks, each bank being controlled by a chip select signal. A set of registers governs SMC operations. For more information on SMC functionality, see the SMC register descriptions. For the memory map, see the product data sheet.

Table 11-1: ADSP-SC58x SMC Register List

Name	Description
SMC_B0CTL	Bank 0 Control Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3ETIM	Bank 3 Extended Timing Register
SMC_B3TIM	Bank 3 Timing Register

SMC Architectural Concepts

The SMC can support connection to multiple different external banks, with each bank controlled by the SMC_AMS[n] chip select signal. Check the processor data sheet for details on the bank address ranges and configurations.

NOTE: The processor data sheet shows the address range allocated to each bank. It is not necessary to populate all of an enabled memory bank.

The processor does not directly support 8-bit accesses to the external memories. So, the SMC address lines start from SMC_A01; there is no SMC_A0 pin.

The SMC does indirectly support 8-bit accesses through the additional byte enable signals SMC_ABE0 and SMC_ABE1. Some 16-bit memory systems allow the processor to perform 8-bit reads and writes, which are selected through the SMC_ABE0 and SMC_ABE1 signals.

The byte enable pins are both three-stated during all asynchronous reads and are driven low during 16-bit asynchronous writes. When an asynchronous write is made to the upper byte of a 16-bit memory, `SMC_ABE1` =0 and `SMC_ABE0` =1. When an asynchronous write is made to the lower byte of a 16-bit memory, `SMC_ABE1` =1 and `SMC_ABE0` =0.

NOTE: Some SRAM devices expect byte-enable signals to be driven low during read accesses rather than being three-stated, which can be achieved using external pull-down resistors. For applications requiring alternate functions on the byte-enable pins during run time where pull-down resistors are not an option, the same functionality can be achieved using the external logic shown as follows:

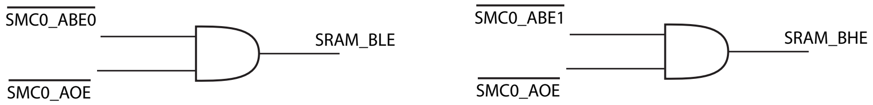


Figure 11-1: External Logic

Avoiding Bus Contention

Bus contention occurs when one device is getting off the bus and another is getting on. If the first device is slow to three-state and the second device is quick to drive, the devices contend. Bus contention causes excessive power dissipation and can lead to device failure.

There are two cases where contention can occur.

- When a read followed by a write to the same memory space occurs, there is a potential for bus contention. The data bus drivers used for the write can potentially contend with the drivers used by the memory device being read.
- When there are back-to-back reads from two different memory spaces, the two memory devices addressed by the two reads can contend at the transition between the two read operations.

To avoid contention, program the turnaround time appropriately in the extended time registers (`SMC_B0ETIM` – `SMC_B3ETIM`). The programming is done by setting the number of clock cycles between these types of accesses on a bank-by-bank basis.

The idle time bit (`SMC_B0ETIM.IT`) applies to similar back-to-back access types on the same bank. The transition time bit (`SMC_B0ETIM.TT`) applies to the `SMC_B0ETIM.IT` bit. For actual turnaround situations, idle time and transition time function in an accumulated fashion. The sequence of access types and times are:

- A write followed by write to same bank – `SMC_B0ETIM.IT`
- A read followed by read to same bank – `SMC_B0ETIM.IT`
- A write followed by read to same bank – `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`
- A read followed by write to same bank – `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`
- Any access to a given bank followed by any access to a different bank – `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`

The reset value of turnaround transition time is two cycles. Program the SMC_B0ETIM.TT bit to a value either greater than or equal to two cycles, depending on memory AC-timing specifications. It is important to be aware that the SMC_B0ETIM.TT bit is programmed to 0 *only* when:

- There are *either* only read accesses *or* only write accesses possible to external memory devices for the current device configuration or processor operation situation.

ARDY Input Control

Each bank can be programmed to sample the SMC_ARDY input after the read or write access timer has counted down. It can also be programmed to ignore this input signal. If enabled and disabled at the sample window, the SMC module uses SMC_ARDY to extend the access time, as required.

The processor treats SMC_ARDY as an asynchronous input. The input must reach the desired value (either asserted or deasserted) more than two SCLK0_0 cycles before the completion of access time (scheduled rising edge of SMC_AWE or SMC_ARE). This timing determines whether the SMC extends the access with the assertion of SMC_ARDY. After the processor samples SMC_ARDY high, the total delay between SMC_ARDY going high at the pads and SMC_ARE being deasserted at the pads is a maximum of five SCLK0_0 cycles. (The memory device asserts SMC_ARDY.)

Asynchronous SRAM writes are also possible with the SMC_ARDY signal enabled. In asynchronous SRAM writes, the write access is extended beyond the programmed write access cycles depending on the SMC_ARDY signal state. Once SMC_ARDY is sampled asserted, the SMC_AWE signal is deasserted after two *CLKOUT* cycles and the write access ends.

The polarity of SMC_ARDY is programmable on a per-bank basis. Since SMC_ARDY is not sampled until an access is in-progress to a bank in which the SMC_ARDY enable is asserted, it is not driven by default. When using flash memory, connect the WAIT input to SMC_ARDY.

To avoid stalls in the case of erroneous SMC_ARDY behavior, set the SMC_B0CTL.RDYABTEN bit to enable the SMC_ARDY abort counter. When the abort counter is enabled, it starts counting down as soon as the programmed read/write access cycles expire. If the SMC interface does not sample the SMC_ARDY signal as asserted within 64 cycles, a counter-timeout occurs. This timeout ensures that the processor does not hang if SMC_ARDY is not sampled correctly.

SMC Operating Modes

The SMC supports the following operating modes:

- Asynchronous Flash Mode
- Asynchronous Page Mode

Asynchronous Flash Mode

When the access selected mode is asynchronous flash (`SMC_B0CTL.MODE` =01), external bank accesses operate the same as in standard asynchronous mode, except for the pin configuration. Use this mode when accessing burst devices in non-read array modes.

Asynchronous Page Mode

When asynchronous page mode access is selected (`SMC_B0CTL.MODE` =10), asynchronous page reads are enabled. The SMC module supports page sizes of 4, 8 and 16 words. When performing a page mode read, the first access in the page proceeds according to the read access time configured in `SMC_B0TIM` register. This access opens the page and the subsequent reads in that page have a period equal to the page wait states programmed in the `SMC_B0ETIM` register. Besides the start of the setup phase, the read address is incremented at the start of every page cycle.

The SMC module supports page mode access only for back-to-back accesses, such as cache line fills (16 words), 64-bit instruction reads (4 words), and DMA reads. It treats write accesses in asynchronous page mode as simple asynchronous flash write accesses.

SMC Event Control

SMC event control consists of recording the status of SMC errors. Accesses to reserved locations and writes to read-only registers result in bus errors. The SMC translates bus errors into internal SCB crossbar errors which get translated into interrupts. To report errors occurring in the slave memory devices (for both this memory interface and the MMR interface), the core combines the SCB crossbar response signals. This combination generates a combined error signal indication which is routed to the fault management unit.

SMC Programmable Timing Characteristics

This section describes the programmable timing characteristics for the SMC. Timing relationships depend on the programming of the SMC bank registers, whether initiation is from the core or from DMA. The relationships also depend on the sequence of transactions (read followed by read, read followed by write, and others).

NOTE: All memory control, address, and data signals are driven out of the chip based on the falling edge of the *CLKOUT* signal. The *CLKOUT* signal is SCLK0_0 on the chip pins (pad delayed).

Asynchronous SRAM Reads and Writes

The ***Basic Asynchronous SRAM Write Followed by Read*** figure shows a basic single write and read operation to an external device with the SMC programmed in asynchronous SRAM mode.

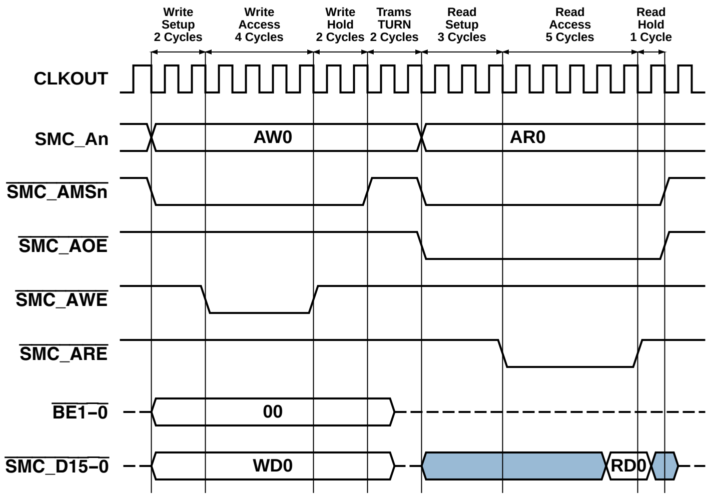

Write Setup 2 Cycles
Write Access 4 Cycles
Write Hold 2 Cycles
Trams TURN 2 Cycles
Read Setup 3 Cycles
Read Access 5 Cycles
Read Hold 1 Cycle
CLKOUT
SMC_An
AW0
AR0
SMC_AMSn
SMC_AOE
SMC_AWE
SMC_ARE
BE1–0
00
SMC_D15–0
WD0
RD0

Figure 11-2: Basic Asynchronous SRAM Write Followed by Read

For the current bank, the programmed time cycles are:

- Write setup time = 2 cycles
- Write access time = 4 cycles
- Write hold time is = 2 cycles
- Read setup time = 3 cycles
- Read access time = 5 cycles
- Read hold time = 1 cycle
- Turnaround transition time = 2 cycles
- Idle transition time = 0 cycles

The asynchronous SRAM bus cycles proceed as follows.

1. At the start of the write setup period, the chip select signal (`SMC_AMS[n]`) for the target bank is asserted. The write data (`WD0`), address (`AW0`), and byte enables become valid.
2. At the end of the setup phase and at the start of the write access period, the write enable (`SMC_AWE`) is asserted.
3. At the end of the programmed write access, the `SMC_AWE` signal is deasserted. The target device is assumed to have captured the write data before `SMC_AWE` is deasserted.
4. At the end of the write hold period, the `SMC_AWE` signal is deasserted because the pending access is a read access, and the turnaround transition time cycles start. The write data and byte enables become invalid within 1 cycle of the `SMC_AMS0` signal deasserting.
5. At the end of turnaround transition time, the read setup period starts with the assertion of the `SMC_AMS0` and `SMC_AOE` signals and a new read address (`AR0`) is presented on the address bus.

6. At the start of the read access period, the read enable signal, `SMC_ARE` is asserted.
7. At the end of the read access period, the `SMC_ARE` signal is deasserted and the read hold period starts. Read data is latched along with `SMC_ARE` deasserting.
8. At the end of the read hold period, the SMC pulls the `SMC_AMS[n]` signal high and appends turnaround transition cycles unless there is a pending read request to the same bank.

Asynchronous SRAM Reads with IDLE Transition Cycles Inserted

The *Asynchronous SRAM Read with IDLE Transition* figure shows two consecutive asynchronous SRAM modes reads to the same bank separated by programmed IDLE transition time cycles.

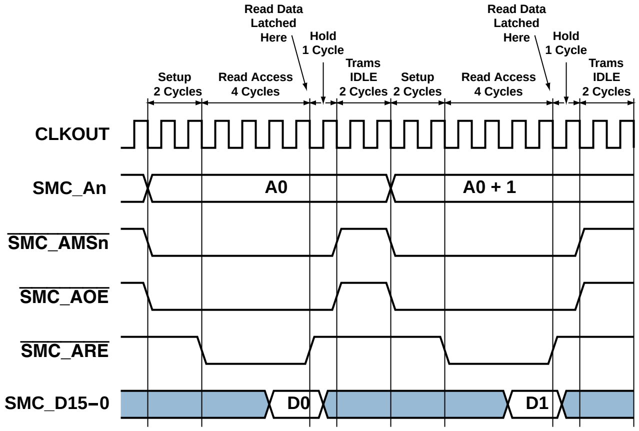

Read Data Latched Here
Hold 1 Cycle
Trams IDLE 2 Cycles
Setup 2 Cycles
Read Access 4 Cycles
Read Data Latched Here
Hold 1 Cycle
Trams IDLE 2 Cycles
Setup 2 Cycles
Read Access 4 Cycles
CLKOUT
SMC_An
A0
A0 + 1
SMC_AMSn
SMC_AOE
SMC_ARE
SMC_D15–0
D0
D1

Figure 11-3: Asynchronous SRAM Read with IDLE Transition

Programmed cycle times are:

- `SMC_B0TIM.RST` = 2 cycles
- `SMC_B0TIM.RAT` = 4 cycles
- `SMC_B0TIM.RHT` = 1 cycle
- IDLE transition time = 2 cycles

At the start of the IDLE transition cycle, the SMC deasserts the `SMC_AMS[n]` and `SMC_AOE` signals. The setup period of the second read starts at the end of the IDLE transition cycle with the assertion of the `SMC_AMS[n]` and `SMC_AOE` signals and a new address on the address bus.

High-Speed Asynchronous SRAM Read Burst

The *Fast Asynchronous SRAM Reads, Burst of Four Word* figure shows a high-speed asynchronous SRAM read bus cycle. This read bus cycle is typical for SRAM devices with small access times connecting through SCB read bursts, especially for boot purposes.

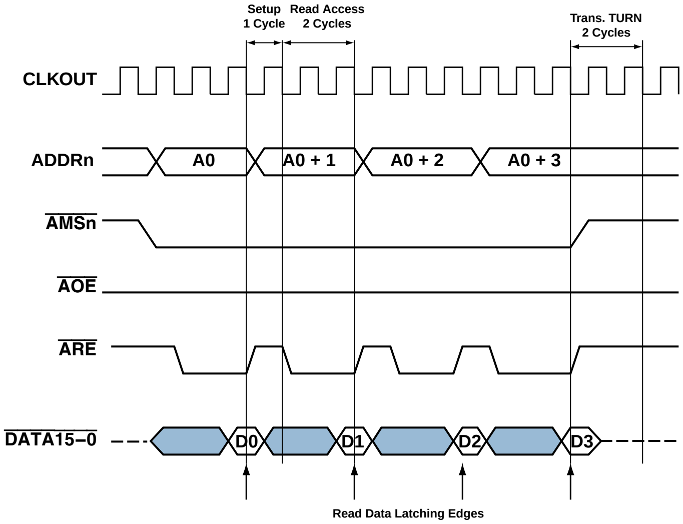


Figure 11-4: Fast Asynchronous SRAM Reads, Burst of Four Word

In this case, the target SMC bank has been programmed with:

- read setup time = 1 cycle
- read access time = 2 cycles
- read hold time = 0
- `SMC_B0ETIM.PREAT` = 0
- `SMC_B0ETIM.PREST` = 0
- IDLE transition time = 0

The `SMC_AMS[n]` signal is asserted at the start of the setup cycle of the first read out of the burst. Since the hold time and the idle transition time have been programmed to 0, the `SMC_AMS[n]` signal does not deassert until the entire set of reads concludes. Only the `SMC_ARE` signal deasserts periodically for 1 cycle for the setup period. The read address changes to the next address at the start of each individual setup cycle. Read data words are latched at the end of each individual read access period.

High-Speed Asynchronous SRAM Writes

High-speed asynchronous SRAM writes are similar to the high-speed read accesses. The ***Fast Asynchronous SRAM Writes*** figure shows the bus protocol for a write burst of 4 words. Here, the write setup time is 1 cycle and the write access time has been programmed to 2 cycles. Write hold time, pre-access time, pre-setup time, and idle transition time are programmed to 0.

The chip select signal `SMC_AMS[n]` asserts at the start of the entire write burst and deasserts only at the end of the last individual write access period. Write address, byte enables and write data for each individual write access are presented onto the bus at the start of each individual write setup cycle. The `SMC_AWE` signal asserts for the write access period and deasserts during the setup period for each individual data write.

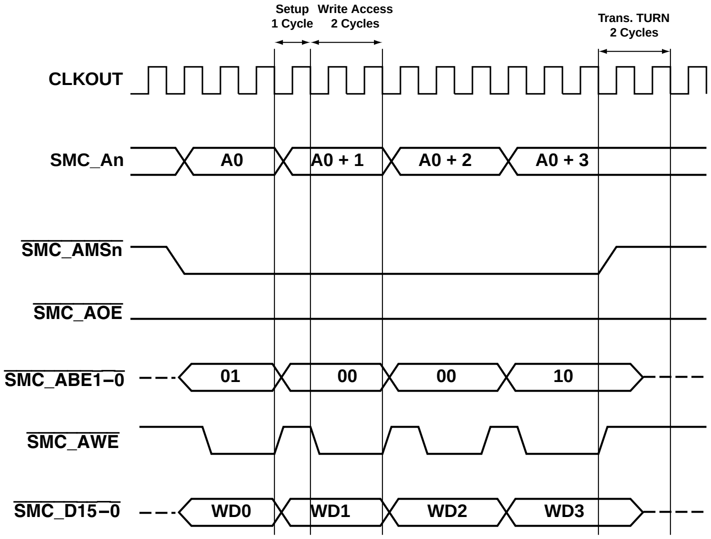

Setup
1 Cycle
Write Access
2 Cycles
Trans. TURN
2 Cycles
CLKOUT
SMC_An
A0
A0 + 1
A0 + 2
A0 + 3
SMC_AMSn
SMC_AOE
SMC_ABE1–0
01
00
00
10
SMC_AWE
SMC_D15–0
WD0
WD1
WD2
WD3

Figure 11-5: Fast Asynchronous SRAM Writes

Asynchronous SRAM Reads with ARDY

The *Asynchronous SRAM Read with ARDY* figure shows an extended asynchronous SRAM read bus cycle with `SMC_ARDY` enabled.

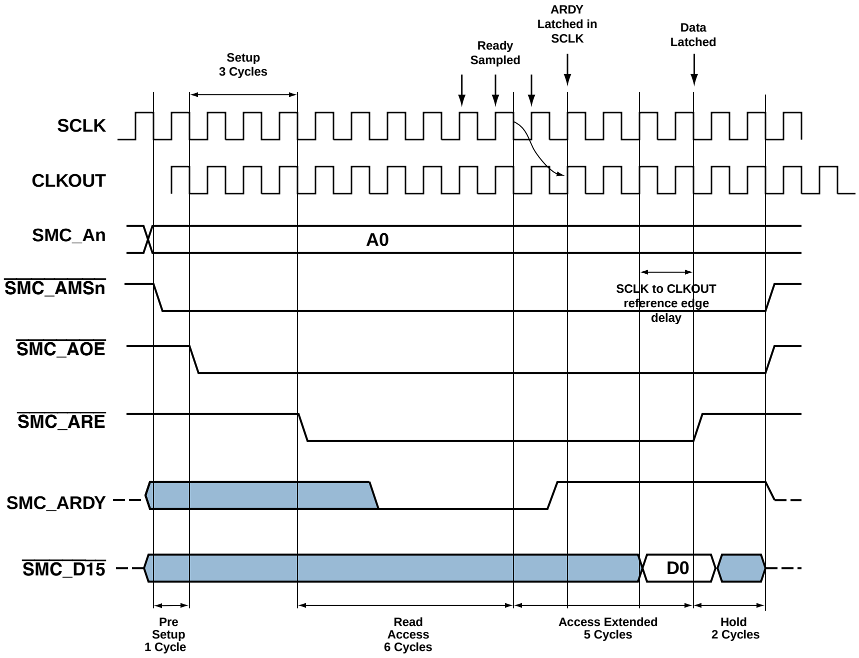

ARDY
Latched in
SCLK
Data
Latched
Ready
Sampled
Setup
3 Cycles
SCLK
CLKOUT
SMC_An
A0
SMC_AMSn
SCLK to CLKOUT
reference edge
delay
SMC_AOE
SMC_ARE
SMC_ARDY
SMC_D15
D0
Pre
Setup
1 Cycle
Read
Access
6 Cycles
Access Extended
5 Cycles
Hold
2 Cycles

Figure 11-6: Asynchronous SRAM Read with ARDY

NOTE: SCLK in the *Asynchronous SRAM Read with ARDY* figure is SCLK0_0.

The programmed SMC bank control parameters are:

- Pre-setup time = 1 cycle
- Read setup time = 3 cycles
- Read access time = 6 cycles
- Read hold time = 2 cycles
- `SMC_B0CTL.RDYPOL` =1 (memory is ready when `SMC_ARDY` =1)

The bus cycles proceed as follows:

- At the start of the pre-setup phase, `SMC_AMS[n]` asserts, and read address `SMC_A01` is presented on the address bus.
- At the start of the setup period, `SMC_AOE` is asserted.
- At the start of the read access, `SMC_ARE` is asserted.
- The CLKOUT signal is SCLK0_0 which is driven out of the pads. The CLKOUT signal falling edge can be delayed from the internal SCLK0_0 falling edge. See the data sheet for the specification related to this delay. All output signals out of the pads, for example `SMC_ARE` and `SMC_AOE`, are driven with regard to the falling edge of CLKOUT.
- The SMC starts sampling the `SMC_ARDY` signal on every rising edge of internal SCLK0_0 two cycles before the programmed number of read access cycles expires. The read access is extended (`SMC_ARE` is kept asserted) until the SMC samples `SMC_ARDY` high.
- Once the SMC samples the `SMC_ARDY` signal (asserted by memory device) high in SCLK0_0, the read signal is pulled off internally in the SCLK0_0 domain. The total delay between the `SMC_ARDY` signal going high at the pads and the deassertion of the `SMC_ARE` signal at the pads can be a maximum of five SCLK0_0 cycles.
- Read data is latched at the falling edge of CLKOUT on the same edge where the SMC deasserts `SMC_ARE`.
- Hold bus cycles start after the SMC deasserts the `SMC_ARE` signal.
- At the end of the hold period, the `SMC_AMS[n]` and `SMC_AOE` signals deassert and the SMC goes into the transition state.

Asynchronous Flash Reads

The ***Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles*** figure illustrates a single asynchronous flash mode read bus cycle.

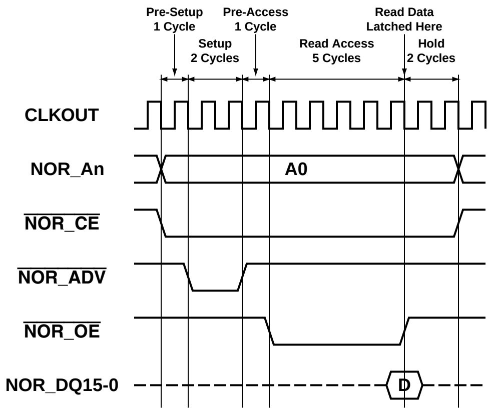


Figure 11-7: Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles

In this case, the target SMC bank has been programmed with:

- Pre-setup time = 1 cycle
- Read setup time = 2 cycles
- Pre-access time = 1 cycle
- Read access time = 5 cycles
- Read hold time = 2 cycles

The read bus cycle is almost identical to the asynchronous SRAM read bus cycle. The only difference is the behavior of the `SMC_AOE` signal which the SMC uses as the flash address valid `SMC_NORDV` signal. The flash address valid `SMC_NORDV` signal asserts at the start of the setup cycle and deasserts at the end of the setup cycle.

The pre-access cycle inserts a one-cycle gap between the deassertion of the flash address valid `SMC_NORDV` signal and the assertion of the flash read strobe `NOR_OE` at the start of read access. The SMC also uses asynchronous flash reads with `SMC_ARDY` enabled for flash devices which use `SMC_NORWT` in asynchronous mode. In this case, the read bus cycle operation is identical to the asynchronous SRAM with `SMC_ARDY` enabled except for the `SMC_AOE`/`SMC_NORDV` signal behavior.

The ***32-bit Asynchronous Flash Read*** figure shows a 32-bit read access to a flash device in asynchronous mode which is split into two 16-bit external memory accesses. For this bank, read setup and read hold are programmed as two cycles whereas the read access time is five cycles. The flash device chip select signal (`NOR_CE`) remains asserted for the entire duration of both read accesses. `NOR_CE` is deasserted at the end of the hold period of the second read access. The SMC asserts the `SMC_NORDV` signal during the setup phase of both read accesses. Read data is latched at the end of the read access period.

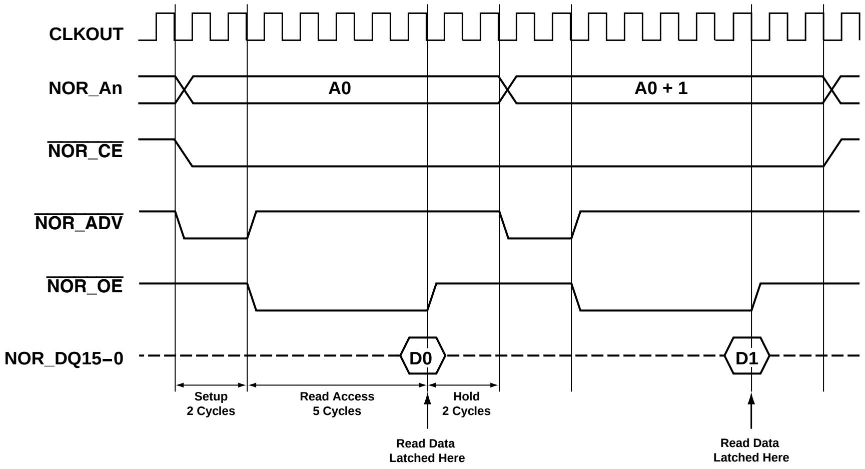

CLKOUT
NOR_An
A0
A0 + 1
NOR_CE
NOR_ADV
NOR_OE
NOR_DQ15–0
D0
D1
Setup
2 Cycles
Read Access
5 Cycles
Hold
2 Cycles
Read Data
Latched Here
Read Data
Latched Here

Figure 11-8: 32-bit Asynchronous Flash Read

Asynchronous Flash Writes

The ***Asynchronous Flash Write Operation*** figure shows a single asynchronous flash write bus cycle.

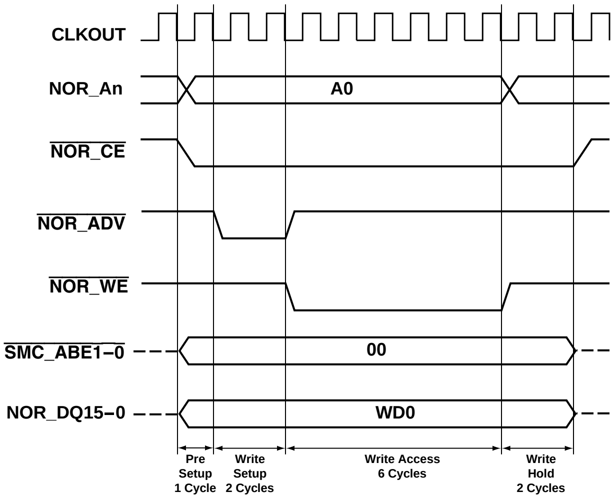

CLKOUT
NOR_An
A0
NOR_CE
NOR_ADV
NOR_WE
SMC_ABE1–0
00
NOR_DQ15–0
WD0
Pre
Setup
1 Cycle
Write
Setup
2 Cycles
Write Access
6 Cycles
Write
Hold
2 Cycles

Figure 11-9: Asynchronous Flash Write Operation

For this example, the SMC has been programmed with:

- Pre-setup time = 1 cycle
- Write setup time = 2 cycles
- Write access time = 6 cycles
- Write hold time = 2 cycles
- Pre-access time = 0

The asynchronous flash write bus cycle is again almost identical to the asynchronous SRAM write. The `SMC_AWE` pin is connected to flash write enable signal (NOR_WE). However, in asynchronous flash writes the SMC uses the

`SMC_AOE` signal as the address valid signal (`SMC_NORDV`). The `SMC_AOE` signal asserts during the setup period, unlike in asynchronous SRAM writes where the `SMC_AOE` signal never asserts.

Asynchronous Flash Page Mode Reads

The ***Asynchronous Page Mode Read Bus Cycle*** figure shows an asynchronous page mode bus read cycle for a burst of five reads. The reads are split into four reads followed by a single read.

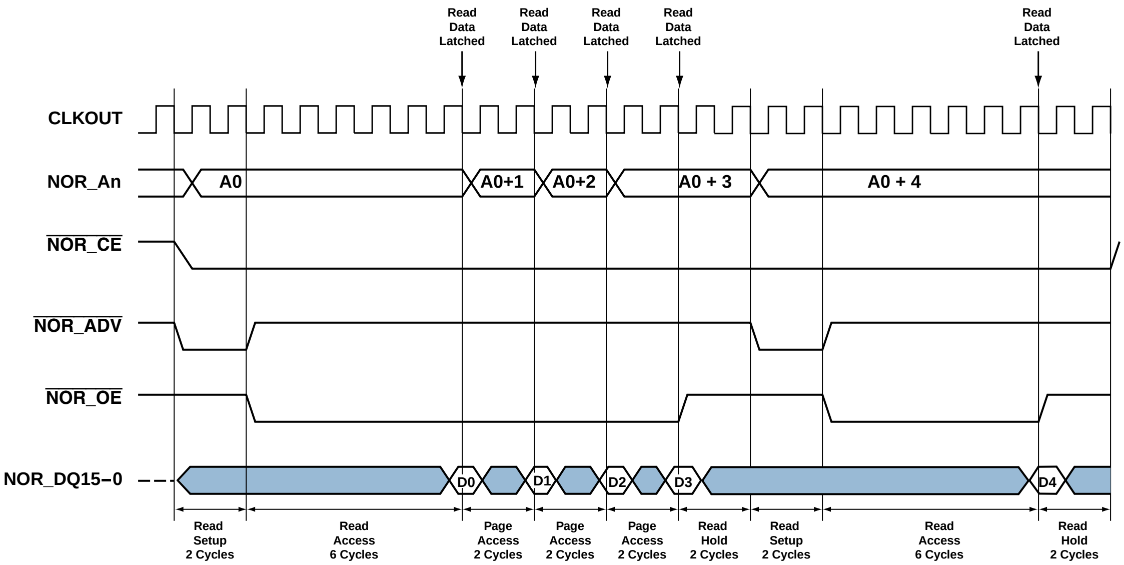

Read Data Latched
Read Data Latched
Read Data Latched
Read Data Latched
Read Data Latched
CLKOUT
NOR_An
A0
A0+1
A0+2
A0 + 3
A0 + 4
NOR_CE
NOR_ADV
NOR_OE
NOR_DQ15–0
D0
D1
D2
D3
D4
Read Setup 2 Cycles
Read Access 6 Cycles
Page Access 2 Cycles
Page Access 2 Cycles
Page Access 2 Cycles
Read Hold 2 Cycles
Read Setup 2 Cycles
Read Access 6 Cycles
Read Hold 2 Cycles

Figure 11-10: Asynchronous Page Mode Read Bus Cycle

The programmed bank parameters are:

- Read setup time = 2 cycles
- Read access time = 6 cycles
- Page wait = 2 cycles
- Hold time =2 cycles

The maximum number of read bursts in a total page access depends on the bank `SMC_B0CTL.PGSZ` bits (00 =4 words, 01 =8 words, 1x =16 words). The first read access is extended for three more page-read cycles whose period is equal to the page wait states. Besides the start of the setup phase, the read address is incremented at the start of every page cycle. Read data is latched with the falling edge of CLKOUT the end of the read access period, and also at the end of the page cycles.

Asynchronous FIFO Reads and Writes

The ***Asynchronous FIFO Read Bus Cycles*** figure shows the read bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with `SMC_B0CTL.SELCTRL` =01 (`SMC_AMS[n]` is OR'ed with `SMC_ARE`).

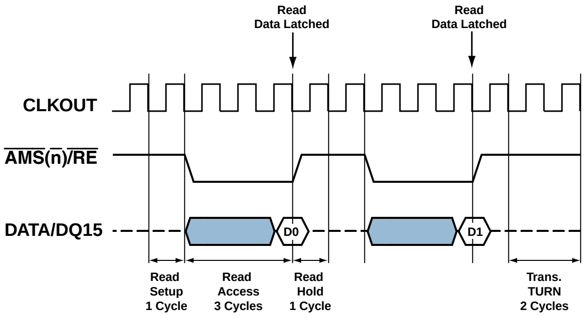


Figure 11-11: Asynchronous FIFO Read Bus Cycles

Other settings are:

- Read setup time = 1 cycle
- Read access time = 3 cycles
- Read hold time = 1 cycle
- Idle transition time = 0 cycles
- Turnaround transition time = 2 cycles

The `SMC_AMS[n]` signal connects to the read enable (`RE`) of the FIFO device, and the data bus connects to the output data bus (`DQ`) of the FIFO. The `SMC_AMS[n]` signal or the FIFO read strobe assert only during the read access. Read data is latched at the falling edge of CLKOUT at the end of the read access, when the SMC deasserts `SMC_AMS[n]`.

The ***Asynchronous FIFO Write Bus Cycles*** figure illustrates the write bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with `SMC_B0CTL.SELCTRL` =10 (`SMC_AMS[n]` is OR'ed with `SMC_AWE`). Other settings are:

- Write setup time = 1 cycle
- Write access time = 3 cycles
- Write hold time = 1 cycle
- Idle transition time = 0
- Turnaround transition time = 2 cycles

The `SMC_AMS[n]` signal connects to the write enable (WE) of the FIFO device. The data bus connects to the input data bus (DIN) of the FIFO. The `SMC_AMS[n]` signal or the FIFO write strobe asserts only during the write access. However, the SMC asserts write data at the start of the setup cycle and removes it at the end of the hold period for each individual write access.

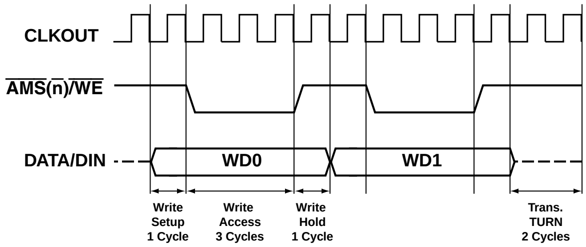


Figure 11-12: Asynchronous FIFO Write Bus Cycles

SMC Programming Model

The following general guidelines are used for configuring and enabling the SMC interface. Failure to follow these guidelines can lead to erroneous behavior.

- In asynchronous page mode, always program `SMC_B0CTL.RDYEN` to 0.
- Enable the ARDY abort counter (the `SMC_B0CTL.RDYABTEN` bit =1) whenever the `SMC_ARDY` signal is enabled (the `SMC_B0CTL.RDYEN` is set to 1). This step ensures that the interface does not hang due to erroneous `SMC_ARDY` signal behavior or erroneous sampling of the `SMC_ARDY` signal.
- Do not program read access time (`SMC_B0TIM.RAT`), write access time (`SMC_B0TIM.WAT`), read setup time (`SMC_B0TIM.RST`), and write setup time (`SMC_B0TIM.WST`) to 0.
- Never program page mode wait-states (`SMC_B0ETIM.PGWS`) to 0 or 1.
- Program the page size bits (`SMC_B0CTL.PGSZ`) to match the configurations of the flash device connected to the SMC interface.
- Select the `SMC_B0CTL.RDYPOL` bit to be the complement of the `WAIT` polarity that is configured in the flash device.
- In asynchronous SRAM and asynchronous flash modes with `SMC_ARDY` enabled, and where `SMC_B0TIM.RHT`, `SMC_B0TIM.WHT`, `SMC_B0TIM.RAT`, and `SMC_B0TIM.WAT` are the read and write hold and access times and `SMC_B0ETIM.IT` and `SMC_B0ETIM.TT` are the idle and transition times, ensure that the following conditions are true:
  - `SMC_B0TIM.RHT` + `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`≥ 2
  - `SMC_B0TIM.WHT` + `SMC_B0ETIM.IT` + `SMC_B0ETIM.TT`≥ 2
  - `SMC_B0TIM.RAT`≥ 5
  - `SMC_B0TIM.WAT`≥ 5

ADSP-SC58x SMC Register Descriptions

Static Memory Controller (SMC) contains the following registers.

Table 11-2: ADSP-SC58x SMC Register List

Name	Description
SMC_B0CTL	Bank 0 Control Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3ETIM	Bank 3 Extended Timing Register
SMC_B3TIM	Bank 3 Timing Register

Bank 0 Control Register

The `SMC_B0CTL` register enables bank 0 accesses and configures the memory access features for this bank.

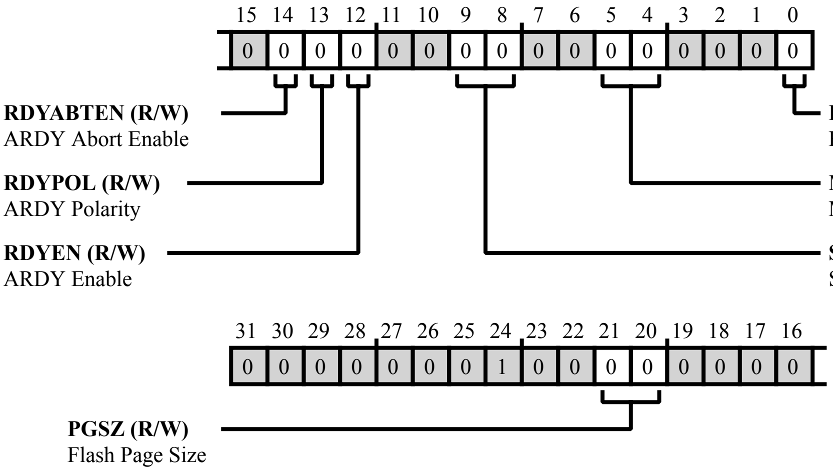

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDYABTEN (R/W)
ARDY Abort Enable
RDYPOL (R/W)
ARDY Polarity
RDYEN (R/W)
ARDY Enable
EN (R/W)
Bank 0 Enable
MODE (R/W)
Memory Access Mode
SELCTRL (R/W)
Select Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
PGSZ (R/W)
Flash Page Size

Figure 11-13: SMC_B0CTL Register Diagram

Table 11-3: SMC_B0CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The `SMC_B0CTL.PGSZ` bits select the flash page size, if page flash or sync burst flash protocol has been enabled (`SMC_B0CTL.MODE` > 1). Note that the `SMC_B0CTL.PGSZ` bits must be set to match the flash protocol of the external flash memory device in the system. The typical `SMC_B0CTL.PGSZ` selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical `SMC_B0CTL.PGSZ` selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 11-3: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The `SMC_B0CTL.RDYABTEN` bit enables the abort counter for the `SMC_ARDY` pin, if enabled (`SMC_B0CTL.RDYEN` =1). After `SMC_B0TIM.RAT` or `SMC_B0TIM.WAT` cycles, the SMC starts sampling the `SMC_ARDY` pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK0_0. If the SMC detects that `SMC_ARDY` remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The `SMC_B0CTL.RDYPOL` bit selects the polarity (active high or low) for the `SMC_ARDY` pin, if enabled (`SMC_B0CTL.RDYEN` =1). When the SMC samples the `SMC_ARDY` pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The `SMC_B0CTL.RDYEN` bit enables `SMC_ARDY` pin operation for bank 0 accesses. When enabled, the SMC uses `SMC_ARDY` (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores `SMC_ARDY` for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The `SMC_B0CTL.SELCTRL` bits select the handling of the `SMC_AMS[n]`, `SMC_ARE`, `SMC_AOE`, and `SMC_AWE` pins for memory access control.	
		0	AMS0 only
		1	AMS0 ORed with ARE
		2	AMS0 ORed with AOE
		3	AMS0 ORed with AWE

Table 11-3: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The `SMC_B0CTL.MODE` bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 0 Enable. The `SMC_B0CTL.EN` bit enables accesses to the memory in bank 0. When this bit is disabled, accesses to bank 0 return an error response.	
		0	Disable access
		1	Enable access

Bank 0 Extended Timing Register

The `SMC_B0ETIM` register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the `SMC_B0TIM` register.

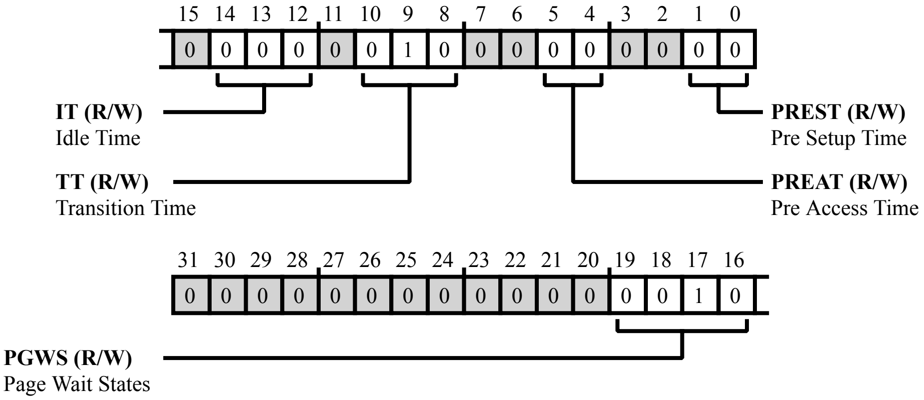

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
IT (R/W)
Idle Time
TT (R/W)
Transition Time
PREST (R/W)
Pre Setup Time
PREAT (R/W)
Pre Access Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PGWS (R/W)
Page Wait States

Figure 11-14: SMC_B0ETIM Register Diagram

Table 11-4: SMC_B0ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The `SMC_B0ETIM.PGWS` bits select a page access extension time (in SCLK0_0 cycles) that the SMC waits during read accesses when configured for flash page protocol (`SMC_B0CTL.MODE` =2). The wait time is from 2 to 15 SCLK0_0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0_0 clock cycles
14:12 (R/W)	IT	Idle Time. The `SMC_B0ETIM.IT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC waits between de-asserting the `SMC_AMS[n]` pin and asserting the `SMC_AMS[n]` pin for the next access. Note that the `SMC_B0ETIM.IT` period may be extended using the `SMC_B0ETIM.TT` selection. The idle time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		7	7 SCLK0_0 clock cycles

Table 11-4: SMC_B0ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The `SMC_B0ETIM.TT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC extends the `SMC_B0ETIM.IT` to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK0_0 cycles.	
		0	No bank transition
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The `SMC_B0ETIM.PREAT` bits select the pre-access time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AOE`/ADV pin before asserting the `SMC_ARE`/`SMC_AWE` pin for the current access. The pre-access time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The `SMC_B0ETIM.PREST` bits select the pre-setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AMS[n]` pin before asserting the `SMC_AOE`/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles

Bank 0 Timing Register

The `SMC_B0TIM` register configures bank 0 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

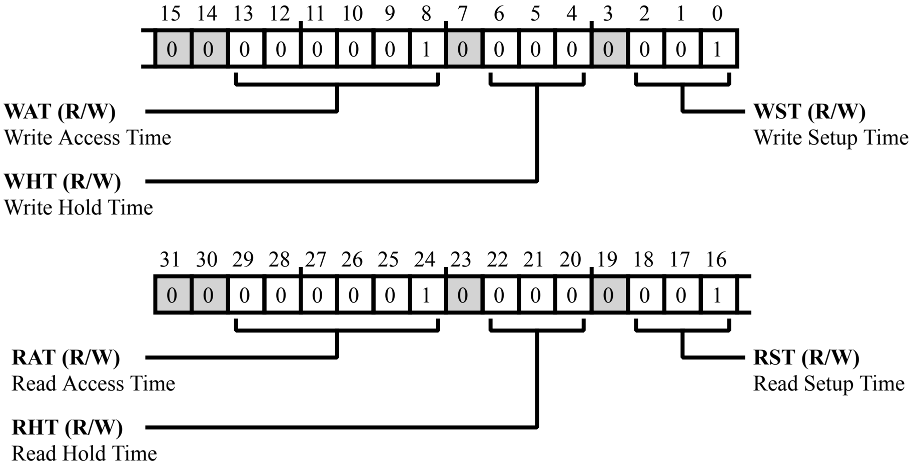

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
WAT (R/W)
Write Access Time
WHT (R/W)
Write Hold Time
WST (R/W)
Write Setup Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
RAT (R/W)
Read Access Time
RHT (R/W)
Read Hold Time
RST (R/W)
Read Setup Time

Figure 11-15: SMC_B0TIM Register Diagram

Table 11-5: SMC_B0TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The `SMC_B0TIM.RAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_ARE` pin for a read access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The `SMC_B0TIM.RHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_ARE` pin before asserting the `SMC_AOE` pin for the next access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Table 11-5: SMC_B0TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The `SMC_B0TIM.RST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_ARE` pin for an access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The `SMC_B0TIM.WAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AWE` pin for a write access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The `SMC_B0TIM.WHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AWE` pin before de-asserting the `SMC_AOE` pin for the current access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The `SMC_B0TIM.WST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_AWE` pin for a write access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Bank 1 Control Register

The SMC_B1CTL register enables bank 1 accesses and configures the memory access features for this bank.

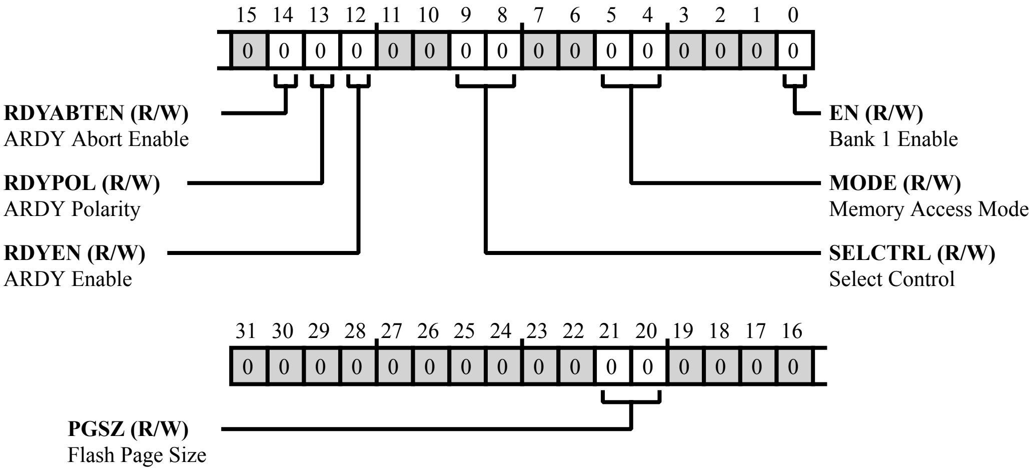

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDYABTEN (R/W)
ARDY Abort Enable
RDYPOL (R/W)
ARDY Polarity
RDYEN (R/W)
ARDY Enable
EN (R/W)
Bank 1 Enable
MODE (R/W)
Memory Access Mode
SELCTRL (R/W)
Select Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PGSZ (R/W)
Flash Page Size

Figure 11-16: SMC_B1CTL Register Diagram

Table 11-6: SMC_B1CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B1CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B1CTL.MODE > 1). Note that the SMC_B1CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B1CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B1CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 11-6: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	RDYABTEN	ARDY Abort Enable. The `SMC_B1CTL.RDYABTEN` bit enables the abort counter for the `SMC_ARDY` pin, if enabled (`SMC_B1CTL.RDYEN` =1). After `SMC_B1TIM.RAT` or `SMC_B1TIM.WAT` cycles, the SMC starts sampling the `SMC_ARDY` pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK0_0. If the SMC detects that `SMC_ARDY` remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus. 0 Disable abort counter 1 Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The `SMC_B1CTL.RDYPOL` bit selects the polarity (active high or low) for the `SMC_ARDY` pin, if enabled (`SMC_B1CTL.RDYEN` =1). When the SMC samples the `SMC_ARDY` pin in the selective active state, the transaction completes. 0 Low active ARDY 1 High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The `SMC_B1CTL.RDYEN` bit enables `SMC_ARDY` pin operation for bank 1 accesses. When enabled, the SMC uses `SMC_ARDY` (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores `SMC_ARDY` for accesses to this memory bank. 0 Disable ARDY 1 Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The `SMC_B1CTL.SELCTRL` bits select the handling of the `SMC_AMS[n]`, `SMC_ARE`, `SMC_AOE`, and `SMC_AWE` pins for memory access control. 0 AMS1 only 1 AMS1 ORed with ARE 2 AMS1 ORed with AOE 3 AMS1 ORed with AWE

Table 11-6: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The `SMC_B1CTL.MODE` bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 1 Enable. The `SMC_B1CTL.EN` bit enables accesses to the memory in bank 1. When this bit is disabled, accesses to bank 1 return an error response.	
		0	Disable access
		1	Enable access

Bank 1 Extended Timing Register

The `SMC_B1ETIM` register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the `SMC_B1TIM` register.

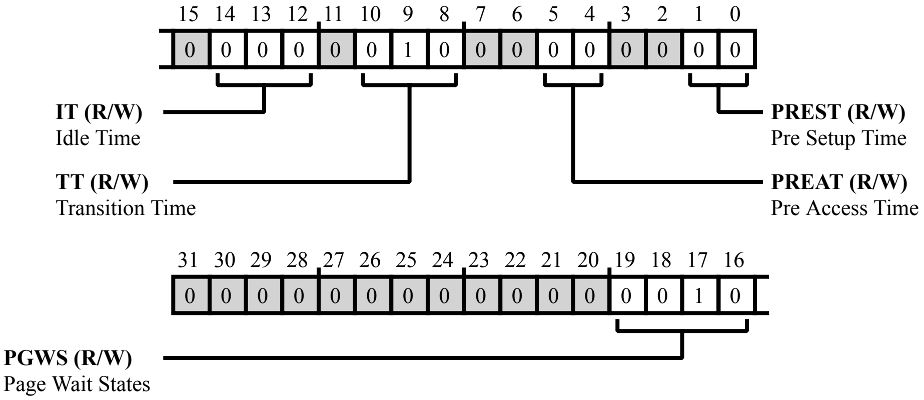

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
IT (R/W)
Idle Time
TT (R/W)
Transition Time
PREST (R/W)
Pre Setup Time
PREAT (R/W)
Pre Access Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PGWS (R/W)
Page Wait States

Figure 11-17: SMC_B1ETIM Register Diagram

Table 11-7: SMC_B1ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The `SMC_B1ETIM.PGWS` bits select a page access extension time (in SCLK0_0 cycles) that the SMC waits during read accesses when configured for flash page protocol (`SMC_B1CTL.MODE` =2). The wait time is from 2 to 15 SCLK0_0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0_0 clock cycles
14:12 (R/W)	IT	Idle Time. The `SMC_B1ETIM.IT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC waits between de-asserting the `SMC_AMS[n]` pin and asserting the `SMC_AMS[n]` pin for the next access. Note that the `SMC_B1ETIM.IT` period may be extended using the `SMC_B1ETIM.TT` selection. The idle time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		7	7 SCLK0_0 clock cycles

Table 11-7: SMC_B1ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The `SMC_B1ETIM.TT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC extends the `SMC_B1ETIM.IT` to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK0_0 cycles.	
		0	No bank transition
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The `SMC_B1ETIM.PREAT` bits select the pre-access time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AOE`/ADV pin before asserting the `SMC_ARE`/`SMC_AWE` pin for the current access. The pre-access time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The `SMC_B1ETIM.PREST` bits select the pre-setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AMS[n]` pin before asserting the `SMC_AOE`/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles

Bank 1 Timing Register

The `SMC_B1TIM` register configures bank 1 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

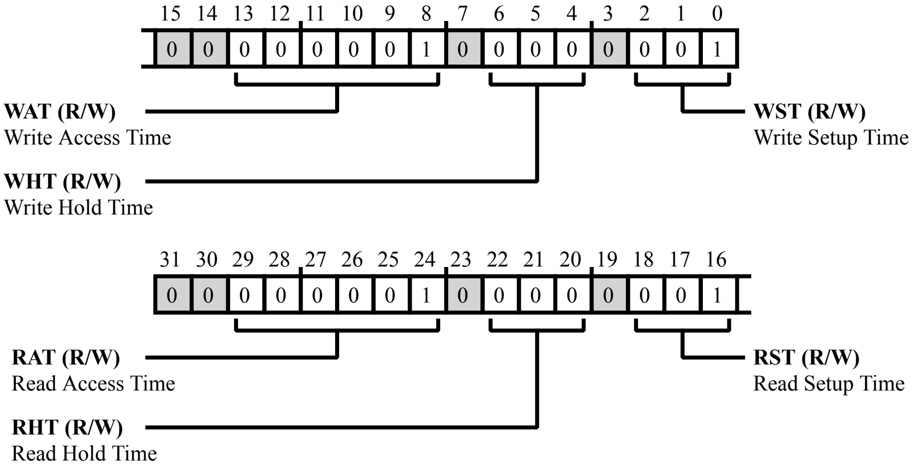

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
WAT (R/W)
Write Access Time
WHT (R/W)
Write Hold Time
WST (R/W)
Write Setup Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
RAT (R/W)
Read Access Time
RHT (R/W)
Read Hold Time
RST (R/W)
Read Setup Time

Figure 11-18: SMC_B1TIM Register Diagram

Table 11-8: SMC_B1TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The `SMC_B1TIM.RAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_ARE` pin for a read access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The `SMC_B1TIM.RHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_ARE` pin before asserting the `SMC_AOE` pin for the next access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Table 11-8: SMC_B1TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The `SMC_B1TIM.RST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_ARE` pin for an access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The `SMC_B1TIM.WAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AWE` pin for a write access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The `SMC_B1TIM.WHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AWE` pin before de-asserting the `SMC_AOE` pin for the current access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The `SMC_B1TIM.WST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_AWE` pin for a write access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Bank 2 Control Register

The `SMC_B2CTL` register enables bank 2 accesses and configures the memory access features for this bank.

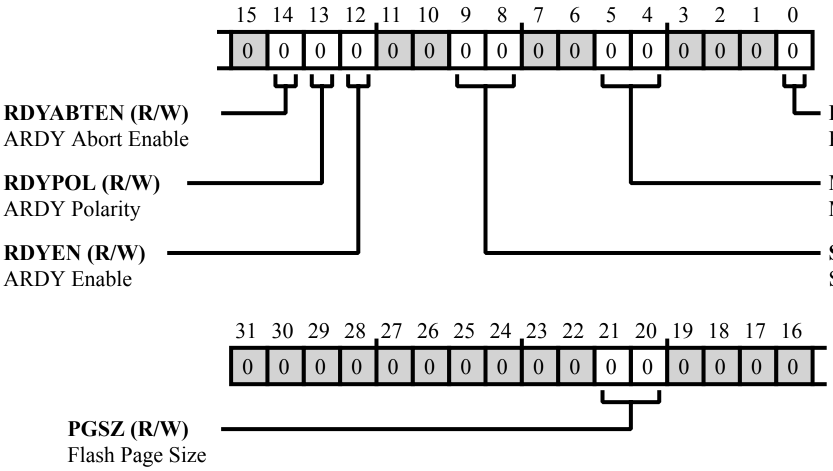

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDYABTEN (R/W)
ARDY Abort Enable
RDYPOL (R/W)
ARDY Polarity
RDYEN (R/W)
ARDY Enable
EN (R/W)
Bank 2 Enable
MODE (R/W)
Memory Access Mode
SELCTRL (R/W)
Select Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PGSZ (R/W)
Flash Page Size

Figure 11-19: SMC_B2CTL Register Diagram

Table 11-9: SMC_B2CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The `SMC_B2CTL.PGSZ` bits select the flash page size, if page flash or sync burst flash protocol has been enabled (`SMC_B2CTL.MODE` > 1). Note that the `SMC_B2CTL.PGSZ` bits must be set to match the flash protocol of the external flash memory device in the system. The typical `SMC_B2CTL.PGSZ` selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical `SMC_B2CTL.PGSZ` selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 11-9: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The `SMC_B2CTL.RDYABTEN` bit enables the abort counter for the `SMC_ARDY` pin, if enabled (`SMC_B2CTL.RDYEN` =1). After `SMC_B2TIM.RAT` or `SMC_B2TIM.WAT` cycles, the SMC starts sampling the `SMC_ARDY` pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK0_0. If the SMC detects that `SMC_ARDY` remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The `SMC_B2CTL.RDYPOL` bit selects the polarity (active high or low) for the `SMC_ARDY` pin, if enabled (`SMC_B2CTL.RDYEN` =1). When the SMC samples the `SMC_ARDY` pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The `SMC_B2CTL.RDYEN` bit enables `SMC_ARDY` pin operation for bank 2 accesses. When enabled, the SMC uses `SMC_ARDY` (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores `SMC_ARDY` for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The `SMC_B2CTL.SELCTRL` bits select the handling of the `SMC_AMS[n]`, `SMC_ARE`, `SMC_AOE`, and `SMC_AWE` pins for memory access control.	
		0	AMS2 only
		1	AMS2 ORed with ARE
		2	AMS2 ORed with AOE
		3	AMS2 ORed with AWE

Table 11-9: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The `SMC_B2CTL.MODE` bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 2 Enable. The `SMC_B2CTL.EN` bit enables accesses to the memory in bank 2. When this bit is disabled, accesses to bank 2 return an error response.	
		0	Disable access
		1	Enable access

Bank 2 Extended Timing Register

The `SMC_B2ETIM` register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the `SMC_B2TIM` register.

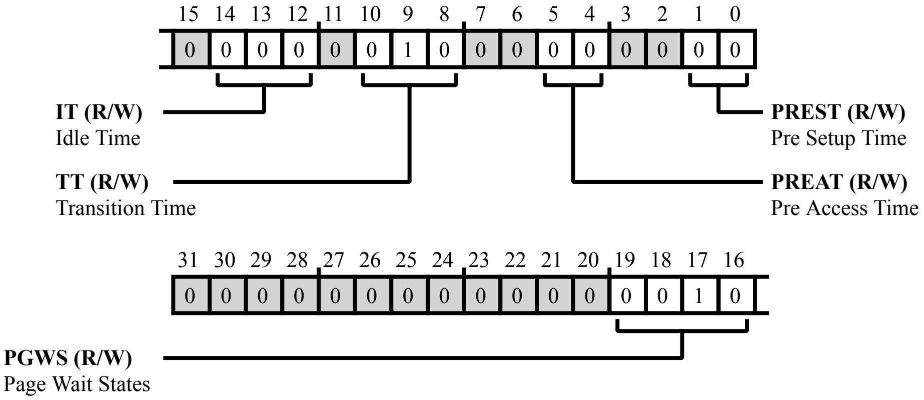

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
IT (R/W)
Idle Time
TT (R/W)
Transition Time
PREST (R/W)
Pre Setup Time
PREAT (R/W)
Pre Access Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PGWS (R/W)
Page Wait States

Figure 11-20: SMC_B2ETIM Register Diagram

Table 11-10: SMC_B2ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The `SMC_B2ETIM.PGWS` bits select a page access extension time (in SCLK0_0 cycles) that the SMC waits during read accesses when configured for flash page protocol (`SMC_B2CTL.MODE` =2). The wait time is from 2 to 15 SCLK0_0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0_0 clock cycles
14:12 (R/W)	IT	Idle Time. The `SMC_B2ETIM.IT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC waits between de-asserting the `SMC_AMS[n]` pin and asserting the `SMC_AMS[n]` pin for the next access. Note that the `SMC_B2ETIM.IT` period may be extended using the `SMC_B2ETIM.TT` selection. The idle time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		7	7 SCLK0_0 clock cycles

Table 11-10: SMC_B2ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The `SMC_B2ETIM.TT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC extends the `SMC_B2ETIM.IT` to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK0_0 cycles.	
		0	No bank transition
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The `SMC_B2ETIM.PREAT` bits select the pre-access time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AOE`/ADV pin before asserting the `SMC_ARE`/`SMC_AWE` pin for the current access. The pre-access time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The `SMC_B2ETIM.PREST` bits select the pre-setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AMS[n]` pin before asserting the `SMC_AOE`/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles

Bank 2 Timing Register

The `SMC_B2TIM` register configures bank 2 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

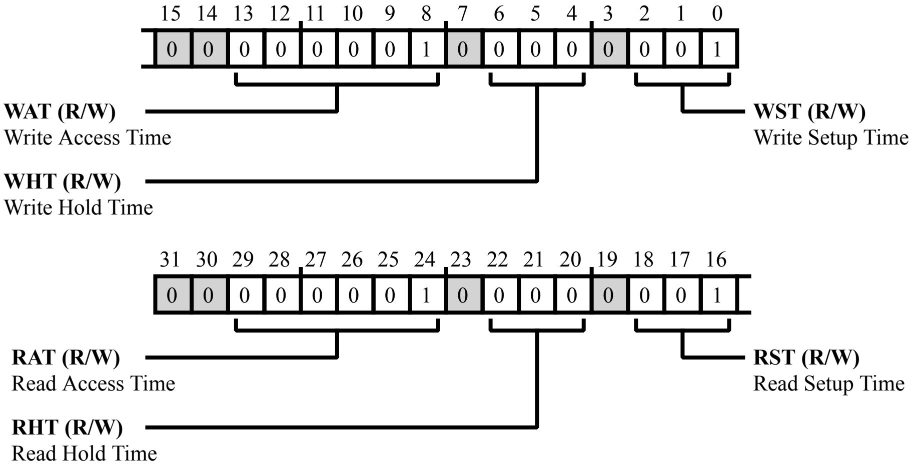

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
WAT (R/W)
Write Access Time
WHT (R/W)
Write Hold Time
WST (R/W)
Write Setup Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
RAT (R/W)
Read Access Time
RHT (R/W)
Read Hold Time
RST (R/W)
Read Setup Time

Figure 11-21: SMC_B2TIM Register Diagram

Table 11-11: SMC_B2TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The `SMC_B2TIM.RAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_ARE` pin for a read access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The `SMC_B2TIM.RHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_ARE` pin before asserting the `SMC_AOE` pin for the next access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Table 11-11: SMC_B2TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The `SMC_B2TIM.RST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_ARE` pin for an access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The `SMC_B2TIM.WAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AWE` pin for a write access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The `SMC_B2TIM.WHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AWE` pin before de-asserting the `SMC_AOE` pin for the current access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The `SMC_B2TIM.WST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_AWE` pin for a write access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Bank 3 Control Register

The `SMC_B3CTL` register enables bank 3 accesses and configures the memory access features for this bank.

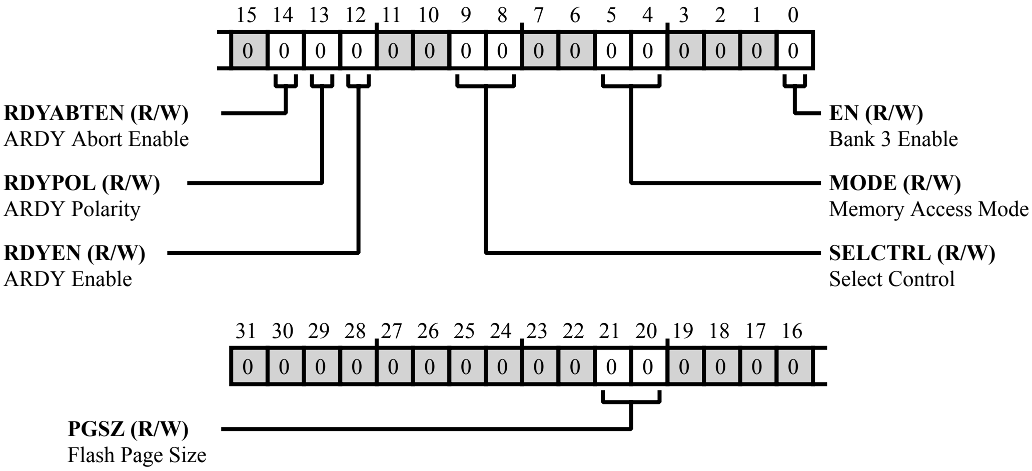

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDYABTEN (R/W)
ARDY Abort Enable
RDYPOL (R/W)
ARDY Polarity
RDYEN (R/W)
ARDY Enable
EN (R/W)
Bank 3 Enable
MODE (R/W)
Memory Access Mode
SELCTRL (R/W)
Select Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PGSZ (R/W)
Flash Page Size

Figure 11-22: SMC_B3CTL Register Diagram

Table 11-12: SMC_B3CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The `SMC_B3CTL.PGSZ` bits select the flash page size, if page flash or sync burst flash protocol has been enabled (`SMC_B3CTL.MODE` > 1). Note that the `SMC_B3CTL.PGSZ` bits must be set to match the flash protocol of the external flash memory device in the system. The typical `SMC_B3CTL.PGSZ` selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical `SMC_B3CTL.PGSZ` selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 11-12: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The `SMC_B3CTL.RDYABTEN` bit enables the abort counter for the `SMC_ARDY` pin, if enabled (`SMC_B3CTL.RDYEN` =1). After `SMC_B3TIM.RAT` or `SMC_B3TIM.WAT` cycles, the SMC starts sampling the `SMC_ARDY` pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK0_0. If the SMC detects that `SMC_ARDY` remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The `SMC_B3CTL.RDYPOL` bit selects the polarity (active high or low) for the `SMC_ARDY` pin, if enabled (`SMC_B3CTL.RDYEN` =1). When the SMC samples the `SMC_ARDY` pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The `SMC_B3CTL.RDYEN` bit enables `SMC_ARDY` pin operation for bank 3 accesses. When enabled, the SMC uses `SMC_ARDY` (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores `SMC_ARDY` for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The `SMC_B3CTL.SELCTRL` bits select the handling of the `SMC_AMS[n]`, `SMC_ARE`, `SMC_AOE`, and `SMC_AWE` pins for memory access control.	
		0	AMS3 only
		1	AMS3 ORed with ARE
		2	AMS3 ORed with AOE
		3	AMS3 ORed with AWE

Table 11-12: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The `SMC_B3CTL.MODE` bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 3 Enable. The `SMC_B3CTL.EN` bit enables accesses to the memory in bank 3. When this bit is disabled, accesses to bank 3 return an error response.	
		0	Disable access
		1	Enable access

Bank 3 Extended Timing Register

The `SMC_B3ETIM` register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the `SMC_B3TIM` register.

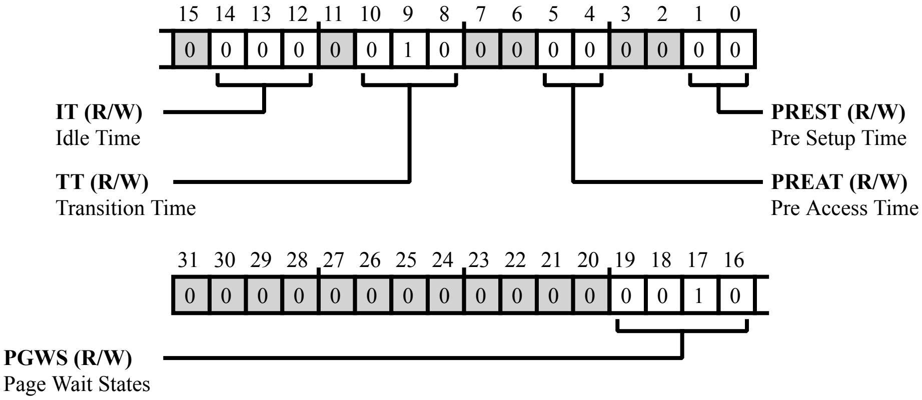

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
IT (R/W)
Idle Time
TT (R/W)
Transition Time
PREST (R/W)
Pre Setup Time
PREAT (R/W)
Pre Access Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PGWS (R/W)
Page Wait States

Figure 11-23: SMC_B3ETIM Register Diagram

Table 11-13: SMC_B3ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The `SMC_B3ETIM.PGWS` bits select a page access extension time (in SCLK0_0 cycles) that the SMC waits during read accesses when configured for flash page protocol (`SMC_B3CTL.MODE` =2). The wait time is from 2 to 15 SCLK0_0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0_0 clock cycles
14:12 (R/W)	IT	Idle Time. The `SMC_B3ETIM.IT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC waits between de-asserting the `SMC_AMS[n]` pin and asserting the `SMC_AMS[n]` pin for the next access. Note that the `SMC_B3ETIM.IT` period may be extended using the `SMC_B3ETIM.TT` selection. The idle time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		7	7 SCLK0_0 clock cycles

Table 11-13: SMC_B3ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The `SMC_B3ETIM.TT` bits select a bus idle time (in SCLK0_0 cycles) that the SMC extends the `SMC_B3ETIM.IT` to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK0_0 cycles.	
		0	No bank transition
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The `SMC_B3ETIM.PREAT` bits select the pre-access time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AOE`/ADV pin before asserting the `SMC_ARE`/`SMC_AWE` pin for the current access. The pre-access time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The `SMC_B3ETIM.PREST` bits select the pre-setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AMS[n]` pin before asserting the `SMC_AOE`/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		3	3 SCLK0_0 clock cycles

Bank 3 Timing Register

The `SMC_B3TIM` register configures bank 3 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

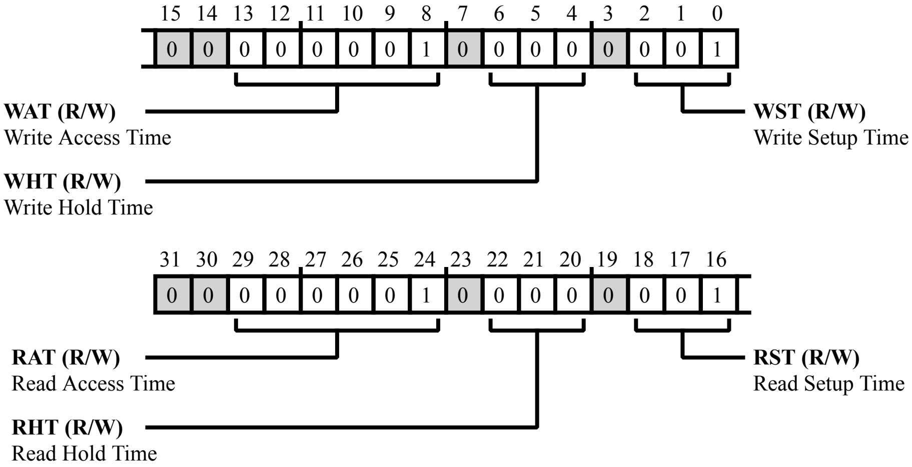

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
WAT (R/W)
Write Access Time
WHT (R/W)
Write Hold Time
WST (R/W)
Write Setup Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
RAT (R/W)
Read Access Time
RHT (R/W)
Read Hold Time
RST (R/W)
Read Setup Time

Figure 11-24: SMC_B3TIM Register Diagram

Table 11-14: SMC_B3TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The `SMC_B3TIM.RAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_ARE` pin for a read access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The `SMC_B3TIM.RHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_ARE` pin before asserting the `SMC_AOE` pin for the next access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

Table 11-14: SMC_B3TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The `SMC_B3TIM.RST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_ARE` pin for an access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The `SMC_B3TIM.WAT` bits select the access time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AWE` pin for a write access. The access time is from 1 to 63 SCLK0_0 cycles.	
		0	Not supported
		1	1 SCLK0_0 clock cycle
		63	63 SCLK0_0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The `SMC_B3TIM.WHT` bits select the hold time (in SCLK0_0 cycles) that the SMC waits after de-asserting the `SMC_AWE` pin before de-asserting the `SMC_AOE` pin for the current access. The hold time is from 0 to 7 SCLK0_0 cycles.	
		0	0 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The `SMC_B3TIM.WST` bits select the setup time (in SCLK0_0 cycles) that the SMC asserts the `SMC_AOE` pin before asserting the `SMC_AWE` pin for a write access. The setup time is from 1 to 8 SCLK0_0 cycles.	
		0	8 SCLK0_0 clock cycles
		1	1 SCLK0_0 clock cycle
		7	7 SCLK0_0 clock cycles

12 One-Time Programmable Memory Controller (OTPC)

This chapter describes the operation of the OTP controller. The OTP module is a complete system integrating an OTP memory core with a programming controller, charge pump, and voltage regulator. A built-in Hamming Code Error Correction (ECC), and a fully implemented double-redundant program or read scheme protect the OTP data.

OTP memory access is through the OTP API Overview provided by the ROM.

CAUTION: OTP memory does not support burst transfers, which are required to support cache line fills. As such, OTP memory should not be made cacheable. If it is, the OTP controller returns an error when a read access is attempted.

OTPC Features

The OTP memory and controller have the following features:

- Built-in redundant read mode
- Built-in integrated power supply
- Built-in Hamming Code Error Correction (ECC)
- Full word serial (single bit at a time) programming with internal VPP

Error Correction

The OTP memory features a Hamming error correction implementation. Signal bit errors are automatically corrected, and dual-bit errors are detected. Refer to OTPC Interrupt Signals.

ECC is always enabled. ECC applies to each 16-bit segment. Because of this functionality, each 16-bit location can only be written to once. Writing to a 16-bit location a second time results in unexpected behavior.

OTP Layout

This section details the memory layout of the OTP memory.

OTP Layout

Table 12-1: ADSP-SCxx OTP Layout

Name	Byte Address	Size (bits)	Description
Customer OTP Area - 896 bytes			
huk	0x0 + 0 - 0x1c	256	Hardware unique key
pvt_128key0	0x168 + 0 - 0x174	128	Customer private key 0 128bits
pvt_128key1	0x178 + 0 - 0x184	128	Customer private key 1 128bits
pvt_128key2	0x188 + 0 - 0x194	128	Customer private key 2 128bits
pvt_128key3	0x198 + 0 - 0x1a4	128	Customer private key 3 128bits
ek	0x1a8 + 0 - 0x1c4	256	Endorsement key
secure_emu_key	0x1c8 + 0 - 0x1d4	128	Secure emulation key
Reserved	0x1d8 + 0 - 0x1fc	320	Reserved
public_key0	0x200 + 0 - 0x23c	512	Customer public key 0
public_key1	0x240 + 0 - 0x27c	512	Customer public key 1
boot_info	0x280 + 0 - 0x2bc	512	Customer programmable boot information
antiroll_nv_cntr	0x2c0 + 0 - 0x2fc	512	AntiRollback NV counter
gp1	0x300 + 0 - 0x33c	512	General purpose 1
Reserved	0x340 + 0 - 0x340	24	Reserved
bootModeDisable	0x343 + 24 - 0x343	8	Boot mode disable bits
preboot_ddr_cfg	0x344 + 0 - 0x370	384	User preboot DDR configuration
stageID	0x374 + 0 - 0x376	48	Stage ID
Reserved	0x37a + 16 - 0x37a	16	Reserved
Reserved	0x37c + 0 - 0x37c	32	Reserved
Overlaid Fields in Customer OTP Area			
otpTiming	0x2be + 16 - 0x2be	16	OTP read timing override
Overlaid Fields in ADI BOOT			
Customer BOOT - 116 Bytes			
lock	0x48c + 0 - 0x48c	1	Lockbit
Reserved	0x48c + 1 - 0x4fc	927	Reserved

OTPC Event Control

The following sections provide information on OTP events and error management.

OTPC Interrupt Signals

When making 32-bit accesses to OTP memory, a double-bit error in any 16-bit segment triggers the OTPC_INT interrupt. The OTPC also has the OTPC dual bit error (OTPC0_ERR) with the SEC ID of 5 and the GIC ID of 37. See the System Event Controller (SEC) and Generic Interrupt Controller (GIC) chapter for more information.

OTPC Status and Error Signals

The OTP controller does not produce error signals.

OTP API Overview

The ROM provides a set of functions to facilitate OTP field access. The OTP memory is broken up into a set of specialized fields that are described in this section. The API removes the requirement of understanding the details of the layout or OTP access procedures.

All OTP accesses are made through the provided API.

OTP Programming

The OTP programming API provides a simple access, abstracting particulars of the OTP controller.

Any fields that contain zero or null pointers are skipped.

All addresses are assumed to be byte addresses unless otherwise noted.

A list of APIs follows:

`bool adi_rom_otp_pgm(otp_data* data);`	OTP Program
`bool adi_rom_lock();`	Lock API

OTP Program

Program OTP memory using a struct containing the following predefined data fields.

Name	OTP Program	-
PP Define	`FUNC_ROM_OTPPGM`	
Prototype	`bool adi_rom_otp_pgm(otp_data* data);`	-
Argument	data	struct containing data to program OTP with
Return Value	bool	true for programming success
Stack Requirements	valid stack	-

```
bool res = adi_rom_otp_pgm(data);
```

The following type of struct is available for programming. Refer to the ROM header file for the exact definition

```
typedef struct {
    uint32_t  (*huk)[8];
    uint32_t  (*pvt_128key1)[4];
    uint32_t  (*pvt_128key2)[4];
    uint32_t  (*pvt_128key3)[4];
    uint32_t  (*pvt_128key4)[4];
    uint32_t  (*pvt_192key1)[6];
    uint32_t  (*pvt_192key2)[6];
    uint32_t  (*public_key0)[16];
    uint32_t  (*public_key1)[16];
    uint32_t  (*ek)[8];
    uint32_t  (*secure_emu_key)[4];
    uint8_t   bootModeDisable;
    uint32_t  (*boot_info)[16];
    uint32_t  (*gp0)[16];
    uint32_t antiroll_nv_cntr;
    uint8_t   stageID;
    uint32_t  (*preboot_ddr_cfg)[11];
} otp_data;
```

NOTE: Make OTP memory a non-cacheable region if the core needs access to it.

OTP Reading

This API provides a unified source for retrieving OTP data fields.

All addresses are assumed to be byte addresses, unless otherwise noted.

A list of APIs follow:

`bool adi_rom_otp_get( OTPCMD cmd, uint32_t* data);`	OTP Get Field

OTP Get Field

Retrieves indicated data from OTP memory.

Name	OTP Get Field	
Prototype	`bool adi_rom_otp_get( OTPCMD cmd, uint32_t* data);`	
Argument	cmd	Indicates what data to fetch, based on the`OTPCMD` enum.
Argument	data	memory location to write the data to
Return Value	bool	true for a successful read
Stack Requirements	valid stack	

```
bool res = adi_rom_otp_get(otpcmd_info,data);
```

The data specified by the `OTPCMD` enum parameter is fetched from OTP memory and placed in the location specified by data. The `OTPCMD` enum contains entries for each field defined in OTP memory, for the most current list please refer to the OTP header file.

An example of the enum style follows:

```
                /*  Field Name               Description                  No of 
Bits */
                typedef enum {
                  /* add msi bits */
                  otpcmd_reserved0 = 0,
                  otpcmd_pvt_128key1,      /* Private 128-bit Key 1 
[128] */
                  otpcmd_pvt_128key2,      /* Private 128-bit Key 2 
[128] */
                  otpcmd_pvt_128key3,      /* Private 128-bit Key 3 
[128] */
                  otpcmd_pvt_128key4,      /* Private 128-bit Key 4 
[128] */
                  otpcmd_pvt_192key1,      /* Private 192-bit Key 1 
[192] */
                  otpcmd_pvt_192key2,      /* Private 192-bit Key 2 
[192] */
                  otpcmd_huk,              /* Hardware Unique Key 
[256] */
                  otpcmd_ek,               /* Endorsement Key (EK) 
[256] */
                  otpcmd_secure_emu_key,   /* Secure Emulation Key 
[128] */
                  otpcmd_public_key1,      /* Customer Public Key 1 
[512] */
                  otpcmd_public_key2,      /* Customer Public Key 2 
[512] */
                  otpcmd_antiroll_nv_cntr, /* Anti-Rollback NV Counter 
[32] */
                  otpcmd_nonvolatile_cntr, /* NV Counter 
[32] */
                  otpcmd_bootModeDisable,  /* Boot Mode disable 
field              [8] */
                  otpcmd_stageID,          /* Stage 
ID                            [8] */
                  otpcmd_gp0,              /* General Purpose 
[512] */
                  otpcmd_boot_info,        /* Customer Programmable Boot info 
[384] */
                  otpcmd_preboot_ddr_cfg,  /* User Preboot DDR configuration 
[352] */
                  otpcmd_reserved1        /* invalid */
                } OTPCMD;
```

OTP Counters

The OTPC module implements a counter API to allow easy reading or writing of the counter without dealing with the complexities of rewriting OTP memory sections that are ECC protected.

The OTPC module provides two functional APIs for counters. These APIs are not extra; the module uses the same `get` and `pgm` APIs. The APIs are functionally unique in the way that they set and retrieve data as counters in OTP memory.

The API uses a different method to count bits because each bit in OTP memory can only be set =1 once, and the ECC protects each 16-bit unit. This functionality essentially means that each 16-bit unit can only be written to once. Therefore, a counter that can count 0–31 requires 32 × 16 bits of memory.

The API receives and returns the value of the counter as a `uint8_t` binary number. Writing a value less than the current value of the counter or greater than the maximum value results in an error.

To implement this functionality, the driver counts by shifting 1's from the left, treating each block as 1 bit. A three-bit counter is encoded as follows.

bit 2	bit 1	bit 0	Value
0000	0000	0000	0
0001	0000	0000	1
0001	0001	0000	2
0001	0001	0001	3

Lock API

This API locks the device.

Name	Lock API	-
PP Define	`FUNC_ROM_LOCK`	-
Prototype	`bool adi_rom_lock();`	-
Return Value	bool	true for success
Stack Requirements	valid stack	-

```
bool res = adi_rom_lock();
```

Calling this function locks the device, making it a secure. Once locked, the `OTPC_SECU_STATE` register indicates that the part is locked, and access is limited. For more information, refer to the security documentation regarding a locked device.

NOTE: Locked Status. The `OTPC_SECU_STATE` register is updated only after the part is rebooted. After calling the lock function, the register still indicates that the part is open.

ADSP-SC58x OTPC Interrupt List

Table 12-2: ADSP-SC58x OTPC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
5	OTPC0_ERR	OTPC0 Dual-bit error	Level	

ADSP-SC58x OTPC Register Descriptions

OTP Memory Controller (OTPC) contains the following registers.

Table 12-3: ADSP-SC58x OTPC Register List

Name	Description
`OTPC_SECU_STATE`	OTP Security State Register
`OTPC_STAT`	OTP Status Register

OTP Security State Register

The `OTPC_SECU_STATE` register provides lock status information.

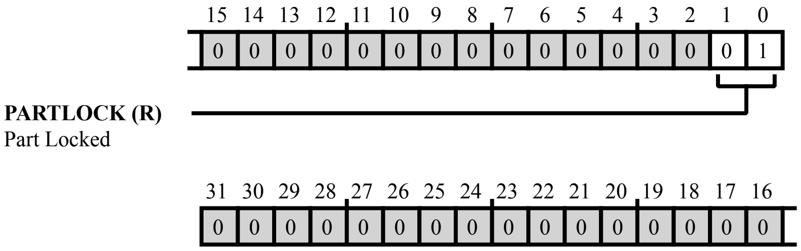

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
PARTLOCK (R)
Part Locked
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 12-1: OTPC_SECU_STATE Register Diagram

Table 12-4: OTPC_SECU_STATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	PARTLOCK	Part Locked. The `OTPC_SECU_STATE.PARTLOCK` bit indicates a locked part.	
		0	OPEN part
		1	Locked part
		2	Unlocked part

OTP Status Register

The `OTPC_STAT` register bits indicate errors and flag status and control the protection bits.

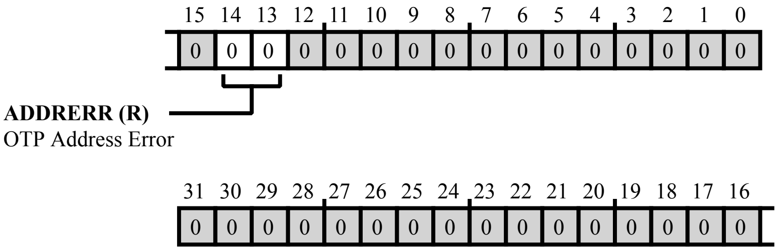

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRERR (R)
OTP Address Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 12-2: OTPC_STAT Register Diagram

Table 12-5: OTPC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14:13 (R/NW)	ADDRERR	OTP Address Error. The `OTPC_STAT.ADDRERR` bit field indicates errors which occur when the OTP programming address is out of range or tries to access protected space.	
		0	No error - proper OTP address
		1	OTP address out of range
		2	8-bit OTP address
		3	Protected OTP address

13 System Memory Protection Unit (SMPU)

The SMPU provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

SMPU Features

The system memory protection unit has the following features.

- After reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces by any master.
- Each SMPU instance can be configured to monitor multiple regions. Each can be individually enabled.
- Each region can be configured with its own protection settings.
- Provides general read or write protection.
- Read and write transactions are restricted or allowed depending on the transaction ID.

On the ADSP-SC58x, ten SMPU instances are available to protect the L2, external memory (DMC/SMC), and memory mapped I/O (PCIe) interfaces. Six instances are allotted to protect the L2 memory, two instances for DMC0 and DMC1, one instance for SMC and one instance for PCIe as shown below:

Table 13-1: SMPU Instances

Module	SMPU Instance
SMC	0
Core_L2_RAM_Boot_ROM0	2
DMA_L2_RAM_Boot_ROM0	3
Core_L2_ROM1_Boot_ROM1	4
DMA_L2_ROM1_Boot_ROM1	5
Core_L2_ROM2_Boot_ROM2	6
DMA_L2_ROM2_Boot_ROM2	7
PCIe	8

Table 13-1: SMPU Instances (Continued)

Module	SMPU Instance
DMC0	9
DMC1	10

All the SMPU instances can be configured up to eight regions.

- Up to eight outstanding read or write transactions supported on SMPU instances for Core_L2_RAM_Boot_ROM0, DMC0 and DMC1.
- Up to four outstanding read or write transactions supported on the SMPU instances for DMA_L2_RAM_Boot_ROM0, Core_L2_ROM1_Boot_ROM1, DMA_L2_ROM1_Boot_ROM1, Core_L2_ROM2_Boot_ROM2, DMA_L2_ROM2_Boot_ROM2, SMC, and PCIe.
- Exclusive access enabled by hardware for the SMPU instances of Core_L2_RAM_Boot_ROM0, DMC 0, DMC1, and SMC. Up to four exclusive-access-capable masters can be supported.

SMPU Functional Description

The following sections provide details on the function of the SMPU module. If the region security settings allow transactions to go through, the ID in the ID-based region protection settings can still filter the transactions.

For the memory that an SMPU protects, programs can configure region-based settings with the `SMPU_RCTL[n]` registers. (There can be multiple SMPUs in a system.) The `SMPU_RCTL[n]` registers define the ID-based protection for memory regions.

If the target address does not reside in any configured memory region, the transaction permission resorts back to the global configuration setting.

Protection Units

Each SMPU provides two protection units, A and B for ID-based matching in the region-based memory protection. This feature provides a degree of flexibility for the user to match against multiple IDs.

Instruction Fetches

When the core executes instructions from memory, this operation is also considered a memory transaction. If the SMPU is configured to protect a memory region from read accesses that contain instructions, the core cannot fetch and execute these instructions.

Using Cache

When the processor uses both cache and the SMPU, there are a few issues to be aware of. If the SMPU is configured to protect a memory region from write accesses, instruction fetches from a core are still possible since instructions are not updated and replaced during run time.

NOTE: The debugger typically replaces an instruction with a breakpoint instruction for software breakpoints. If a memory region is protected against write accesses, software breakpoints are not possible unless the SMPU is configured with the appropriate system master ID of the debugger. The configuration allows it to perform a write-access.

In the case where memory is used for data, a read access or cache fill is not possible if the memory is blocked from read accesses. If read accesses are allowed but write accesses are disallowed, then there is an issue with coherency. The cache is filled but when the cache is updated and must be written back to the SMPU protected memory, the write-access is blocked.

In general, practice caution when using both the SMPU and cache.

Speculative Reads

If speculative reads are enabled (`SMPU_CTL.RSDIS` =0), the SMPU forwards the read transaction directly to the memory before checking the protection setting corresponding to the addressed memory region. This functionality saves one clock cycle in the clock domain of the SMPU. The SMPU checks the protection setting while the read transaction occurs with the memory. If the protection setting dictates that the target memory address is blocked, the SMPU blocks the read to the master.

If speculative reads are disabled (`SMPU_CTL.RSDIS` =1), the SMPU checks the protection settings first and forwards the transaction to memory only if it passes the configured protection settings. This functionality incurs a one-cycle latency per read.

NOTE: Reads affect certain memory operations such as automatic clearing of the memory (that is, FIFOs). When the SMPU protects this type of memory, disable read speculation since the blocking can occur without the read transaction reaching the target memory.

ADSP-SC58x SMPU Register List

The System Memory Protection Unit (SMPU) provides selective protection of the processor's memory resources. The SMPU includes a set of processor events that can be monitored during program execution. A set of registers governs SMPU operations. For more information on SMPU functionality, see the SMPU register descriptions.

Table 13-2: ADSP-SC58x SMPU Register List

Name	Description
`SMPU_BADDR`	Bus Error Address Register
`SMPU_BDTLS`	Bus Error Details Register
`SMPU_CTL`	SMPU Control Register
`SMPU_EXACADD[n]`	Exclusive Access IDn Address
`SMPU_EXACSTAT[n]`	Exclusive Access Status
`SMPU_IADDR`	Interrupt Address Register

Table 13-2: ADSP-SC58x SMPU Register List (Continued)

Name	Description
`SMPU_IDTLS`	Interrupt Details Register
`SMPU_RADDR[n]`	Region n Address Register
`SMPU_RCTL[n]`	Region n Control Register
`SMPU_REVID`	SMPU Revision ID Register
`SMPU_RIDA[n]`	Region n ID A Register
`SMPU_RIDB[n]`	Region n ID B Register
`SMPU_RIDMSKA[n]`	Region n ID Mask A Register
`SMPU_RIDMSKB[n]`	Region n ID Mask B Register
`SMPU_SECURECTL`	SMPU Control Secure Accesses Register
`SMPU_SECURERCTL[n]`	Region n Control Secure Accesses Register
`SMPU_STAT`	SMPU Status Register

ADSP-SC58x SMPU Interrupts

The SMPU has one interrupt with the SEC ID = 242. See the System Event Controller (SEC) and Generic Interrupt Controller (GIC) chapter for complete information on interrupt generation and use.

Memory Writes

A write transaction to address n is prevented when the following is true.

Address n is in memory region m and memory region m is write-protected (`SMPU_RCTL[n].WPROTEN` =1) and ID is not a match. (See ID Comparison). The block occurs because the memory region is configured for write-protection and the ID comparison does not result in a match. If an ID comparison results in a match, the write transaction is allowed through.

Memory Reads

A read transaction from address n is prevented when the following is true:

- Address n is in memory region and memory region m is read-protected (`SMPU_RCTL[n].RPROTEN` =1) and
- ID is not a match

The block occurs because the memory region is configured for read-protection and the ID comparison does not result in a match. If the ID comparison results in a match, the read transaction is permitted. (See ID Comparison).

ID Comparison

ID comparison automatically occurs during region-based memory protection. ID matches allow the transaction to bypass the configured memory protection for that region. The following sections describe the calculation of a write ID match and read ID match.

Write Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming write transaction:

- The `SMPU_RCTL[n].WIDCINV` bit
- The `SMPU_RIDA[n].ID` and `SMPU_RIDB[n].ID` bit fields
- The `SMPU_RIDMSKA[n].MSK` and `SMPU_RIDMSKB[n].MSK` bit fields

Write IDA match = ((ID of incoming write transaction AND `SMPU_RIDMSKA[n].MSK`) == (`SMPU_RIDA[n].ID` AND `SMPU_RIDMSKA[n].MSK`))

Write IDB match = ((ID of incoming write transaction AND `SMPU_RIDMSKB[n].MSK`) == (`SMPU_RIDB[n].ID` AND `SMPU_RIDMSKB[n].MSK`))

Write ID match = (Write IDA match OR Write IDB match) XOR `SMPU_RCTL[n].WIDCINV` bit

Read Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming read transaction:

- The `SMPU_RCTL[n].RIDCINV` bit
- The `SMPU_RIDA[n].ID` and `SMPU_RIDB[n].ID` bit fields
- The `SMPU_RIDMSKA[n].MSK` and `SMPU_RIDMSKB[n].MSK` bit fields

Read IDA match = ((ID of incoming read transaction AND `SMPU_RIDMSKA[n].MSK`) == (`SMPU_RIDA[n].ID` AND `SMPU_RIDMSKA[n].MSK`))

Read IDB match = ((ID of incoming read transaction AND `SMPU_RIDMSKB[n].MSK`) == (`SMPU_RIDB[n].ID` AND `SMPU_RIDMSKB[n].MSK`))

Read ID match = (Read IDA match OR Read IDB match) XOR `SMPU_RCTL[n].RIDCINV`

In the two cases described above, the incoming transaction (either write or read) ID is AND'ed with the configured mask value in protection unit A. It is then compared to the value of the configured ID value which is also AND'ed with the configured mask value in protection unit A. The mask provides a method to allow a group of IDs to match. This process is also performed for protection unit B. The two outcomes (from A and B) are then OR'ed together.

Depending on the setting of the `SMPU_RCTL[n].RIDCINV` or the `SMPU_RCTL[n].WIDCINV` bits, the ID match comparison is inverted or not. The final result after applying the inversion, `SMPU_RCTL[n].RIDCINV`, or `SMPU_RCTL[n].WIDCINV`, determines whether the transaction bypasses the protection.

Usage

The masks, `SMPU_RIDMSKA[n]` and `SMPU_RIDMSKB[n]`, are AND'ed with both the incoming transaction ID and the configured ID in `SMPU_RIDA[n].ID` and `SMPU_RIDB[n].ID`, respectively. By default the masks are zero. If ID-based region protection is enabled by setting the `SMPU_RCTL[n].WPROTEN` or `SMPU_RCTL[n].RPROTEN` bit fields and the masks are not set, the ID comparison essentially compares zeros. The comparison allows all transactions to bypass (if the region-based security setting is also configured in a way to allow transactions to go through for the region). To have the ID-based region protection to function, the mask registers and ID registers must also be set.

System IDs

The ***System Master IDs*** table provides the IDs for the system masters. An x means that the bit can be a 0 or a 1. There are multiple IDs associated with that particular system master.

Table 13-3: System Master IDs

Master Name	SCB Input Switch	Master ID SCB Input	ID
SPORT0_A	0	0	13'b00000000x0000
SPORT0_B	0	1	13'b00000000x0001
SPORT1_A	0	2	13'b00000000x0010
SPORT1_B	0	3	13'b00000000x0011
SPORT2_A	0	4	13'b00000000x0100
SPORT2_B	0	5	13'b00000000x0101
SPORT3_A	0	6	13'b00000000x0110
SPORT3_B	0	7	13'b00000000x0111
SPORT4_A	1	0	13'b00010000x0000
SPORT4_B	1	1	13'b00010000x0001
SPORT5_A	1	2	13'b00010000x0010
SPORT5_B	1	3	13'b00010000x0011
SPORT6_A	1	4	13'b00010000x0100
SPORT6_B	1	5	13'b00010000x0101
SPORT7_A	1	6	13'b00010000x0110
SPORT7_B	1	7	13'b00010000x0111
CRC0_CH0	0	8	13'b00000000x1000

Table 13-3: System Master IDs (Continued)

Master Name	SCB Input Switch	Master ID SCB Input	ID
CRC0_CH1	0	9	13'b00000000x1001
CRC1_CH0	1	8	13'b00010000x1000
CRC1_CH1	1	9	13'b00010000x1001
EMAC0	2	0	13'b00100xxxx0000
MLB	3	0	13'b0011000000000
UART0_TX	2	1	13'b00100000x0001
UART0_RX	2	2	13'b00100000x0010
SDIO	2	3	13'b0010000000011
SINC	2	4	13'b00100000x0100
SPI0TX	4	0	13'b01000000x0000
SPI0RX	4	1	13'b01000000x0001
SPI1TX	4	2	13'b01000000x0010
SPI1RX	4	3	13'b01000000x0011
SPI2TX	4	4	13'b01000000x0100
SPI2RX	4	5	13'b01000000x0101
PPI_F0	4	6	13'b01000000x0110
PPI_F1	4	7	13'b01000000x0111
LP0	3	1	13'b00110000x0001
HAE_IN0	5	0	13'b01010000x0000
HAE_IN1	5	1	13'b01010000x0001
HAE_OUT	5	2	13'b01010000x0010
USB0	2	5	13'b0010000000101
UART1_TX	5	3	13'b01010000x0011
UART1_RX	5	4	13'b01010000x0100
EMAC1	5	5	13'b01010xxxx0101
LP1	3	2	13'b00110000x0010
USB1	6	0	13'b0110000000000
CRYPTO	6	3	13'b0110000000011
UART2_TX	6	4	13'b01100000x0100
UART2_RX	6	5	13'b01100000x0101
FIR_CH0	5	6	13'b0101000000110

Table 13-3: System Master IDs (Continued)

Master Name	SCB Input Switch	Master ID SCB Input	ID
FIR_CH1	5	7	13'b0101000000111
IIR_CH0	6	6	13'b0110000000110
IIR_CH1	6	7	13'b0110000000111
EMDMA0_CH0	7	0	13'b0111000000000
EMDMA0_CH1	7	1	13'b0111000000001
EMDMA1_CH0	7	2	13'b0111000000010
EMDMA1_CH1	7	3	13'b0111000000011
STD_BW_MDMA_SRC_CH	8	0	13'b10000000x0000
STD_BW_MDMA_DST_CH	8	1	13'b10000000x0001
PCIE_M	8	2	13'b1000xxxxx0010
ENH_BW_MDMA_SRC_CH	9	0	13'b10010000x0000
ENH_BW_MDMA_DST_CH	9	1	13'b10010000x0001
FFT_CH0	9	2	13'b10010000x0010
FFT_CH1	9	3	13'b10010000x0011
SH0_DPORT	10	0	13'b1010000000000
SH0_IPORT	10	1	13'b1010000000001
SH1_DPORT	10	2	13'b1010000000010
SH1_IPORT	10	3	13'b1010000000011
PL310_M0 = L2 cache Master Port 0	10	5	13'b1010xxxxx0101
PL310_M0 = L2 cache Master Port 1	10	4	13'b1010xxxxx0100
DBG	3	3	13'b0011000000011
ETR	3	4	13'b0011000000100

Memory Region

Memory regions can start at address 0x00000000 or at any address that is a multiple of its size. The ***Supported Memory Region Size and Alignment*** table shows the memory region sizes that the processor supports and the alignment of the memory region. (X values are do-not-care).

SMPU0 and SMPU2 support a maximum of four regions. SMPU1 supports a maximum of eight regions.

Table 13-4: Supported Memory Region Size and Alignment

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
	0b00000	0xXXXXX000	-

Table 13-4: Supported Memory Region Size and Alignment (Continued)

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
4KB			
8KB	0b00001	0xXXXXN000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
16KB	0b00010	0xXXXXN000	0x0, 0x4, 0x8, 0xC
32KB	0b00011	0xXXXXN000	0x0, 0x8
64KB	0b00100	0xXXXX0000	-
128KB	0b00101	0xXXXN0000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
256KB	0b00110	0xXXXN0000	0x0, 0x4, 0x8, 0xC
512KB	0b00111	0xXXXN0000	0x0, 0x8
1MB	0b01000	0xXXX00000	-
2MB	0b01001	0xXXN00000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
4MB	0b01010	0xXXN00000	0x0, 0x4, 0x8, 0xC
8MB	0b01011	0xXXN00000	0x0, 0x8
16MB	0b01100	0xXX000000	-
32MB	0b01101	0xXN000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
64MB	0b01110	0xXN000000	0x0, 0x4, 0x8, 0xC
128MB	0b01111	0xXN000000	0x0, 0x8
256MB	0b10000	0xX0000000	-
512MB	0b10001	0xN0000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
1GB	0b10010	0xN0000000	0x0, 0x4, 0x8, 0xC
2GB	0b10011	0xN0000000	0x0, 0x8
4GB	0b10100	0x00000000	-

For the case where the region size is selected as 4 GB, the region address must be at address 0x00000000.

NOTE: If a memory region address is not aligned to its size, the memory region start address protected by the SMPU is the configured address with the corresponding least significant bits masked. For example, if the size is configured for 16 KB (`SMPU_RCTL[n].SIZE` =0b00010), and the base address is configured for `SMPU_RADDR[n].BADDR`=0x00005018, the actual base address used by the SMPU is 0x00004000. When `SMPU_RADDR[n].BADDR` is read back, the program reads 0x00005000. This functionality is because only bits [11:0] are reserved as 0's. Programs must use care when setting the base address as it is not always the true base address.

SMPU Definitions

To make the best use of the SMPU, it is useful to understand the terms in this section.

Global Protection

Guarding of the entire memory space for the particular SMPU instantiation.

Region-Based Protection

Guarding individual segments of memory inside the memory space for the particular SMPU instantiation.

ID Match

A successful comparison of the ID associated with the incoming transaction and the ID and MASK configured in the SMPU.

SMPU Block Diagram

The ***SMPU Top-Level Block Diagram*** shows the SMPU block.

As seen in the diagram, the SMPU sits between the memory port (SCB master port) and the SCB fabric (SCB slave port). It acts as a gateway analyzing the transaction requests. It either rejects the transaction request or allows access based on the user-programmed configuration of the SMPU.

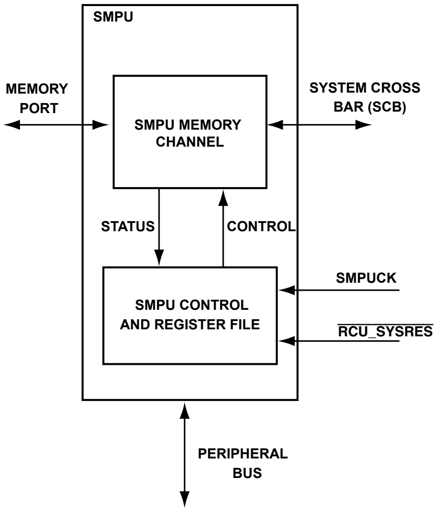

SMPU
MEMORY PORT
SMPU MEMORY CHANNEL
SYSTEM CROSS BAR (SCB)
STATUS
CONTROL
SMPU CONTROL AND REGISTER FILE
SMPUCK
$\overline{\text{RCU\_SYSRES}}$
PERIPHERAL BUS

Figure 13-1: SMPU Top-Level Block Diagram

SMPU Architectural Concepts

The following sections provide brief descriptions of the architecture of the SMPU module.

Default Setting

At reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces.

Latency

The SMPU adds latency to all the transactions to the memory except reads when read speculation is enabled (`SMPU_CTL.RSDIS` =0). In this case, read accesses are always forwarded to the memory and read responses are generated according to the SMPU settings. If read speculation is disabled (`SMPU_CTL.RSDIS` =1), reads are blocked if they cause a security or protection violation. The SMPU generates the SCB read response that corresponds to a blocked transaction.

If read speculation is enabled, the SMPU adds 1 clock cycle latency to the read transaction. If read speculation is disabled, the SMPU adds 2 clock cycles latency to the read transaction.

SMPU Operating Modes

The SMPU does not have any strict modes of operation. However, it can be configured for region-based protection where a master with a particular ID can be blocked or allowed based on settings in the `SMPU_RCTL[n]` register.

Region-based protection is programmed with registers:

- `SMPU_RCTL[n]`
- `SMPU_RADDR[n]`
- `SMPU_RIDA[n]`
- `SMPU_RIDMSKA[n]`
- `SMPU_RIDB[n]`
- `SMPU_RIDMSKB[n]`

Exclusive Accesses. On the ADSP-SC5xx processors, the hardware supports the exclusive accesses. The exclusive accesses are supported for the SMPU instances that correspond to SMC, DMC 0, DMC 1 and Core_L2_0. The exclusive accesses are automatically enabled when one of the cores (ARM Core 0, SHARC+ Core 1, and SHARC+ Core 2) executes the exclusive access instruction. Refer to the Programming Reference manual for more details on how the exclusive access works.

SMPU Interrupt Signals

There is one interrupt signal associated with the SMPU. If interrupts are enabled, the `SMPU_STAT.IRQ` bit is set. The SMPU_IRQ signal is asserted when the SMPU detects a memory access violation. The target address triggering the interrupt is found in the `SMPU_IADDR` register. The `SMPU_IDTLS` register provides further details about the cause of the interrupt.

Write errors are prioritized over read errors.

Protection violations (an ID-based violation) can trigger the SMPU interrupt, and can be enabled independently. The protection violation interrupt is enabled by setting the `SMPU_CTL.PINTEN` bit.

The SMPU interrupt is asserted for any of the following conditions:

If a second memory access violation occurs while the `SMPU_STAT.IRQ` bit is set, the `SMPU_STAT.IOVR` (interrupt overrun) bit is set. The `SMPU_IADDR` and the `SMPU_IDTLS` registers are not updated until the `SMPU_STAT.IRQ` bit is cleared. Any information on the subsequent interrupt is lost. Once the `SMPU_STAT.IRQ` bit and the `SMPU_STAT.IOVR` bit are cleared, any new memory access violations can trigger an interrupt and its details can be captured.

NOTE: When a blocked access occurs, the SMPU triggers an interrupt when interrupt generation is enabled. The SMPU can also be configured to generate a bus error that propagates back to the system master. The system master can also trigger an interrupt due to this bus error.

NOTE: On the processor, each SMPU instance has an interrupt. All of the SMPU interrupts are OR'ed and mapped to a single SMPU interrupt on the SEC/GIC. While servicing the SMPU interrupt, check all of the `SMPU_STAT` registers to determine which triggered the interrupt. The interrupt service routine clears the `SMPU_STAT.IRQ` bit of the all of the `SMPU_STAT` registers for which the interrupt is triggered.

SMPU Status and Error Signals

If bus errors are enabled (`SMPU_CTL.PBEDIS` =0), the SMPU generates and returns a bus error to the master initiating the blocked access. This bit also sets the `SMPU_STAT.BERR` bit. The `SMPU_BADDR` and `SMPU_BDTLS` registers can be read to get the address and details of the transaction that caused the SMPU to generate the error.

Write errors are prioritized over read errors.

A bus error status is returned to the system master if:

- an ID-based violation happened and the `SMPU_CTL.PBEDIS` bit =0

If a second memory access violation occurs while the `SMPU_STAT.BERR` bit is set, the `SMPU_STAT.BEOVR` bit (bus error overrun) is set. The `SMPU_BADDR` and the `SMPU_BDTLS` registers are not updated until the `SMPU_STAT.IRQ` bit is cleared. The information about the transaction that caused the `SMPU_STAT.BEOVR` bit to be set is lost.

NOTE: If both the protection violation interrupt is not enabled (`SMPU_CTL.PINTEN` =0) and the protection bus error is disabled (`SMPU_CTL.PBEDIS` =1), the SMPU blocks invalid transactions. However, it does not provide any status or interrupt information indicating that a transaction is blocked.

SMPU Programming Example

The following example corresponds to using SMPU1 to protect M4 SRAM from MDMA write accesses. In the example the region configured for protection is 4KB at the start of M4 SRAM Bank A.

1. Enable SMPU1 interrupts by writing to the `SMPU_CTL` register.
2. Define the memory region to be protected by writing to the `SMPU_RADDR[n]` register.

```
*pREG_SMPU_RADDR0 = 0x20000000;
```

3. Program the `SMPU_RIDA[n]` and `SMPU_RIDB[n]` registers with the ID which is to be an exception to the rule.

```
*pREG_SMPU_RIDA0 = 0x0081; //corresponds to MDMA_WR
*pREG_SMPU_RIDB0 = 0x0081;
```

4. Program the `SMPU_RIDMSKA[n]` and`SMPU_RIDMSKB[n]` registers with the mask to be applied to the ID values programmed in the `SMPU_RIDA[n]` and `SMPU_RIDB[n]` registers respectively.

```
*pREG_SMPU_RIDMSKA0 = 0xFFFF;
*pREG_SMPU_RIDMSKB0 = 0xFFFF;
```

5. Program the Region Control register (`SMPU_RCTL[n]` register) with the required fields. Control fields like ID invert, write protection enable, read protection enable and memory region size can be configured in this register.

```
*pREG_SMPU_RCTL0 = ENUM_SMPU_RCTL_WPROTEN|ENUM_SMPU_RCTL_RPROTDIS|
BITM_SMPU_RCTL_WIDCINV|BITM_SMPU_RCTL_RIDCINV| ((0<<BITP_SMPU_RCTL_SIZE) &
BITM_SMPU_RCTL_SIZE)
```

6. Set the `SMPU_RCTL[n].EN` bit.

```
*pREG_SMPU_RCTL0  |= BITM_SMPU_RCTL_EN;
```

Any accesses violation to the protection enabled by this procedure will be blocked. If the corresponding interrupt is enabled, an interrupt is generated on protection violations.

ADSP-SC58x SMPU Register Descriptions

The System Memory Protection Unit (SMPU) contains the following registers.

Table 13-5: ADSP-SC58x SMPU Register List

Name	Description
`SMPU_BADDR`	Bus Error Address Register
`SMPU_BDTLS`	Bus Error Details Register
`SMPU_CTL`	SMPU Control Register
`SMPU_EXACADD[n]`	Exclusive Access IDn Address
`SMPU_EXACSTAT[n]`	Exclusive Access Status
`SMPU_IADDR`	Interrupt Address Register
`SMPU_IDTLS`	Interrupt Details Register
`SMPU_RADDR[n]`	Region n Address Register

Table 13-5: ADSP-SC58x SMPU Register List (Continued)

Name	Description
`SMPU_RCTL[n]`	Region n Control Register
`SMPU_REVID`	SMPU Revision ID Register
`SMPU_RIDA[n]`	Region n ID A Register
`SMPU_RIDB[n]`	Region n ID B Register
`SMPU_RIDMSKA[n]`	Region n ID Mask A Register
`SMPU_RIDMSKB[n]`	Region n ID Mask B Register
`SMPU_SECURECTL`	SMPU Control Secure Accesses Register
`SMPU_SECURERCTL[n]`	Region n Control Secure Accesses Register
`SMPU_STAT`	SMPU Status Register

Bus Error Address Register

Programs read the `SMPU_BADDR` and the `SMPU_BDTLS` registers to determine the cause of a bus error. Write errors are prioritized over read errors.

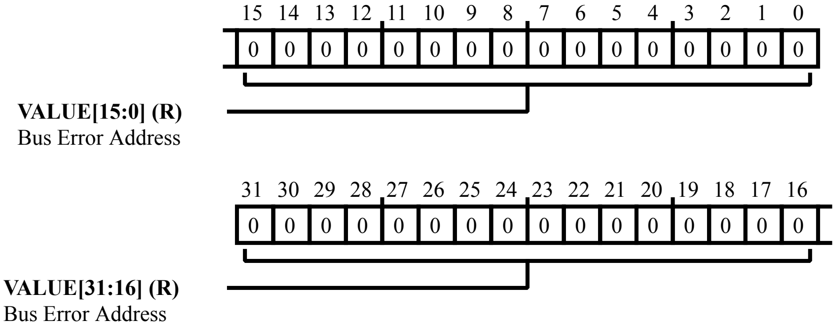

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Bus Error Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Bus Error Address

Figure 13-2: SMPU_BADDR Register Diagram

Table 13-6: SMPU_BADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bus Error Address. The `SMPU_BADDR.VALUE` bit field contains the address of the bus error.

Bus Error Details Register

The SMPU_BDTLS register indicates the ID of the bus error transaction, whether the transaction that caused the last bus error was a read, a write, secure or non-secure.

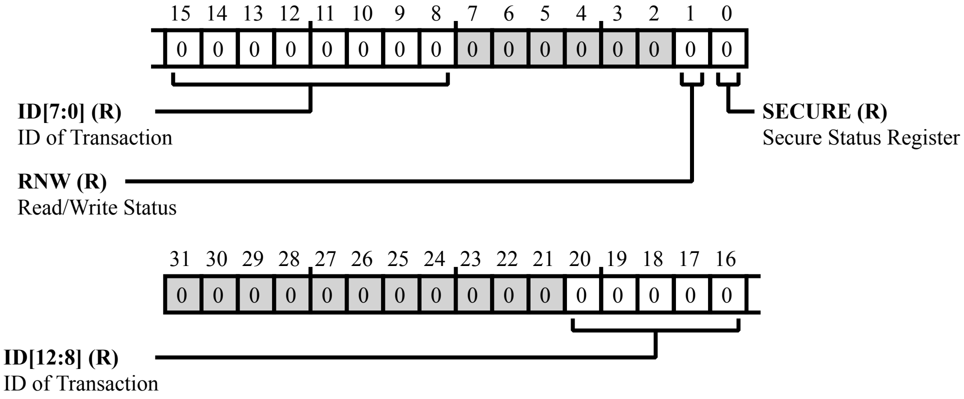

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[7:0] (R)
ID of Transaction
RNW (R)
Read/Write Status
SECURE (R)
Secure Status Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[12:8] (R)
ID of Transaction

Figure 13-3: SMPU_BDTLS Register Diagram

Table 13-7: SMPU_BDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:8 (R/NW)	ID	ID of Transaction. The `SMPU_BDTLS.ID` bit field provides the ID of the transaction that caused the bad address error.	
1 (R/NW)	RNW	Read/Write Status. The `SMPU_BDTLS.RNW` bit indicates whether the last transaction that caused the bad address error was a read or write.	
		0	Transaction that caused last bus error was a write
		1	Transaction that caused last bus error was a read
0 (R/NW)	SECURE	Secure Status Register. The `SMPU_BDTLS.SECURE` bit indicates whether the last transaction that caused the bad address error was secure or non-secure.	
		0	Transaction that caused last bus error was non-secure
		1	Transaction that caused last bus error was secure

SMPU Control Register

The `SMPU_CTL` register provides access to the locking control, error interrupts and SMPU violations.

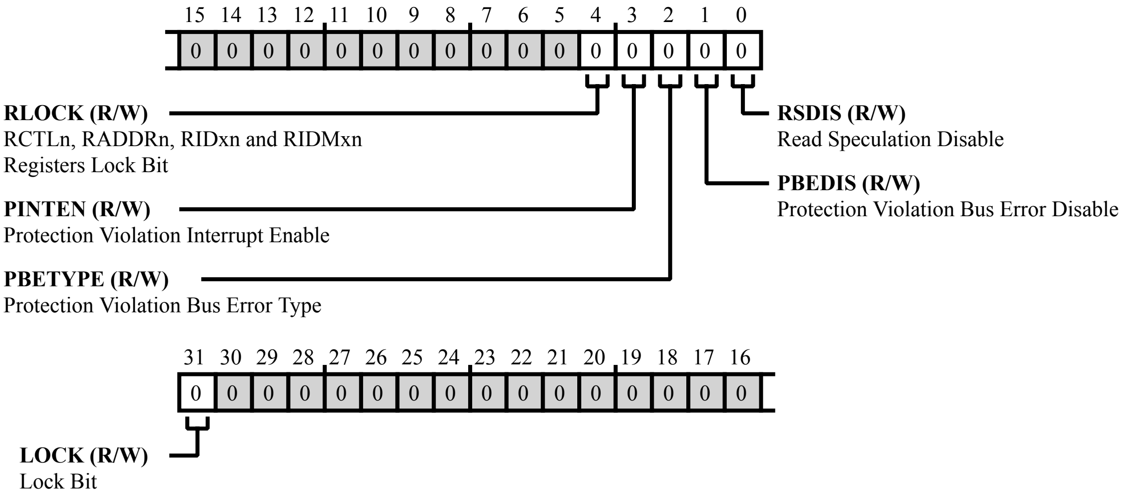

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RLOCK (R/W)
RCTLn, RADDRn, RIDxn and RIDMxn Registers Lock Bit
PINTEN (R/W)
Protection Violation Interrupt Enable
PBETYPE (R/W)
Protection Violation Bus Error Type
RSDIS (R/W)
Read Speculation Disable
PBEDIS (R/W)
Protection Violation Bus Error Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit

Figure 13-4: SMPU_CTL Register Diagram

Table 13-8: SMPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit. When the `SMPU_CTL.LOCK` bit is set and the global lock signal is asserted from the SPU, the `SMPU_CTL` register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.	
		0	CTL Global Lock Disable. The `SMPU_CTL` register is not write-protected.
		1	CTL Global Lock Enable. The `SMPU_CTL` register is write-protected.
4 (R/W)	RLOCK	RCTLn, RADDRn, RIDxn and RIDMxn Registers Lock Bit. When the `SMPU_CTL.RLOCK` bit is set, all the registers associated with region-based control (`SMPU_RCTL[n]`, `SMPU_RADDR[n]`, `SMPU_RIDA[n]`, `SMPU_RIDB[n]`, `SMPU_RIDMSKA[n]` and `SMPU_RIDMSKB[n]`) are write-protected when the global lock signal is active from the SPU. Write access is allowed again when the global lock signal is deasserted.	
		0	Region Registers Write-Protect Enable. All region registers are not write-protected.
		1	Region Registers Write-Protect Disable. All region registers are write-protected.

Table 13-8: SMPU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	PINTEN	Protection Violation Interrupt Enable. The `SMPU_CTL.PINTEN` bit controls whether or not an interrupt is generated when a protection violation occurs.	
		0	Protection Violation IRQ Disable. The protection violation interrupt is disabled.
		1	Protection Violation IRQ Enable. The protection violation interrupt is enabled.
2 (R/W)	PBETYPE	Protection Violation Bus Error Type. The `SMPU_CTL.PBETYPE` bit controls whether a protection violation produces a decode error or a slave error.	
		0	Decode Error Type. Decode error for transactions that violate the configured protection.
		1	Save Error Type. Slave Error for transactions which violate the configured protection
1 (R/W)	PBEDIS	Protection Violation Bus Error Disable. If set, the `SMPU_CTL.PBEDIS` bit blocks protection violations, but does not cause a bus error.	
		0	Bus Error Generation Enable. Transactions which violate the configured protection are blocked and cause a bus error.
		1	Bus Error Generation Disable. Transactions which violate the configured protection are blocked but do not cause a bus error.
0 (R/W)	RSDIS	Read Speculation Disable. The `SMPU_CTL.RSDIS` bit controls whether or not the read addresses are checked before being sent to the slave.	
		0	Read Speculation Enable. Read addresses are sent to the slave without checking.
		1	Read Speculation Disable. Read addresses are checked before being sent to the slave.

Exclusive Access IDn Address

The `SMPU_EXACADD[n]` register provides the address ID of an exclusive access.

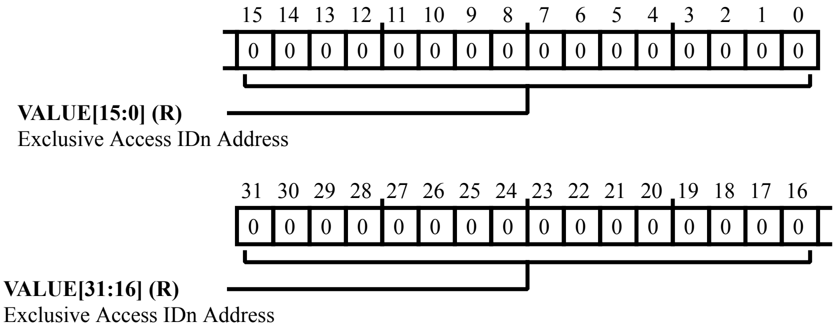

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Exclusive Access IDn Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Exclusive Access IDn Address

Figure 13-5: SMPU_EXACADD[n] Register Diagram

Table 13-9: SMPU_EXACADD[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Exclusive Access IDn Address.

Exclusive Access Status

The `SMPU_EXACSTAT[n]` register provides the exclusive access ID, read size and read length as well as the indication that the access was valid.

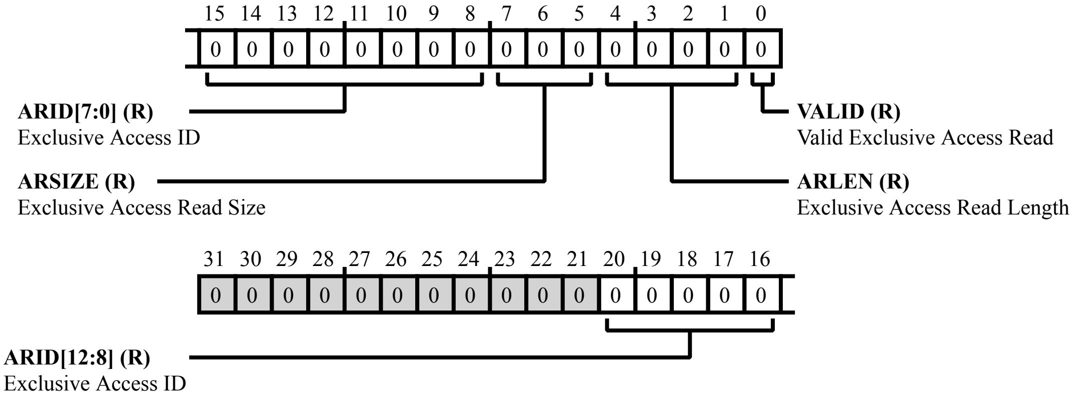

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARID[7:0] (R)
Exclusive Access ID
ARSIZE (R)
Exclusive Access Read Size
VALID (R)
Valid Exclusive Access Read
ARLEN (R)
Exclusive Access Read Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARID[12:8] (R)
Exclusive Access ID

Figure 13-6: SMPU_EXACSTAT[n] Register Diagram

Table 13-10: SMPU_EXACSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20:8 (R/NW)	ARID	Exclusive Access ID.
7:5 (R/NW)	ARSIZE	Exclusive Access Read Size.
4:1 (R/NW)	ARLEN	Exclusive Access Read Length.
0 (R/NW)	VALID	Valid Exclusive Access Read.

Interrupt Address Register

The `SMPU_IADDR` register indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The SMPU issues a bus error for this condition.

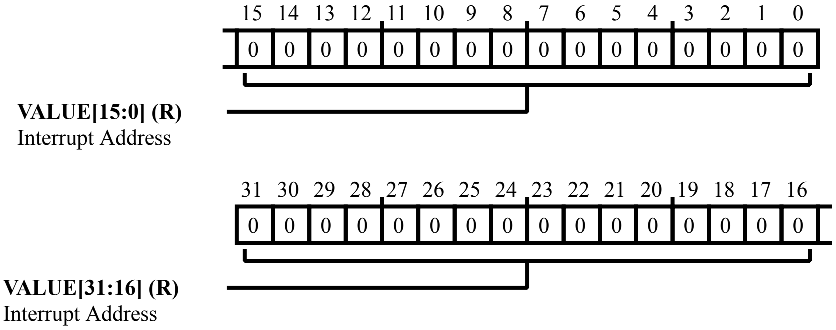

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Interrupt Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Interrupt Address

Figure 13-7: SMPU_IADDR Register Diagram

Table 13-11: SMPU_IADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Interrupt Address. The `SMPU_IADDR.VALUE` bit field is the address where an attempt to access an unimplemented address or a non-aligned access has occurred.

Interrupt Details Register

The `SMPU_IDTLS` register provides the ID of the last signaled interrupt, whether the interrupt was caused by a read or write, and whether the transaction that caused the last signaled interrupt was secure.

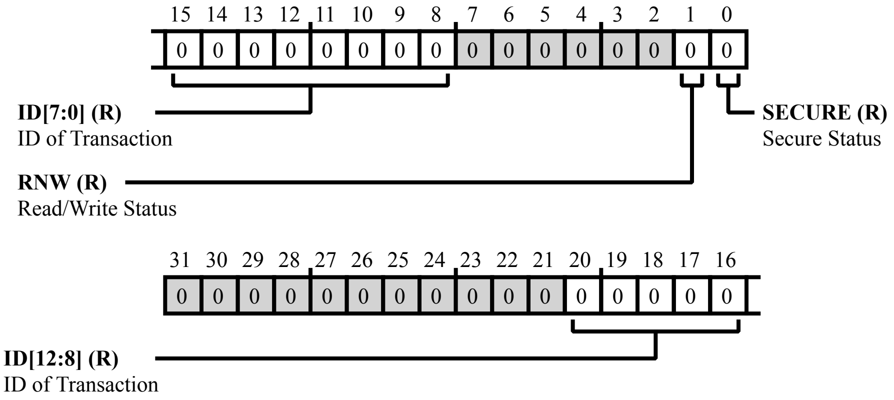

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[7:0] (R)
ID of Transaction
SECURE (R)
Secure Status
RNW (R)
Read/Write Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID[12:8] (R)
ID of Transaction

Figure 13-8: SMPU_IDTLS Register Diagram

Table 13-12: SMPU_IDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:8 (R/NW)	ID	ID of Transaction. The `SMPU_IDTLS.ID` bit field provides the ID of the transaction that caused the interrupt.	
1 (R/NW)	RNW	Read/Write Status. The `SMPU_IDTLS.RNW` bit indicates whether the last transaction that caused the interrupt was a read or write.	
		0	Transaction that caused last signaled interrupt was a write
		1	Transaction that caused last signaled interrupt was a read
0 (R/NW)	SECURE	Secure Status. The `SMPU_IDTLS.SECURE` bit indicates whether the last transaction that caused the interrupt was secure or non-secure.	
		0	Transaction that caused last signaled interrupt was non-secure
		1	Transaction that caused last signaled interrupt was secure

Region n Address Register

The `SMPU_RADDR[n]` register is used to define the base address for a memory region to be protected.

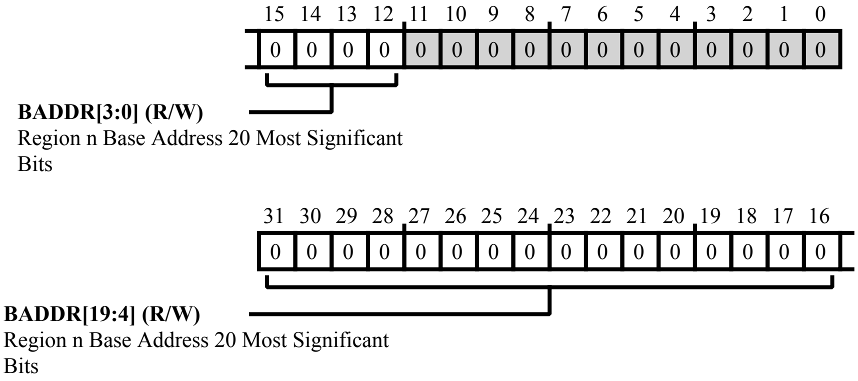

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BADDR[3:0] (R/W)
Region n Base Address 20 Most Significant Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BADDR[19:4] (R/W)
Region n Base Address 20 Most Significant Bits

Figure 13-9: SMPU_RADDR[n] Register Diagram

Table 13-13: SMPU_RADDR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:12 (R/W)	BADDR	Region n Base Address 20 Most Significant Bits. The `SMPU_RADDR[n].BADDR` bit field defines the base address for a memory region to be protected.

Region n Control Register

The `SMPU_RCTL[n]` register is used to define the level of protection for a region of memory. The protection of a region is controlled and defined by this register and the `SMPU_RADDR[n]`, `SMPU_RIDA[n]`, `SMPU_RIDB[n]`, `SMPU_RIDMSKA[n]`, and `SMPU_RIDMSKB[n]` registers.

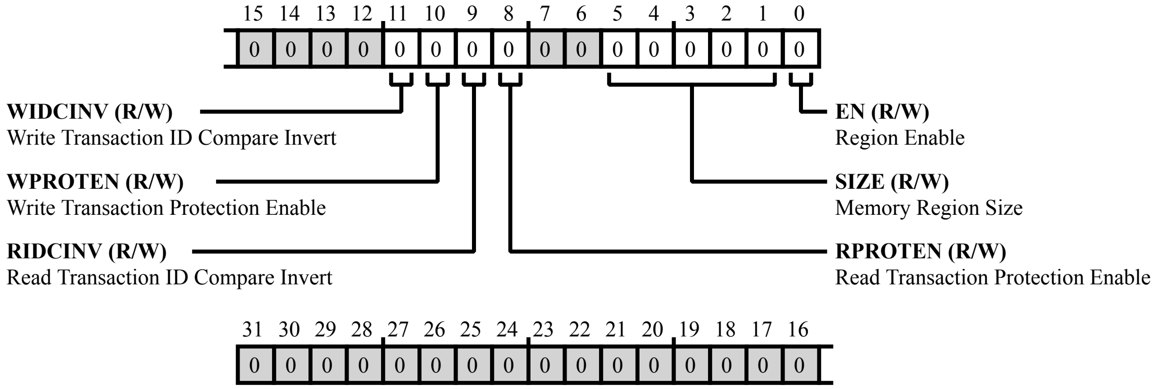

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WIDCINV (R/W)
Write Transaction ID Compare Invert
WPROTEN (R/W)
Write Transaction Protection Enable
RIDCINV (R/W)
Read Transaction ID Compare Invert
EN (R/W)
Region Enable
SIZE (R/W)
Memory Region Size
RPROTEN (R/W)
Read Transaction Protection Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-10: SMPU_RCTL[n] Register Diagram

Table 13-14: SMPU_RCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	WIDCINV	Write Transaction ID Compare Invert. The `SMPU_RCTL[n].WIDCINV` bit inverts the write ID match result.	
		0	Write transaction ID comparison result not inverted
		1	Write transaction ID comparison result inverted
10 (R/W)	WPROTEN	Write Transaction Protection Enable. The `SMPU_RCTL[n].WPROTEN` bit enables protection against ID-based write transactions for the memory region.	
		0	Write transaction ID-based protection disabled
		1	Write transaction ID-based protection enabled
9 (R/W)	RIDCINV	Read Transaction ID Compare Invert. When the `SMPU_RCTL[n].RIDCINV` bit is set, the read ID match result is inverted.	
		0	Read transaction ID comparison result not inverted
		1	Read transaction ID comparison result inverted

Table 13-14: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	RPROTEN	Read Transaction Protection Enable. The `SMPU_RCTL[n].RPROTEN` bit enable bit to turn on protection against ID-based read transactions for the memory region.	
		0	Read transaction ID-based protection disabled
		1	Read transaction ID-based protection enabled
5:1 (R/W)	SIZE	Memory Region Size. The `SMPU_RCTL[n].SIZE` bit defines the size of the memory region to be protected.	
		0	4 KB
		1	8 KB
		2	16 KB
		3	32 KB
		4	64 KB
		5	128 KB
		6	256 KB
		7	512 KB
		8	1 MB
		9	2 MB
		10	4 MB
		11	8 MB
		12	16 MB
		13	32 MB
		14	64 MB
		15	128 MB
		16	256 MB
		17	512 MB
		18	1 GB
		19	2 GB
		20	4 GB
		21-31	Reserved

Table 13-14: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Region Enable. The `SMPU_RCTL[n].EN` bit enables the protection of a region.	
		0	Disabled
		1	Enabled

SMPU Revision ID Register

The `SMPU_REVID` register provides the major and minor revision numbers of this module.

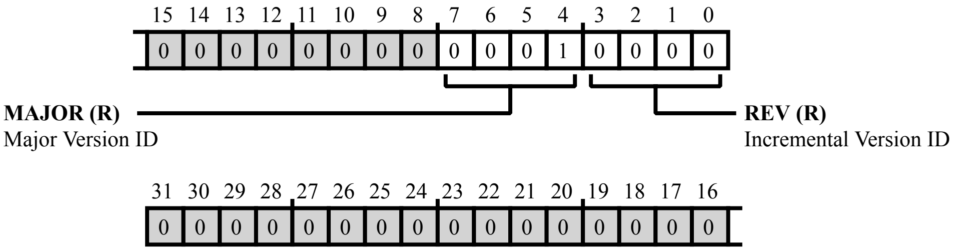

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
MAJOR (R)
Major Version ID
REV (R)
Incremental Version ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-11: SMPU_REVID Register Diagram

Table 13-15: SMPU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Region n ID A Register

The SMPU_RIDA[n] register is used for ID comparison 'A'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more detail, refer to the ID Comparison section.

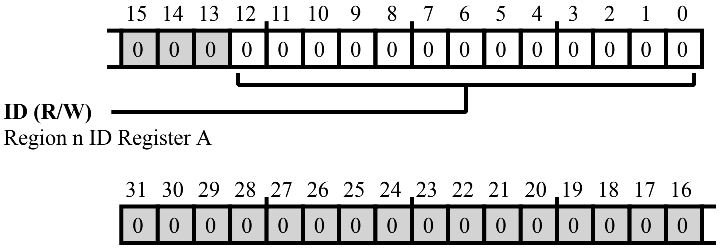

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID (R/W)
Region n ID Register A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-12: SMPU_RIDA[n] Register Diagram

Table 13-16: SMPU_RIDA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	ID	Region n ID Register A. The `SMPU_RIDA[n].ID` bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID B Register

The `SMPU_RIDB[n]` register is used for ID comparison 'B'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more details, refer to the ID Comparison section.

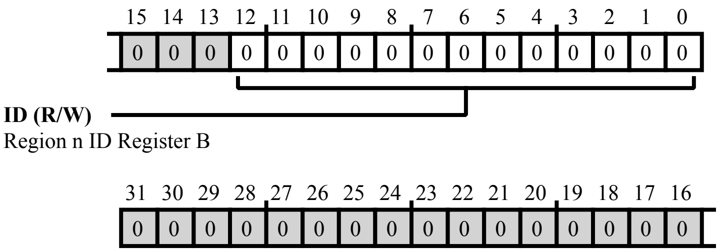

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID (R/W)
Region n ID Register B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-13: SMPU_RIDB[n] Register Diagram

Table 13-17: SMPU_RIDB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	ID	Region n ID Register B. The `SMPU_RIDB[n].ID` bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID Mask A Register

The `SMPU_RIDMSKA[n]` register is used for ID comparison 'A'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

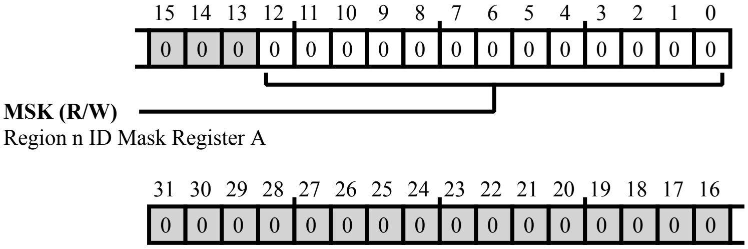

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK (R/W)
Region n ID Mask Register A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-14: SMPU_RIDMSKA[n] Register Diagram

Table 13-18: SMPU_RIDMSKA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	MSK	Region n ID Mask Register A. The `SMPU_RIDMSKA[n].MSK` bit field, combined with the incoming transaction, provides the means to bypass the configured memory protection for a region.

Region n ID Mask B Register

The `SMPU_RIDMSKB[n]` register is used for ID comparison 'B'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

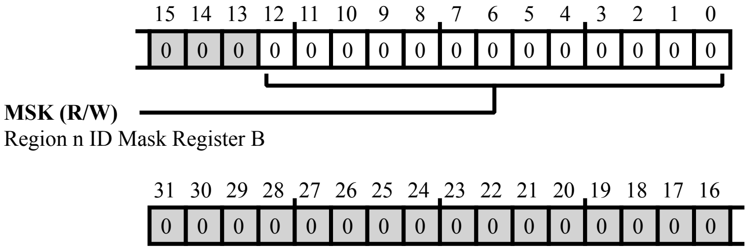

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK (R/W)
Region n ID Mask Register B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-15: SMPU_RIDMSKB[n] Register Diagram

Table 13-19: SMPU_RIDMSKB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	MSK	Region n ID Mask Register B. The `SMPU_RIDMSKB[n].MSK` bit field, combined with the incoming transaction provides the means to bypass the configured memory protection for a region.

SMPU Control Secure Accesses Register

The `SMPU_SECURECTL` register provides the bits required to set up the security settings for the processor. These settings includes error generation and read/write security.

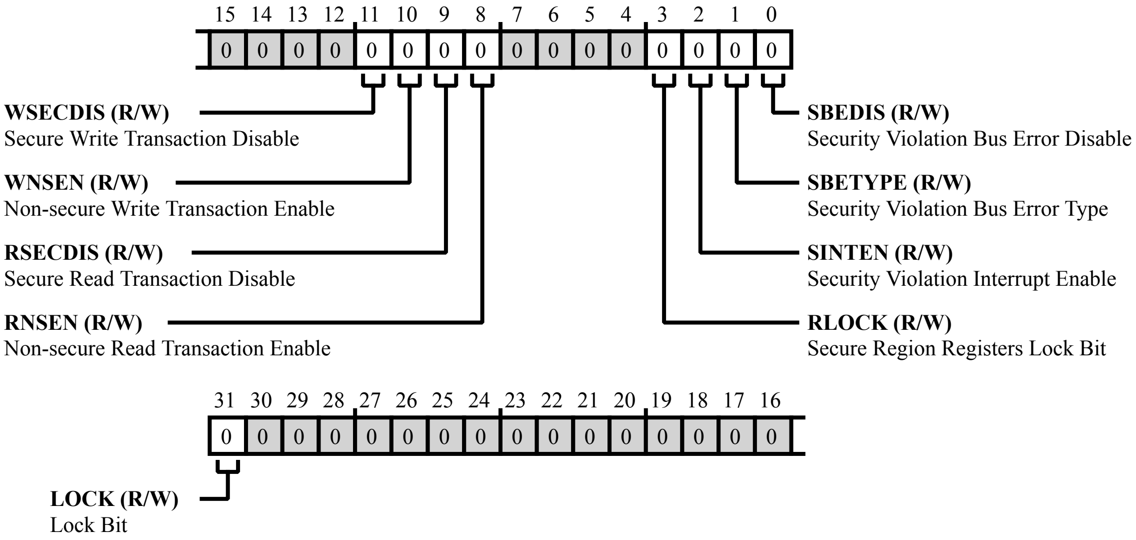

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WSECDIS (R/W)
Secure Write Transaction Disable
WNSEN (R/W)
Non-secure Write Transaction Enable
RSECDIS (R/W)
Secure Read Transaction Disable
RNSEN (R/W)
Non-secure Read Transaction Enable
SBEDIS (R/W)
Security Violation Bus Error Disable
SBETYPE (R/W)
Security Violation Bus Error Type
SINTEN (R/W)
Security Violation Interrupt Enable
RLOCK (R/W)
Secure Region Registers Lock Bit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit

Figure 13-16: SMPU_SECURECTL Register Diagram

Table 13-20: SMPU_SECURECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit. When the `SMPU_SECURECTL.LOCK` bit is set and the global lock signal is asserted from the SPU, the `SMPU_SECURECTL` register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.	
		0	`SMPU_SECURECTL` is not write-protected
		1	`SMPU_SECURECTL` is write-protected
11 (R/W)	WSECDIS	Secure Write Transaction Disable. The `SMPU_SECURECTL.WSECDIS` bit disables secure write transactions.	
		0	Enable secure write transactions
		1	Disable secure write transactions
10 (R/W)	WNSEN	Non-secure Write Transaction Enable. The `SMPU_SECURECTL.WNSEN` bit enables non-secure write transactions.	
		0	Disable non-secure writes
		1	Enable non-secure writes

Table 13-20: SMPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	RSECDIS	Secure Read Transaction Disable. The `SMPU_SECURECTL.RSECDIS` bit disables secure read transactions.	
		0	Enable secure read transactions
		1	Disable secure read transactions
8 (R/W)	RNSEN	Non-secure Read Transaction Enable. The `SMPU_SECURECTL.RNSEN` bit enables non-secure read transactions.	
		0	Disable non-secure read transactions
		1	Enable non-secure read transactions
3 (R/W)	RLOCK	Secure Region Registers Lock Bit. When the `SMPU_SECURECTL.RLOCK` bit is set, the secure region control registers, `SMPU_SECURERCTL[n]`, are write-protected when the global lock signal is active from the SPU. When the global lock signal is deasserted, write access is allowed again.	
		0	Disable write-protection on secure region registers
		1	Enable write-protection on secure region registers
2 (R/W)	SINTEN	Security Violation Interrupt Enable. The `SMPU_SECURECTL.SINTEN` bit enables interrupt generation when a security violation occurs.	
		0	Disable security settings violation interrupt
		1	Enable security settings violation interrupt
1 (R/W)	SBETYPE	Security Violation Bus Error Type. The `SMPU_SECURECTL.SBETYPE` bit controls whether a decode error or a slave error is returned when a security violation occurs.	
		0	Return a decode error error which violates the security settings
		1	Return a slave error which violates the security settings
0 (R/W)	SBEDIS	Security Violation Bus Error Disable. The `SMPU_SECURECTL.SBEDIS` bit controls whether or not a bus error is caused when a security violation occurs.	
		0	Enable bus error
		1	Disable bus error

Region n Control Secure Accesses Register

The `SMPU_SECURERCTL[n]` register contains bits that configure read/write security for a specific region.

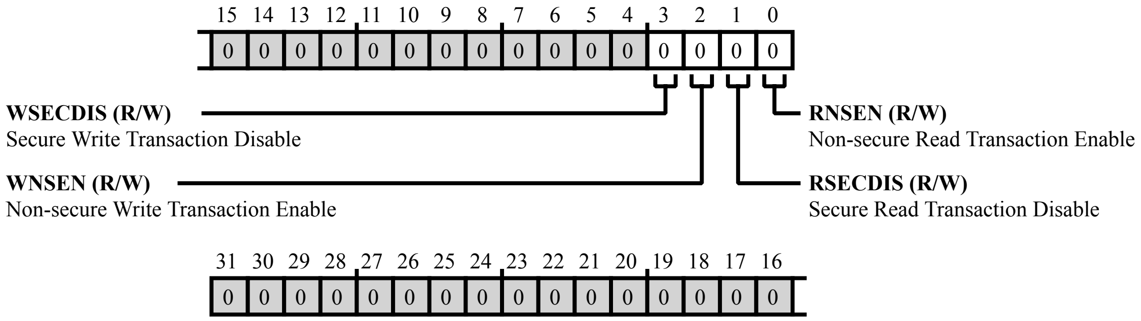

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WSECDIS (R/W)
Secure Write Transaction Disable
WNSEN (R/W)
Non-secure Write Transaction Enable
RNSEN (R/W)
Non-secure Read Transaction Enable
RSECDIS (R/W)
Secure Read Transaction Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 13-17: SMPU_SECURERCTL[n] Register Diagram

Table 13-21: SMPU_SECURERCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	WSECDIS	Secure Write Transaction Disable. The `SMPU_SECURERCTL[n].WSECDIS` bit disables secure write transactions for the memory region.	
		0	Enable secure write transactions to this region
		1	Disable secure write transactions to this region
2 (R/W)	WNSEN	Non-secure Write Transaction Enable. This `SMPU_SECURERCTL[n].WNSEN` bit enables non-secure write transactions for the memory region.	
		0	Disable non-secure write transactions to this region
		1	Enable non-secure write transactions to this region
1 (R/W)	RSECDIS	Secure Read Transaction Disable. The `SMPU_SECURERCTL[n].RSECDIS` bit disables secure read transactions for the memory region.	
		0	Enable secure read transactions to this region
		1	Disable secure read transactions to this region
0 (R/W)	RNSEN	Non-secure Read Transaction Enable. The `SMPU_SECURERCTL[n].RNSEN` bit enables non-secure read transactions for the memory region.	
		0	Disable non-secure read transactions to this region
		1	Enable non-secure read transactions to this region

SMPU Status Register

The SMPU_STAT register provides the state of the SMPU and indicates various errors. All bits in this register are write 1 to clear.

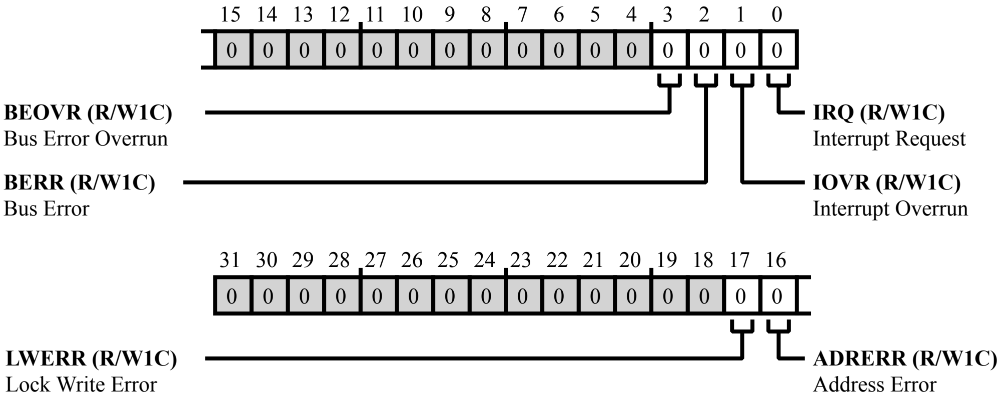

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BEOVR (R/W1C)
Bus Error Overrun
BERR (R/W1C)
Bus Error
IRQ (R/W1C)
Interrupt Request
IOVR (R/W1C)
Interrupt Overrun
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LWERR (R/W1C)
Lock Write Error
ADRERR (R/W1C)
Address Error

Figure 13-18: SMPU_STAT Register Diagram

Table 13-22: SMPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	LWERR	Lock Write Error. The `SMPU_STAT.LWERR` bit is set when `SMPU_CTL.LOCK` bit =1, the global lock signal is asserted from the SPU and a read or write attempt was made to the SMPU_CTL MMR.	
		0	No Lock Write Error
		1	Lock Write Error
16 (R/W1C)	ADRERR	Address Error. The `SMPU_STAT.ADRERR` bit is set when the SMPU MMR is accessed as an unaligned address, or when a read-only MMR is written to.	
		0	No Address Error
		1	Address Error
3 (R/W1C)	BEOVR	Bus Error Overrun. The `SMPU_STAT.BEOVR` bit indicates that another bus error had occurred. Any new information about the most recent violation which caused the bus error is not captured.	
		0	No Bus Error overrun
		1	Bus Error overrun has occurred

Table 13-22: SMPU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	BERR	Bus Error. This `SMPU_STAT.BERR` bit indicates if a bus error was generated.	
		0	No Bus Error since this bit has been cleared
		1	Bus Error has been generated
1 (R/W1C)	IOVR	Interrupt Overrun. The `SMPU_STAT.IOVR` bit indicates if another violation occurred while the previous violation interrupt was not finished being serviced. Information about the most recent violation is then not captured.	
		0	No Interrupt overrun
		1	Interrupt overrun has occurred
0 (R/W1C)	IRQ	Interrupt Request. The `SMPU_STAT.IRQ` bit provides an indication that an interrupt has been generated.	
		0	No Interrupt since this bit has been cleared
		1	Interrupt has been generated

14 General-Purpose Ports (PORT)

This section describes general-purpose ports, pin multiplexing, general-purpose input/output (GPIO) functionality, and pin interrupts. The general-purpose ports provide the following three functions:

- Pin multiplexing scheme
- GPIO functionality
- Pin interrupt requests

NOTE: In this chapter, the naming convention for registers and bits omits the alphabetic group enumeration to refer to any and all of the ports. For example, `PORT_FER` represents registers `PORTA_FER`, `PORTB_FER`, and so on. Likewise `PORT_FER.PX1` represents bits PA1, PB1, and so on.

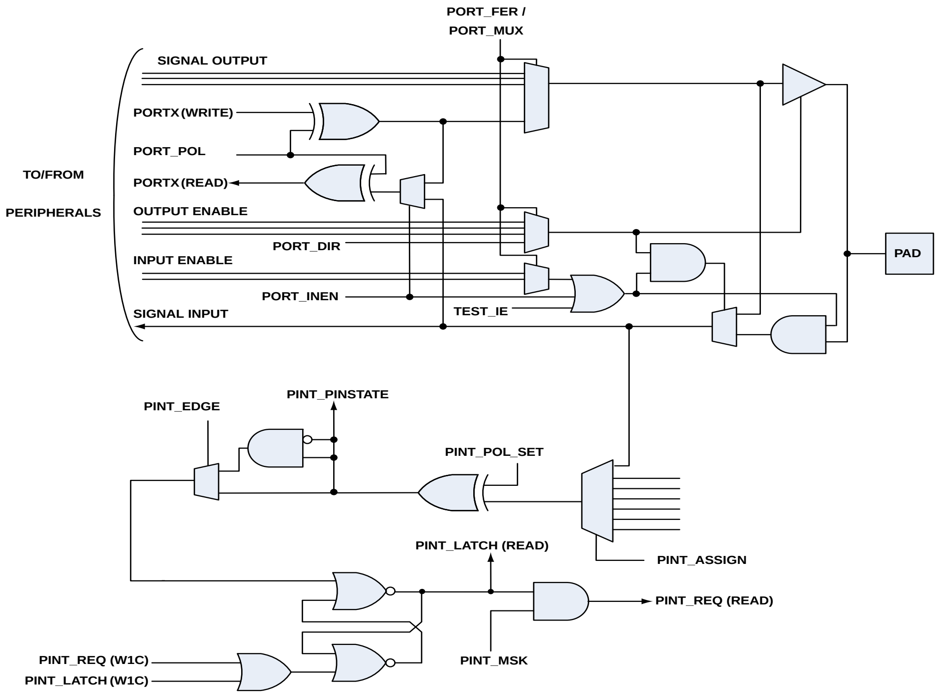

PORT_FER /
PORT_MUX
SIGNAL OUTPUT
PORTX (WRITE)
PORT_POL
TO/FROM
PERIPHERALS
PORTX (READ)
OUTPUT ENABLE
PORT_DIR
INPUT ENABLE
PORT_INEN
TEST_IE
SIGNAL INPUT
PAD
PINT_EDGE
PINT_PINSTATE
PINT_POL_SET
PINT_LATCH (READ)
PINT_ASSIGN
PINT_REQ (READ)
PINT_REQ (W1C)
PINT_LATCH (W1C)
PINT_MSK

Figure 14-1: Simplified GPIO and Pin Interrupt Signal Flow

PORT Features

The PORTs include the following features:

- Input mode, output mode, and open-drain mode of GPIO operation
- Port multiplexing controlled on a pin-by-pin basis
- No external glue hardware required for unused pins
- All port pins provide interrupt request functionality
- Byte-wide pin-to-interrupt request assignment

PORT Functional Description

The number of ports and each's composition are defined in the processor datasheet. Each port has a dedicated set of MMR registers that control pin functions and operates in general-purpose I/O (GPIO) mode by default, as controlled by the port-specific `PORT_FER` register. Each bit in this register, as well as the other PORT MMRs, represents a specific GPIO pin on the specified port.

Input Mode, Output Mode, and Open-Drain Mode of GPIO Operation

At reset, every GPIO pin defaults to input mode with the input drivers disabled. To enable any GPIO input driver, set the bits corresponding to the individual pins in the appropriate input enable register (`PORT_INEN`).

The GPIO output drivers are enabled by setting the corresponding bits in the direction registers (`PORT_DIR`).

The PORT can use every GPIO in open-drain mode by clearing the respective bit in the `PORT_DATA` register or setting the respective bit in the `PORT_DATA_CLR` register. Then, set the corresponding bit in the `PORT_INEN` register. Read from the `PORT_DATA` register to obtain the status from the pin.

Port Multiplexing Controlled on Pin-by-Pin Basis

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (`PORT_FER`) registers and the 32-bit port multiplexing (`PORT_MUX`) registers.

All Port Pins Provide Interrupt Functionality

Pin interrupts are completely decoupled from GPIO functionality. Pins are connected to the system event controller (SEC) via the PINTx modules, each of which is configurable in terms of which port pins are sensed for interrupt generation.

ADSP-SC58x PORT Register List

The PORT module (PORT) regulates the use of the multiplexable processor pins. Every port pin can operate in general-purpose I/O (GPIO) mode or as an alternate function. This GPIO operation is the default after processor reset and is controlled by a set of registers that control GPIO functionality. Every bit in these registers represents a certain GPIO pin of a specific port. For more information on PORT functionality, see the PORT register descriptions.

Table 14-1: ADSP-SC58x PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register

ADSP-SC58x PINT Register List

The Pin Interrupt module (PINT) controls the pin-to-interrupt assignment in a byte-wide manner. The pin-interrupt assignment registers do not consist of 32 individual bits. They consist of four control bytes, each functioning as a multiplexer control. For more information, see the PINT register descriptions.

All PINT registers are 32 bits wide and can be accessed by 32-bit load/store instructions. They also support 16-bit operation where the upper 16 bits are ignored and the application uses the lower 16 bits only. Consequently, all PINT registers support 32-bit accesses as well as 16-bit accesses for the lower half words. Applications may use faster 16-bit accesses as long as they do not require functionality of upper register halves.

Table 14-2: ADSP-SC58x PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

ADSP-SC58x PINT Interrupt List

Table 14-3: ADSP-SC58x PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
38	PINT0_BLOCK	PINT0 Pin Interrupt Block 0	Level	
39	PINT1_BLOCK	PINT1 Pin Interrupt Block 1	Level	
40	PINT2_BLOCK	PINT2 Pin Interrupt Block 2	Level	
41	PINT3_BLOCK	PINT3 Pin Interrupt Block 3	Level	
42	PINT4_BLOCK	PINT4 Pin Interrupt Block 4	Level	
43	PINT5_BLOCK	PINT5 Pin Interrupt Block 5	Level	

ADSP-SC58x PINT Trigger List

Table 14-4: ADSP-SC58x PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
14	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
15	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level
16	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level
17	PINT3_BLOCK	PINT3 Pin Interrupt Block	Level
18	PINT4_BLOCK	PINT4 Pin Interrupt Block	Level
19	PINT5_BLOCK	PINT5 Pin Interrupt Block	Level

Table 14-5: ADSP-SC58x PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-SC58x PADS Register List

The PADS controls signal hysteresis and other system interface signal features for a number of module interfaces.

Table 14-6: ADSP-SC58x PADS Register List

Name	Description
`PADS_DAI0_IE`	DAI0 Port Input Enable Control Register
`PADS_DAI1_IE`	DAI1 Port Input Enable Control Register
`PADS_PCFG0`	Peripheral PAD Configuration0 Register

PORT Architectural Concepts

These sections describe in more detail how the PORT module connects externally to pins and internally to the MMR bus. Ports are named alphabetically beginning with A.

- Internal Interfaces
- External Interfaces
- GPIO Functionality
- Port Multiplexing Control

Internal Interfaces

All of the pin multiplexing, GPIO, and pin interrupt control block MMRs can be accessed through the MMR bus. There is no DMA support. Each of the pin interrupt (PINTx) modules has its own dedicated interrupt request output signal that connects directly to the system event controller (SEC).

External Interfaces

The pin multiplexing hardware can be seen as a layer between the on-chip peripherals and the silicon pads connecting to the physical pins/balls or the package, as controlled by the PORT unit.

GPIO Functionality

By default, the PORT sets every GPIO pin to input mode. The input drivers are not enabled, which avoids the need for unnecessary current sinks and external termination resistors on unused pins.

Input Mode

The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the bits corresponding to the PORT pins in the appropriate input enable register (`PORT_INEN`). When enabled, a read from the `PORT_DATA` register returns the logical state of the input pins.

However, the input signal does not overwrite the state of the internal flip-flop used for providing output to the same pin. Only software can alter the state. If the input driver is enabled, a write to the `PORT_DATA` register can alter the state of the flip-flop, but the change cannot be read back.

Output Mode

Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the bits corresponding to the PORT pins in the appropriate direction register. The PORT implements direction registers as a pair of write-1-to-set (W1S) and write-1-to-clear (W1C) MMRs called `PORT_DIR_SET` and `PORT_DIR_CLR`, respectively. As such, software can alter the direction of the signal flow on individual GPIO pins without impacting other GPIOs on the same port.

Both the `PORT_DIR_SET` and `PORT_DIR_CLR` registers return the same value when read, and a logical 1 indicates an enabled output. The `PORT_DATA` registers control the state of output pins. A logical 0 drives the output low while a logical 1 drives the output high.

While writes to the `PORT_DATA` register can alter all of the GPIOs on a specific port at once, there are also a pair of W1S and W1C MMRs called `PORT_DATA_SET` and `PORT_DATA_CLR`, respectively. These registers enable the manipulation of individual GPIO outputs. The state of the outputs can be obtained by reading the `PORT_DATA` registers. Because the state of the GPIO output can be controlled before the output driver is enabled, set or clear the internal flip-flop first by programming this register to avoid volatile levels on the output pin.

Open-Drain Mode

Every GPIO can also be used in open-drain mode. First, either clear the respective bit in the `PORT_DATA` register or set the respective bit in the `PORT_DATA_CLR` register. Then, set the appropriate bit in the `PORT_INEN` register. Read from the `PORT_DATA` register to return the status from the pin rather than the state of the internal flip-flop.

By toggling the output driver through the `PORT_DIR_SET` and `PORT_DIR_CLR` register pair, the output signal can be pulled low or three-stated, as required. The polarity of the driven signal can be inverted when the internal flip-flop is set. When using a GPIO port in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pins.

Port Multiplexing Control

To configure pins properly, consult the processor datasheet to determine which bits in the `PORT_FER` and `PORT_MUX` register map to the pin of interest, and then set these registers appropriately for the desired function.

After reset, all port pins default to GPIO input mode with their output and input drivers disabled. As a result, all unused port pins can be left unconnected. Disabled pins appear as high-impedance to external circuits.

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (`PORT_FER`) registers and the 32-bit port multiplexing (`PORT_MUX`) registers.

The function enable register specifies whether the pin is used as a GPIO pin or allocated for use by a specific peripheral, but it does not specify what the peripheral function is. Each bit in the 16-bit `PORT_FER` register corresponds to an individual port pin. For example, if bit 1 (PA1) of the `PORTA_FER` register is cleared, the PA_01 pin is configured as a GPIO. When set, one of the available peripheral functions becomes active on the PA_01 pin instead.

Pairs of bits in the `PORT_MUX` register control the multiplexing between the peripheral functions available to an individual pin, as some PORT pins provide up to four possible peripheral functions.

Refer to the Signal Muxing table in the datasheet for the specific `PORT_MUX` settings.

PORT Event Control

The following sections describe event generation in the PORT module.

PORT Interrupt Signals

The pin interrupts are decoupled from GPIO functionality, providing the following advantages.

- Flexible mapping scheme enables pins from up to four different ports to be grouped into one common interrupt scheme.
- Interrupt requests work on input and output pins regardless of whether the pin is functioning as a GPIO or a peripheral.

The processor has a number of interrupt channels dedicated to pin interrupts, managed by a set of pin interrupt (PINTx) blocks. Each PINTx block can sense up to 32 GPIO pins, as described in the following list and figure.

- PINT0 can sense pin activity on PORTA and PORTB
- PINT1 can sense pin activity on PORTB and PORTC
- PINT2 can sense pin activity on PORTC and PORTD
- PINT3 can sense pin activity on PORTD and PORTE
- PINT4 can sense pin activity on PORTE and PORTF
- PINT5 can sense pin activity on PORTF and PORTG. Note that PORTF and PORTG are not available on the low pin count package.

The processor supports both 32-bit and 16-bit peripheral bus accesses to PINTx registers.

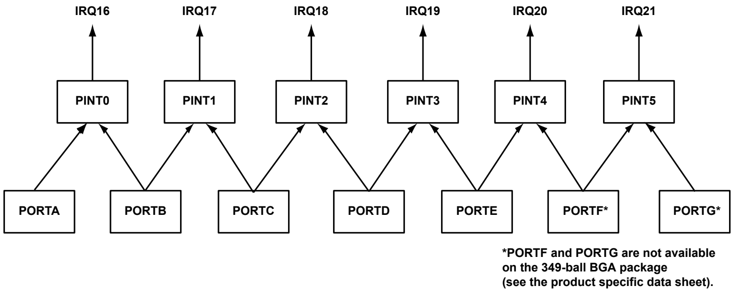

IRQ16
IRQ17
IRQ18
IRQ19
IRQ20
IRQ21
PINT0
PINT1
PINT2
PINT3
PINT4
PINT5
PORTA
PORTB
PORTC
PORTD
PORTE
PORTF*
PORTG*
*PORTF and PORTG are not available
on the 349-ball BGA package
(see the product specific data sheet).

Figure 14-2: GPIO to PINTx Assignment

Pins connect to the PINTx module and then to the system event controller (SEC), as shown in the ***PINTx Block Diagram***.

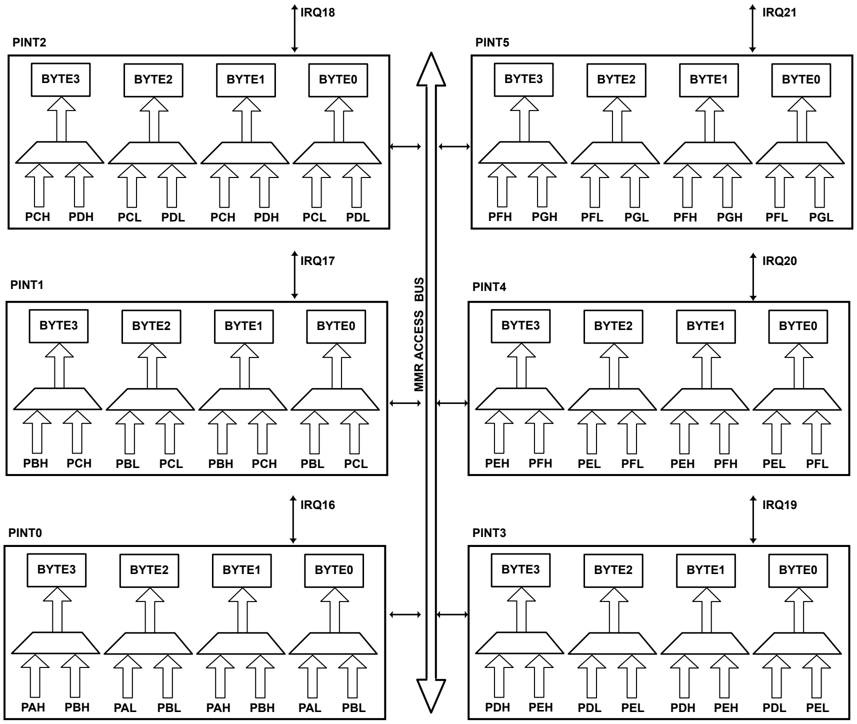

IRQ18
PINT2
BYTE3
BYTE2
BYTE1
BYTE0
PCH
PDH
PCL
PDL
PCH
PDH
PCL
PDL
IRQ21
PINT5
BYTE3
BYTE2
BYTE1
BYTE0
PFH
PGH
PFL
PGL
PFH
PGH
PFL
PGL
IRQ17
PINT1
BYTE3
BYTE2
BYTE1
BYTE0
PBH
PCH
PBL
PCL
PBH
PCH
PBL
PCL
MMR ACCESS BUS
IRQ20
PINT4
BYTE3
BYTE2
BYTE1
BYTE0
PEH
PFH
PEL
PFL
PEH
PFH
PEL
PFL
IRQ16
PINT0
BYTE3
BYTE2
BYTE1
BYTE0
PAH
PBH
PAL
PBL
PAH
PBH
PAL
PBL
IRQ19
PINT3
BYTE3
BYTE2
BYTE1
BYTE0
PDH
PEH
PDL
PEL
PDH
PEH
PDL
PEL

Figure 14-3: ADSP-SC57x/SC58x PINTx Block Diagram

As shown in the ***PINTx Block Diagram***, each port is subdivided into two 8-pin half ports, upper (`PxH`) and lower (`PxL`). The `PINT_ASSIGN` registers control the 8-bit multiplexers associated with these half ports, where the lower half units (eight pins) can be forwarded to either byte 0 or byte 2 of the PINTx blocks, and the upper half units (eight pins) can be forwarded to either byte 1 or byte 3 of the PINTx blocks.

When a half port is assigned to a byte in any PINTx block, the state of the eight pins appears in the `PINT_PINSTATE` register, regardless of whether the pin is enabled for GPIO or peripheral functions (input or output). When neither the input nor output drivers of the pin are enabled, the pin state is read as zero. The `PINT_PINSTATE` register reports the inverted state of the pin when the `PINT_INV_SET` register activates the signal inverter. The inverter can be enabled on an individual bit-by-bit basis. Each bit in the `PINT_INV_SET`/ `PINT_INV_CLR` register pair represents a pin signal.

By default, PORT interrupt request generation is level-sensitive, and an interrupt request occurs when the enabled pin is sensed as active high. Use the `PINT_EDGE_SET` register to change the interrupt request genearion scheme to instead be edge-sensitive (rising edge generates the interrupt request). Use the `PINT_INV_SET` register to invert the polarity such that the PINTx block generates the interrupt request on active-low signals or falling edges.

The PINTx modules also assist when both signal edges must generate unique interrupt requests. If two different interrupt requests are required, the `PINT_ASSIGN` registers can route a single input signal to two different PINTx blocks, where one block inverts the signal in the `PINT_INV_SET` register and the other one does not. In this fashion, a unique software routine is associated with the hardware PINTx block that is generating the unique interrupt request for each signal edge. When both signal edges can be serviced by the same interrupt request, each half port can be routed to two separate bytes within a single PINTx block using the `PINT_ASSIGN` register, and then one of the half ports needs to have the inversion enabled in the `PINT_INV_SET` register. The servicing software routine can then detect from the `PINT_LATCH` register whether a falling, rising, or both edges have occurred.

Regardless of whether level-sensitive or edge-sensitive mode is used, the hardware always latches an interrupt request. Latched signals can be read from the `PINT_LATCH` registers. Only a software or hardware reset clears the latches. To clear the latch, a W1C operation must be performed to the `PINT_REQ` or `PINT_LATCH` register. When in level-sensitive mode, the interrupt request remains asserted if the pin state does not change by the time the interrupt service routine exits.

Because every PINTx block groups up to 32 pin signals, the `PINT_MSK_SET`/ `PINT_MSK_CLR` register pair can control which of the signals can request an interrupt at the system level. Software can interrogate the `PINT_REQ` register for signaling pins. The `PINT_REQ` bits represent a logical AND between the mask and the latch. When any of these bits is set, the block output interrupt request is asserted.

PORT Programming Model

The ***GPIO Programming Model Flow (Part 1)***, ***GPIO Programming Model Flow (Part 2)***, and ***GPIO Programming Model Flow (Part 3)*** figures show the programming model for the general-purpose ports. This programming includes the GPIO input and output operation, open-drain mode, and the pin interrupt PINTx modules.

> **NOTE:** These process flow diagrams connect where call-out letters appear. For example, call-out A in the ***GPIO Programming Model Flow (Part 1)*** diagram connects to call-out A in the ***GPIO Programming Model Flow (Part 2)*** diagram.

The following flowcharts describe the processes for setting up pins for various functions. Begin the process from the start label in the ***GPIO Programming Model Flow (Part 1)*** figure. The first decision (GPIO or peripheral) determines how to program the `PORT_FER` register. Set the bit(s) corresponding to the pin(s) to 1 to enable peripheral functionality or to 0 to enable GPIO mode. For more information on setting up for peripheral functions, refer to Port Multiplexing Control.

If the pin is to be a GPIO pin, a subsequent series of decisions must be made that will impact how the `PORT_DATA`, `PORT_POL`, `PORT_DIR`, and `PORT_INEN` configuration registers must be programmed. Depending on the type of GPIO pin desired, some configurations do not apply and can have different meanings. The following paragraphs briefly describe the function of the different settings for each of the pin functions in the input, output, and open-drain GPIO modes. It is a best practice to use the SET or CLR versions of the PORT registers, where applicable, to effect changes on a pin-by-pin basis rather than on the full port.

For more detailed descriptions of the configurations, see PORT Register Descriptions.

For output mode, first clear the `PORT_DATA` register to set all the pins low. Then write the `PORT_DIR` register to define the direction of each pin (set the bits associated with the desired output pins to 1). In output mode, the other registers are not significant. The ***GPIO Programming Model Flow (Part 1)*** chart shows this flow starting at label 2.

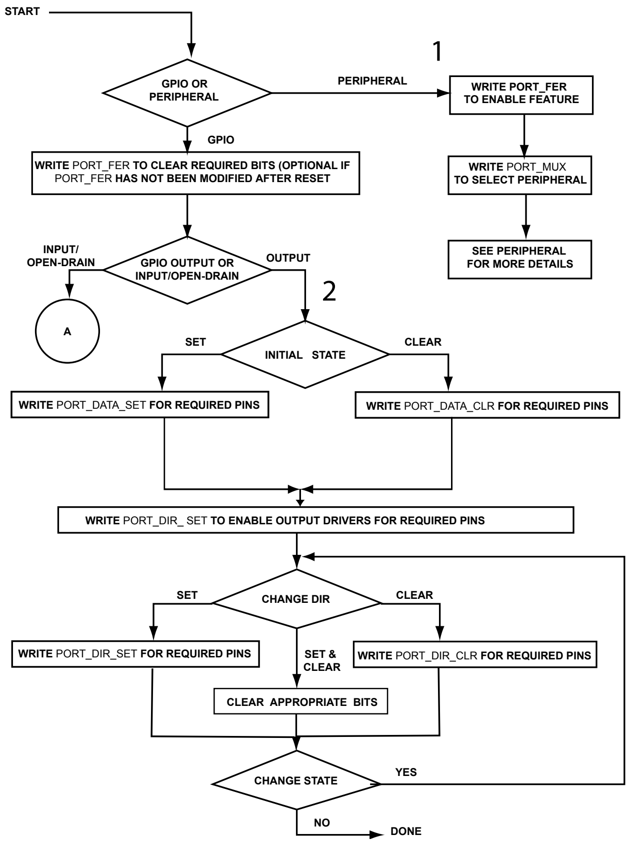

START
GPIO OR PERIPHERAL
PERIPHERAL
1
WRITE PORT_FER TO ENABLE FEATURE
WRITE PORT_MUX TO SELECT PERIPHERAL
SEE PERIPHERAL FOR MORE DETAILS
GPIO
WRITE PORT_FER TO CLEAR REQUIRED BITS (OPTIONAL IF PORT_FER HAS NOT BEEN MODIFIED AFTER RESET
INPUT/ OPEN-DRAIN
GPIO OUTPUT OR INPUT/OPEN-DRAIN
OUTPUT
A
2
SET
INITIAL STATE
CLEAR
WRITE PORT_DATA_SET FOR REQUIRED PINS
WRITE PORT_DATA_CLR FOR REQUIRED PINS
WRITE PORT_DIR_ SET TO ENABLE OUTPUT DRIVERS FOR REQUIRED PINS
SET
CHANGE DIR
CLEAR
WRITE PORT_DIR_SET FOR REQUIRED PINS
SET & CLEAR
WRITE PORT_DIR_CLR FOR REQUIRED PINS
CLEAR APPROPRIATE BITS
CHANGE STATE
YES
NO
DONE

Figure 14-4: GPIO Programming Model Flow (Part 1)

For input mode, first decide the polarity for each pin using the `PORT_POL` register. Program the `PORT_DIR` register to define the appropriate pins as inputs (write a 0 to the bit location associated with the pin). If interrupt requests are desired, configure the PINT module as shown in the ***GPIO Programming Model Flow (Part 3)*** figure starting at label B. Finally, write the `PORT_INEN` register to enable the associated input drivers. The ***GPIO Programming Model Flow (Part 2)*** chart shows this entire flow starting at label 3.

For open-drain mode, set all pins low by clearing the `PORT_DATA` register. Then, use the `PORT_INEN` register to enable the appropriate input drivers. Set the `PORT_DIR` register in this mode to indicate whether the pin is in an active state or not (active being 0). The ***GPIO Programming Model Flow (Part 2)*** chart shows this flow starting at label 4.

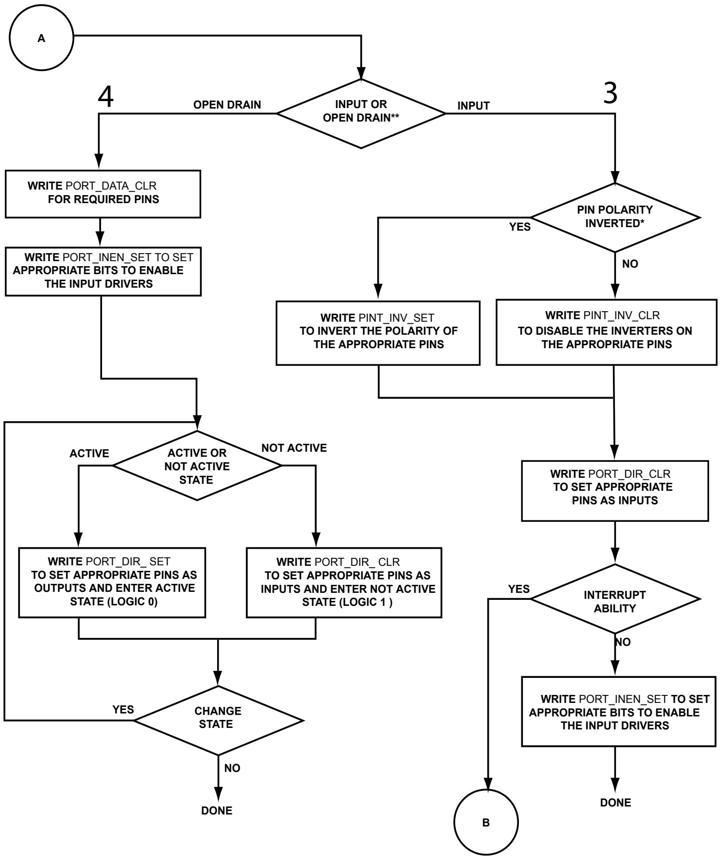

A
4
OPEN DRAIN
INPUT OR OPEN DRAIN**
INPUT
3
WRITE PORT_DATA_CLR FOR REQUIRED PINS
WRITE PORT_INEN_SET TO SET APPROPRIATE BITS TO ENABLE THE INPUT DRIVERS
PIN POLARITY INVERTED*
YES
NO
WRITE PINT_INV_SET TO INVERT THE POLARITY OF THE APPROPRIATE PINS
WRITE PINT_INV_CLR TO DISABLE THE INVERTERS ON THE APPROPRIATE PINS
ACTIVE
ACTIVE OR NOT ACTIVE STATE
NOT ACTIVE
WRITE PORT_DIR_CLR TO SET APPROPRIATE PINS AS INPUTS
WRITE PORT_DIR_ SET TO SET APPROPRIATE PINS AS OUTPUTS AND ENTER ACTIVE STATE (LOGIC 0)
WRITE PORT_DIR_ CLR TO SET APPROPRIATE PINS AS INPUTS AND ENTER NOT ACTIVE STATE (LOGIC 1)
YES
INTERRUPT ABILITY
NO
WRITE PORT_INEN_SET TO SET APPROPRIATE BITS TO ENABLE THE INPUT DRIVERS
YES
CHANGE STATE
NO
DONE
DONE
B

Figure 14-5: GPIO Programming Model Flow (Part 2)

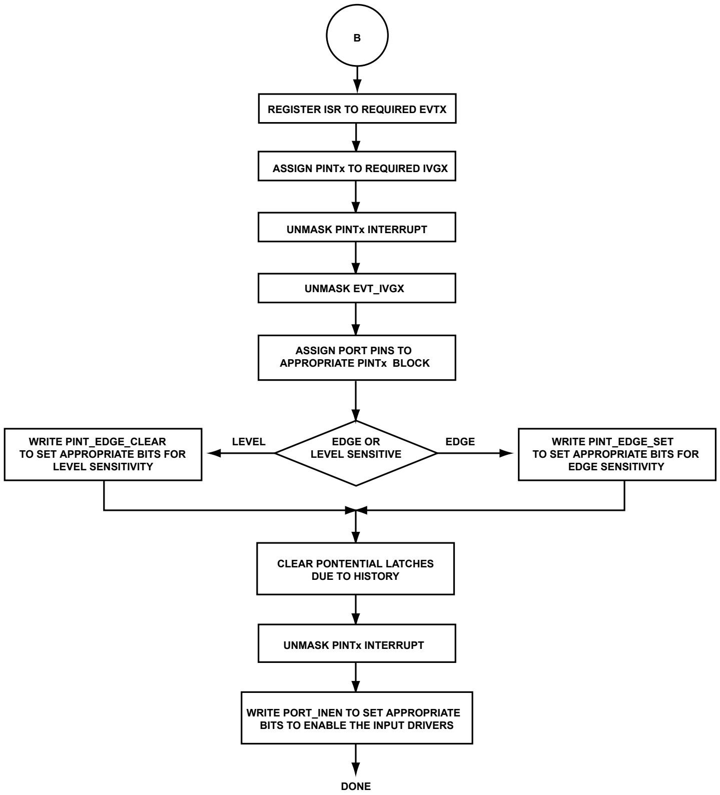

B
REGISTER ISR TO REQUIRED EVTX
ASSIGN PINTx TO REQUIRED IVGX
UNMASK PINTx INTERRUPT
UNMASK EVT_IVGX
ASSIGN PORT PINS TO APPROPRIATE PINTx BLOCK
EDGE OR LEVEL SENSITIVE
LEVEL
EDGE
WRITE PINT_EDGE_CLEAR TO SET APPROPRIATE BITS FOR LEVEL SENSITIVITY
WRITE PINT_EDGE_SET TO SET APPROPRIATE BITS FOR EDGE SENSITIVITY
CLEAR PONTENTIAL LATCHES DUE TO HISTORY
UNMASK PINTx INTERRUPT
WRITE PORT_INEN TO SET APPROPRIATE BITS TO ENABLE THE INPUT DRIVERS
DONE

Figure 14-6: GPIO Programming Model Flow (Part 3)

ADSP-SC58x PORT Register Descriptions

The General-Purpose Input/Output Port (PORT) contains the following registers.

Table 14-7: ADSP-SC58x PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register

Table 14-7: ADSP-SC58x PORT Register List (Continued)

Name	Description
`PORT_INEN`	Port x GPIO Input Enable Register
`PORT_INEN_CLR`	Port x GPIO Input Enable Clear Register
`PORT_INEN_SET`	Port x GPIO Input Enable Set Register
`PORT_LOCK`	Port x GPIO Lock Register
`PORT_MUX`	Port x Multiplexer Control Register
`PORT_POL`	Port x GPIO Polarity Invert Register
`PORT_POL_CLR`	Port x GPIO Polarity Invert Clear Register
`PORT_POL_SET`	Port x GPIO Polarity Invert Set Register

Port x GPIO Data Register

The operation of the PORT_DATA register depends on whether the bit/pin is in output mode or input mode. In both modes, a set bit in the PORT_DATA register corresponds to a signal high on a GPIO pin. A cleared bit in the PORT_DATA register corresponds to a signal low on a GPIO pin.

The PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers control the state of GPIO pins in output mode. To enable output mode (and output drivers), use the PORT_DIR_SET and PORT_DIR_CLR registers.

Writes to the PORT_DATA register affect the state of all pins of the port that are in output mode. To set or clear specific pins without impacting other pins of the port, use the PORT_DATA_SET and PORT_DATA_CLR registers.

When the GPIO pins are in input mode (input driver is enabled with the PORT_INEN register), reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the state of the respective GPIO pins.

Note that when the input driver is not enabled, reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the value previously written to the registers.

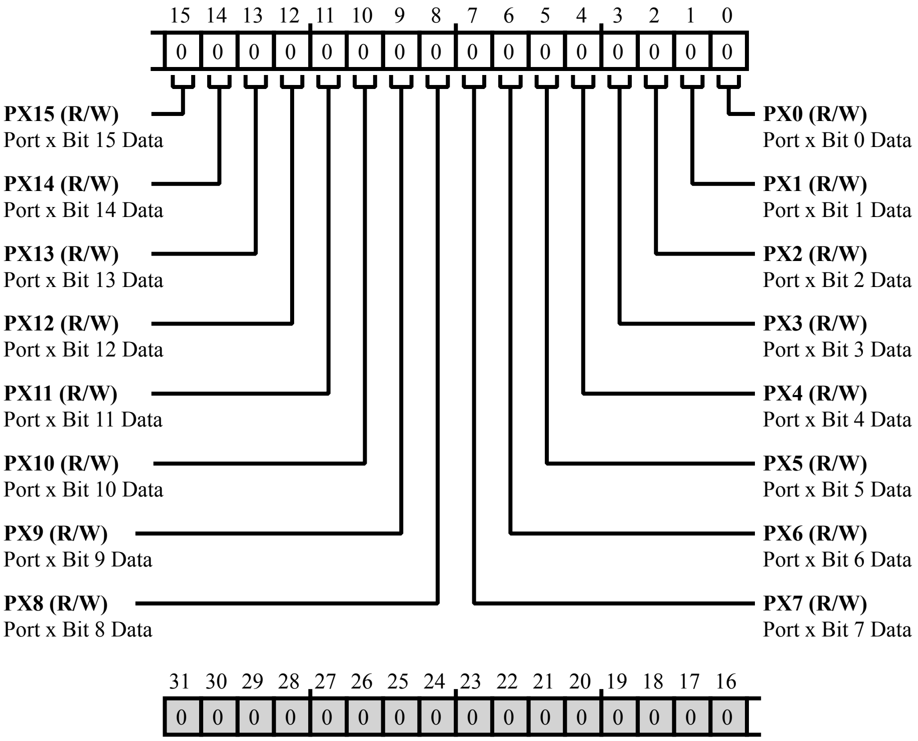

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Data
PX14 (R/W)
Port x Bit 14 Data
PX13 (R/W)
Port x Bit 13 Data
PX12 (R/W)
Port x Bit 12 Data
PX11 (R/W)
Port x Bit 11 Data
PX10 (R/W)
Port x Bit 10 Data
PX9 (R/W)
Port x Bit 9 Data
PX8 (R/W)
Port x Bit 8 Data
PX0 (R/W)
Port x Bit 0 Data
PX1 (R/W)
Port x Bit 1 Data
PX2 (R/W)
Port x Bit 2 Data
PX3 (R/W)
Port x Bit 3 Data
PX4 (R/W)
Port x Bit 4 Data
PX5 (R/W)
Port x Bit 5 Data
PX6 (R/W)
Port x Bit 6 Data
PX7 (R/W)
Port x Bit 7 Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-7: PORT_DATA Register Diagram

Table 14-8: PORT_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Data. The `PORT_DATA.PX15` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
14 (R/W)	PX14	Port x Bit 14 Data. The `PORT_DATA.PX14` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
13 (R/W)	PX13	Port x Bit 13 Data. The `PORT_DATA.PX13` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
12 (R/W)	PX12	Port x Bit 12 Data. The `PORT_DATA.PX12` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
11 (R/W)	PX11	Port x Bit 11 Data. The `PORT_DATA.PX11` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
10 (R/W)	PX10	Port x Bit 10 Data. The `PORT_DATA.PX10` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
9 (R/W)	PX9	Port x Bit 9 Data. The `PORT_DATA.PX9` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Table 14-8: PORT_DATA Register Fields (Continued)

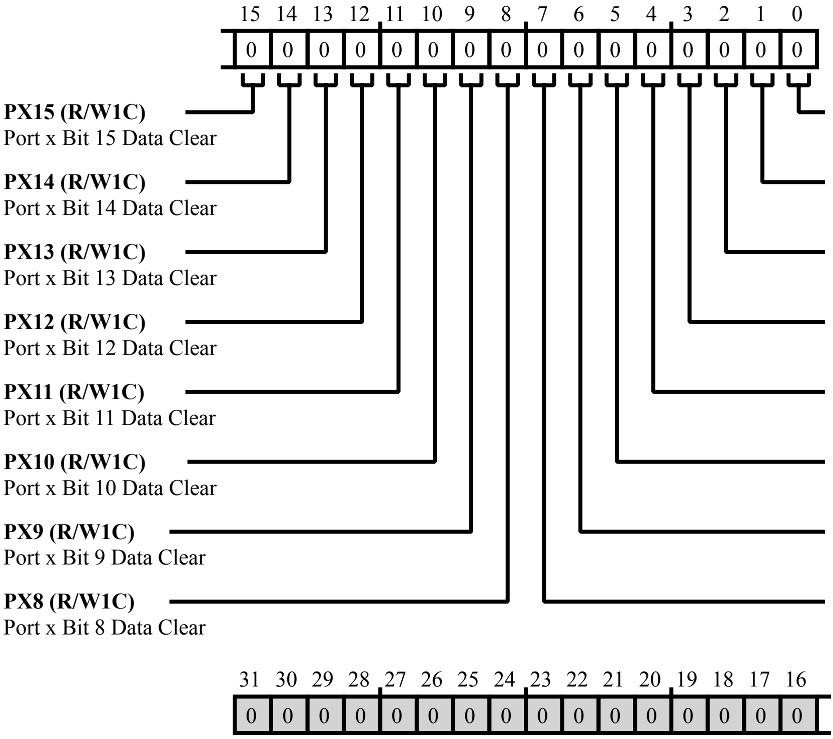
Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PX8	Port x Bit 8 Data. The `PORT_DATA.PX8` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
7 (R/W)	PX7	Port x Bit 7 Data. The `PORT_DATA.PX7` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
6 (R/W)	PX6	Port x Bit 6 Data. The `PORT_DATA.PX6` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
5 (R/W)	PX5	Port x Bit 5 Data. The `PORT_DATA.PX5` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
4 (R/W)	PX4	Port x Bit 4 Data. The `PORT_DATA.PX4` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
3 (R/W)	PX3	Port x Bit 3 Data. The `PORT_DATA.PX3` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
2 (R/W)	PX2	Port x Bit 2 Data. The `PORT_DATA.PX2` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Table 14-8: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PX1	Port x Bit 1 Data. The `PORT_DATA.PX1` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
0 (R/W)	PX0	Port x Bit 0 Data. The `PORT_DATA.PX0` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Port x GPIO Data Clear Register

The PORT_DATA_CLR register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the PORT_DATA register description.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1C)
Port x Bit 15 Data Clear
PX14 (R/W1C)
Port x Bit 14 Data Clear
PX13 (R/W1C)
Port x Bit 13 Data Clear
PX12 (R/W1C)
Port x Bit 12 Data Clear
PX11 (R/W1C)
Port x Bit 11 Data Clear
PX10 (R/W1C)
Port x Bit 10 Data Clear
PX9 (R/W1C)
Port x Bit 9 Data Clear
PX8 (R/W1C)
Port x Bit 8 Data Clear
PX0 (R/W1C)
Port x Bit 0 Data Clear
PX1 (R/W1C)
Port x Bit 1 Data Clear
PX2 (R/W1C)
Port x Bit 2 Data Clear
PX3 (R/W1C)
Port x Bit 3 Data Clear
PX4 (R/W1C)
Port x Bit 4 Data Clear
PX5 (R/W1C)
Port x Bit 5 Data Clear
PX6 (R/W1C)
Port x Bit 6 Data Clear
PX7 (R/W1C)
Port x Bit 7 Data Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-8: PORT_DATA_CLR Register Diagram

Table 14-9: PORT_DATA_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Data Clear. The `PORT_DATA_CLR.PX15` bit clears the pin without impacting other pins of the port.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear. The `PORT_DATA_CLR.PX14` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 14-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Data Clear. The `PORT_DATA_CLR.PX13` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
12 (R/W1C)	PX12	Port x Bit 12 Data Clear. The `PORT_DATA_CLR.PX12` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
11 (R/W1C)	PX11	Port x Bit 11 Data Clear. The `PORT_DATA_CLR.PX11` bit clears the pin without impacting other pins of the port.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear. The `PORT_DATA_CLR.PX10` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
9 (R/W1C)	PX9	Port x Bit 9 Data Clear. The `PORT_DATA_CLR.PX9` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
8 (R/W1C)	PX8	Port x Bit 8 Data Clear. The `PORT_DATA_CLR.PX8` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 14-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PX7	Port x Bit 7 Data Clear. The `PORT_DATA_CLR.PX7` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
6 (R/W1C)	PX6	Port x Bit 6 Data Clear. The `PORT_DATA_CLR.PX6` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
5 (R/W1C)	PX5	Port x Bit 5 Data Clear. The `PORT_DATA_CLR.PX5` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
4 (R/W1C)	PX4	Port x Bit 4 Data Clear. The `PORT_DATA_CLR.PX4` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
3 (R/W1C)	PX3	Port x Bit 3 Data Clear. The `PORT_DATA_CLR.PX3` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
2 (R/W1C)	PX2	Port x Bit 2 Data Clear. The `PORT_DATA_CLR.PX2` bit clears the pin without impacting other pins of the port.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 14-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PX1	Port x Bit 1 Data Clear. The `PORT_DATA_CLR.PX1` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
0 (R/W1C)	PX0	Port x Bit 0 Data Clear. The `PORT_DATA_CLR.PX0` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Port x GPIO Data Set Register

The `PORT_DATA_SET` register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the `PORT_DATA` register description.

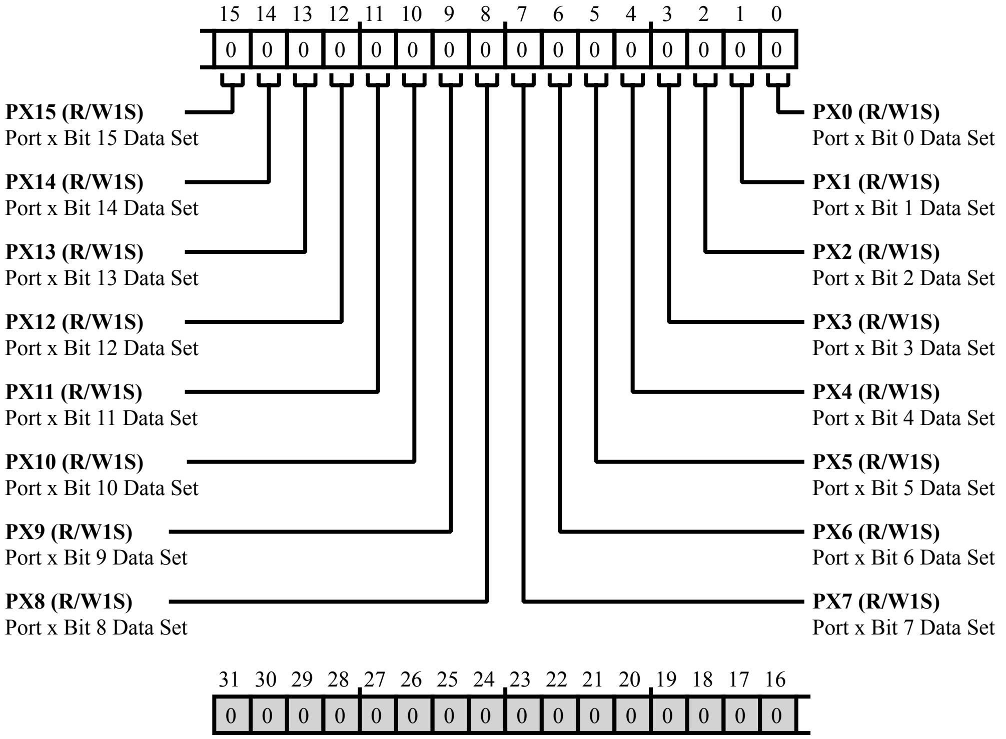

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1S)
Port x Bit 15 Data Set
PX14 (R/W1S)
Port x Bit 14 Data Set
PX13 (R/W1S)
Port x Bit 13 Data Set
PX12 (R/W1S)
Port x Bit 12 Data Set
PX11 (R/W1S)
Port x Bit 11 Data Set
PX10 (R/W1S)
Port x Bit 10 Data Set
PX9 (R/W1S)
Port x Bit 9 Data Set
PX8 (R/W1S)
Port x Bit 8 Data Set
PX0 (R/W1S)
Port x Bit 0 Data Set
PX1 (R/W1S)
Port x Bit 1 Data Set
PX2 (R/W1S)
Port x Bit 2 Data Set
PX3 (R/W1S)
Port x Bit 3 Data Set
PX4 (R/W1S)
Port x Bit 4 Data Set
PX5 (R/W1S)
Port x Bit 5 Data Set
PX6 (R/W1S)
Port x Bit 6 Data Set
PX7 (R/W1S)
Port x Bit 7 Data Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-9: PORT_DATA_SET Register Diagram

Table 14-10: PORT_DATA_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
14 (R/W1S)	PX14	Port x Bit 14 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
13 (R/W1S)	PX13	Port x Bit 13 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Table 14-10: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
11 (R/W1S)	PX11	Port x Bit 11 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit.
10 (R/W1S)	PX10	Port x Bit 10 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
9 (R/W1S)	PX9	Port x Bit 9 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
8 (R/W1S)	PX8	Port x Bit 8 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
7 (R/W1S)	PX7	Port x Bit 7 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
6 (R/W1S)	PX6	Port x Bit 6 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
5 (R/W1S)	PX5	Port x Bit 5 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
4 (R/W1S)	PX4	Port x Bit 4 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
3 (R/W1S)	PX3	Port x Bit 3 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Table 14-10: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
1 (R/W1S)	PX1	Port x Bit 1 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
0 (R/W1S)	PX0	Port x Bit 0 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Port x GPIO Output Toggle Register

The `PORT_DATA_TGL` register permits toggling the state of output GPIO pins. Setting bits in the `PORT_DATA_TGL` register affects the state of specific pins without impacting other pins of the port.

Reading the `PORT_DATA_TGL` returns the state of the `PORT_DATA` register output pin state, but does not return the input pin/signal state.

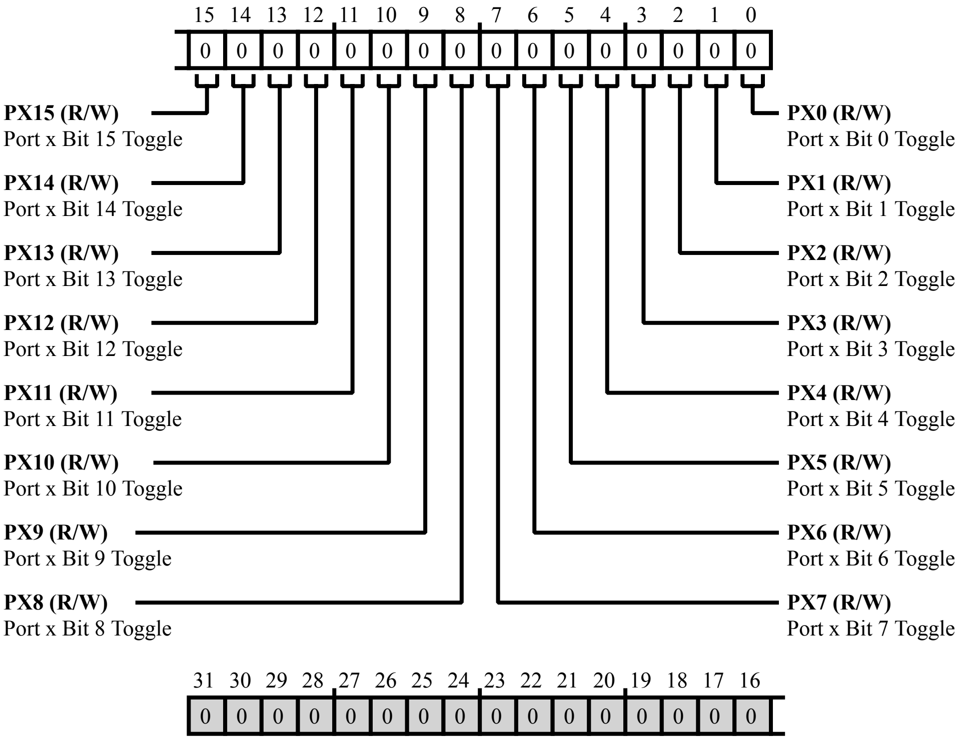

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Toggle
PX14 (R/W)
Port x Bit 14 Toggle
PX13 (R/W)
Port x Bit 13 Toggle
PX12 (R/W)
Port x Bit 12 Toggle
PX11 (R/W)
Port x Bit 11 Toggle
PX10 (R/W)
Port x Bit 10 Toggle
PX9 (R/W)
Port x Bit 9 Toggle
PX8 (R/W)
Port x Bit 8 Toggle
PX0 (R/W)
Port x Bit 0 Toggle
PX1 (R/W)
Port x Bit 1 Toggle
PX2 (R/W)
Port x Bit 2 Toggle
PX3 (R/W)
Port x Bit 3 Toggle
PX4 (R/W)
Port x Bit 4 Toggle
PX5 (R/W)
Port x Bit 5 Toggle
PX6 (R/W)
Port x Bit 6 Toggle
PX7 (R/W)
Port x Bit 7 Toggle
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-10: PORT_DATA_TGL Register Diagram

Table 14-11: PORT_DATA_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Toggle. The `PORT_DATA_TGL.PX15` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit.
14 (R/W)	PX14	Port x Bit 14 Toggle. The `PORT_DATA_TGL.PX14` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Table 14-11: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Toggle. The `PORT_DATA_TGL.PX13` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
12 (R/W)	PX12	Port x Bit 12 Toggle. The `PORT_DATA_TGL.PX12` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
11 (R/W)	PX11	Port x Bit 11 Toggle. The `PORT_DATA_TGL.PX11` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
10 (R/W)	PX10	Port x Bit 10 Toggle. The `PORT_DATA_TGL.PX10` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
9 (R/W)	PX9	Port x Bit 9 Toggle. The `PORT_DATA_TGL.PX9` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
8 (R/W)	PX8	Port x Bit 8 Toggle. The `PORT_DATA_TGL.PX8` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
7 (R/W)	PX7	Port x Bit 7 Toggle. The `PORT_DATA_TGL.PX7` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Table 14-11: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	PX6	Port x Bit 6 Toggle. The `PORT_DATA_TGL.PX6` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
5 (R/W)	PX5	Port x Bit 5 Toggle. The `PORT_DATA_TGL.PX5` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
4 (R/W)	PX4	Port x Bit 4 Toggle. The `PORT_DATA_TGL.PX4` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
3 (R/W)	PX3	Port x Bit 3 Toggle. The `PORT_DATA_TGL.PX3` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
2 (R/W)	PX2	Port x Bit 2 Toggle. The `PORT_DATA_TGL.PX2` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
1 (R/W)	PX1	Port x Bit 1 Toggle. The `PORT_DATA_TGL.PX1` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
0 (R/W)	PX0	Port x Bit 0 Toggle. The `PORT_DATA_TGL.PX0` bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Port x GPIO Direction Register

The `PORT_DIR`, `PORT_DIR_SET`, and `PORT_DIR_CLR` registers select output or input mode for GPIO pins and enable output drivers. Use the `PORT_INEN`, `PORT_INEN_SET`, and `PORT_INEN_CLR` registers to enable or disable input drivers.

Writes to the `PORT_DIR` register affect the state of all pins of the port. To select a direction for specific pins without impacting other pins of the port, use the `PORT_DIR_SET` and `PORT_DIR_CLR` registers.

Setting a bit in the `PORT_DIR` register enables output mode on the corresponding a GPIO pin. Clearing a bit in the `PORT_DIR` register disables output mode on the corresponding GPIO pin.

Input Mode - The default mode of every GPIO pin after reset is the input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the corresponding bits in the `PORT_INEN` register. When enabled, a read from the `PORT_DATA` register returns the logical state of the input pin. The input signal does not overwrite the state of the bit used for the output case. That state can only be altered by software. If the input driver is enabled, a write to the `PORT_DATA` register can alter the state of the bit, but the change cannot be read back.

Output Mode - Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the corresponding bits in the `PORT_DIR`, `PORT_DIR_SET`, or `PORT_DIR_CLR` registers. By using the `PORT_DIR_SET` and `PORT_DIR_CLR` registers, the direction of the signal flow of individual GPIO pins can be altered by separate software threads without mutually impacting other GPIOs on the same port. Both registers return the same value when read. Because the state of the GPIO output can already be controlled before the output driver is enabled, it is recommended to first set or clear the bit (using the `PORT_DATA`, `PORT_DATA_SET`, or `PORT_DATA_CLR` registers) to avoid any volatile levels on the output.

Open-Drain Mode - Every GPIO can also be used in open-drain mode. To accomplish this, first, clear the respective bit in the `PORT_DATA` or `PORT_DATA_CLR` register. Then, set the one bit in the `PORT_INEN` register. Reads from the `PORT_DATA` register then return the status from the pin and do not return the state of the internal flip-flop. By toggling the output driver through the `PORT_DIR_SET` and `PORT_DIR_CLR` register pair, the output signal can be pulled low or three-stated as required. Note that the polarity of the driven signal can be inverted when the internal flip-flop is set instead. When a GPIO port is used in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pin.

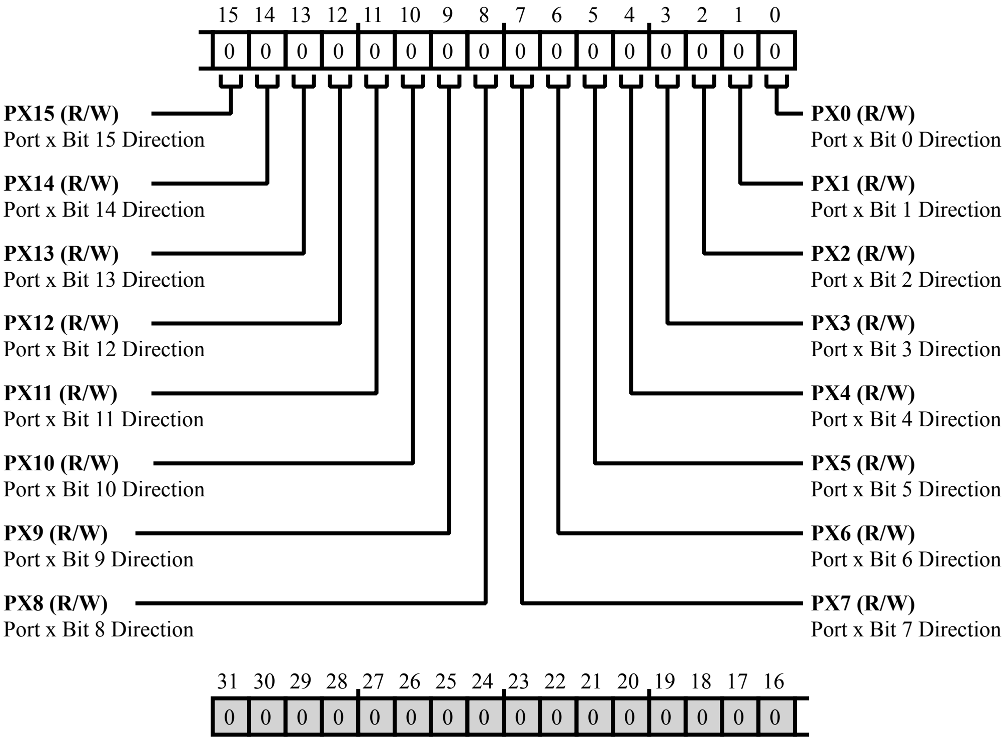

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Direction
PX14 (R/W)
Port x Bit 14 Direction
PX13 (R/W)
Port x Bit 13 Direction
PX12 (R/W)
Port x Bit 12 Direction
PX11 (R/W)
Port x Bit 11 Direction
PX10 (R/W)
Port x Bit 10 Direction
PX9 (R/W)
Port x Bit 9 Direction
PX8 (R/W)
Port x Bit 8 Direction
PX0 (R/W)
Port x Bit 0 Direction
PX1 (R/W)
Port x Bit 1 Direction
PX2 (R/W)
Port x Bit 2 Direction
PX3 (R/W)
Port x Bit 3 Direction
PX4 (R/W)
Port x Bit 4 Direction
PX5 (R/W)
Port x Bit 5 Direction
PX6 (R/W)
Port x Bit 6 Direction
PX7 (R/W)
Port x Bit 7 Direction
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-11: PORT_DIR Register Diagram

Table 14-12: PORT_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
14 (R/W)	PX14	Port x Bit 14 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
13 (R/W)	PX13	Port x Bit 13 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
12 (R/W)	PX12	Port x Bit 12 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Table 14-12: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	PX11	Port x Bit 11 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
10 (R/W)	PX10	Port x Bit 10 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
9 (R/W)	PX9	Port x Bit 9 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
8 (R/W)	PX8	Port x Bit 8 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
7 (R/W)	PX7	Port x Bit 7 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
6 (R/W)	PX6	Port x Bit 6 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
5 (R/W)	PX5	Port x Bit 5 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
4 (R/W)	PX4	Port x Bit 4 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
3 (R/W)	PX3	Port x Bit 3 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
2 (R/W)	PX2	Port x Bit 2 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Table 14-12: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PX1	Port x Bit 1 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
0 (R/W)	PX0	Port x Bit 0 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Port x GPIO Direction Clear Register

The `PORT_DIR_CLR` register disables output mode and disables the output drivers for GPIO pins. For more information, see the `PORT_DIR` register description.

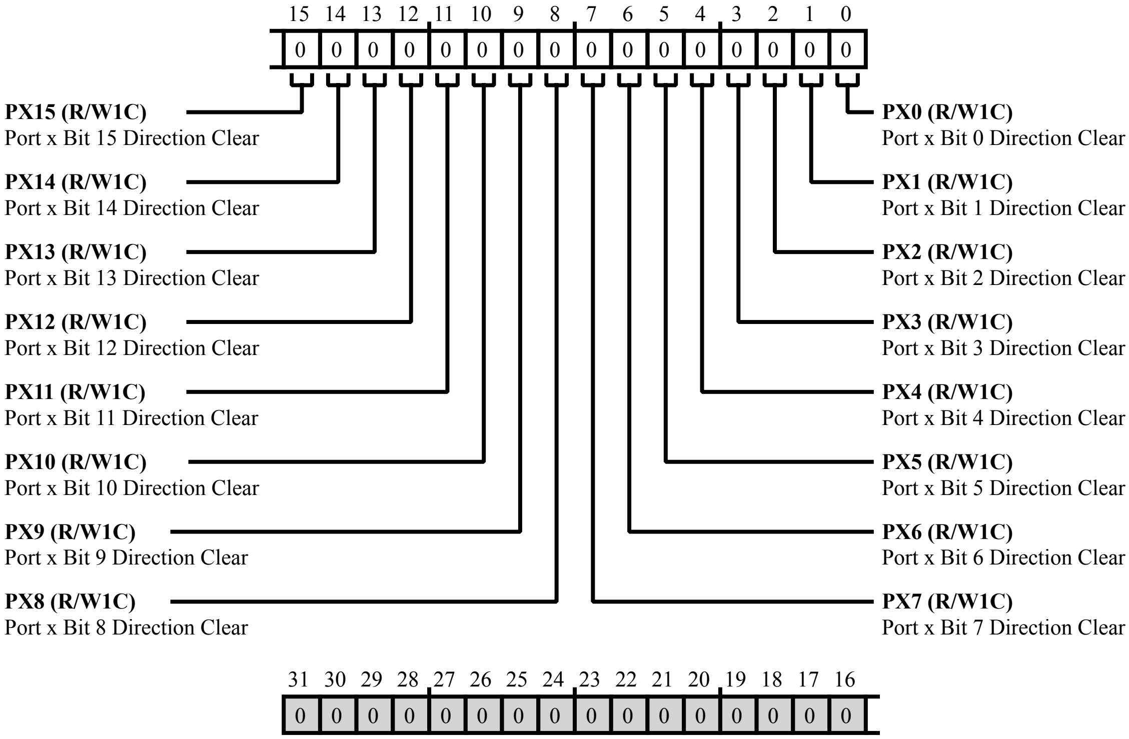

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1C)
Port x Bit 15 Direction Clear
PX14 (R/W1C)
Port x Bit 14 Direction Clear
PX13 (R/W1C)
Port x Bit 13 Direction Clear
PX12 (R/W1C)
Port x Bit 12 Direction Clear
PX11 (R/W1C)
Port x Bit 11 Direction Clear
PX10 (R/W1C)
Port x Bit 10 Direction Clear
PX9 (R/W1C)
Port x Bit 9 Direction Clear
PX8 (R/W1C)
Port x Bit 8 Direction Clear
PX0 (R/W1C)
Port x Bit 0 Direction Clear
PX1 (R/W1C)
Port x Bit 1 Direction Clear
PX2 (R/W1C)
Port x Bit 2 Direction Clear
PX3 (R/W1C)
Port x Bit 3 Direction Clear
PX4 (R/W1C)
Port x Bit 4 Direction Clear
PX5 (R/W1C)
Port x Bit 5 Direction Clear
PX6 (R/W1C)
Port x Bit 6 Direction Clear
PX7 (R/W1C)
Port x Bit 7 Direction Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-12: PORT_DIR_CLR Register Diagram

Table 14-13: PORT_DIR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Direction Clear. The `PORT_DIR_CLR.PX15` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
14 (R/W1C)	PX14	Port x Bit 14 Direction Clear. The `PORT_DIR_CLR.PX14` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 14-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Direction Clear. The `PORT_DIR_CLR.PX13` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
12 (R/W1C)	PX12	Port x Bit 12 Direction Clear. The `PORT_DIR_CLR.PX12` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
11 (R/W1C)	PX11	Port x Bit 11 Direction Clear. The `PORT_DIR_CLR.PX11` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
10 (R/W1C)	PX10	Port x Bit 10 Direction Clear. The `PORT_DIR_CLR.PX10` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
9 (R/W1C)	PX9	Port x Bit 9 Direction Clear. The `PORT_DIR_CLR.PX9` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
8 (R/W1C)	PX8	Port x Bit 8 Direction Clear. The `PORT_DIR_CLR.PX8` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 14-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PX7	Port x Bit 7 Direction Clear. The `PORT_DIR_CLR.PX7` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
6 (R/W1C)	PX6	Port x Bit 6 Direction Clear. The `PORT_DIR_CLR.PX6` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
5 (R/W1C)	PX5	Port x Bit 5 Direction Clear. The `PORT_DIR_CLR.PX5` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
4 (R/W1C)	PX4	Port x Bit 4 Direction Clear. The `PORT_DIR_CLR.PX4` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
3 (R/W1C)	PX3	Port x Bit 3 Direction Clear. The `PORT_DIR_CLR.PX3` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
2 (R/W1C)	PX2	Port x Bit 2 Direction Clear. The `PORT_DIR_CLR.PX2` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 14-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PX1	Port x Bit 1 Direction Clear. The `PORT_DIR_CLR.PX1` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
0 (R/W1C)	PX0	Port x Bit 0 Direction Clear. The `PORT_DIR_CLR.PX0` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Port x GPIO Direction Set Register

The `PORT_DIR_SET` register enables output mode and output drivers for GPIO pins. For more information, see the `PORT_DIR` register description.

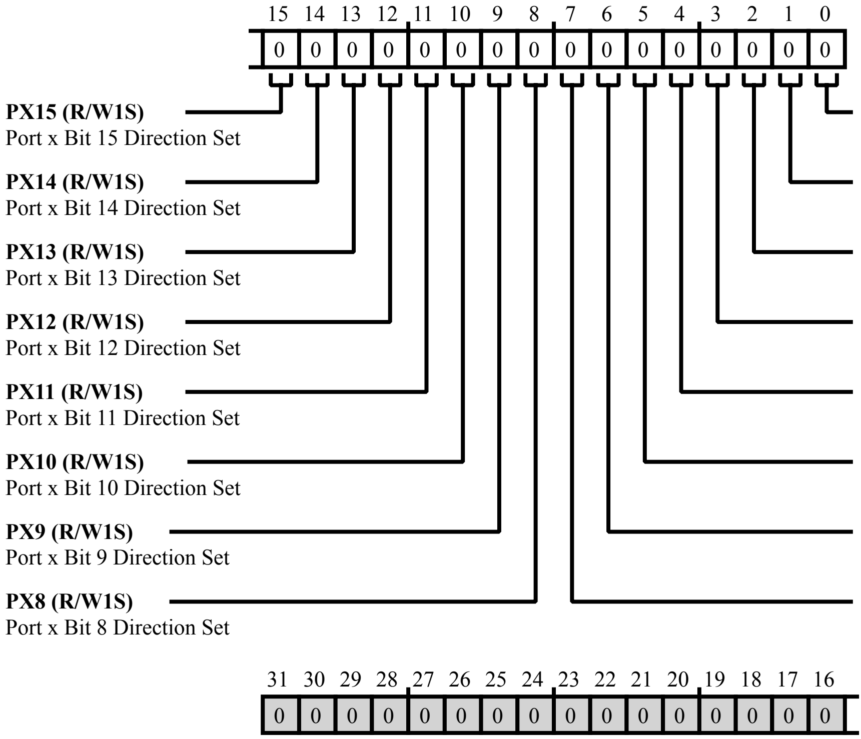

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1S)
Port x Bit 15 Direction Set
PX14 (R/W1S)
Port x Bit 14 Direction Set
PX13 (R/W1S)
Port x Bit 13 Direction Set
PX12 (R/W1S)
Port x Bit 12 Direction Set
PX11 (R/W1S)
Port x Bit 11 Direction Set
PX10 (R/W1S)
Port x Bit 10 Direction Set
PX9 (R/W1S)
Port x Bit 9 Direction Set
PX8 (R/W1S)
Port x Bit 8 Direction Set
PX0 (R/W1S)
Port x Bit 0 Direction Set
PX1 (R/W1S)
Port x Bit 1 Direction Set
PX2 (R/W1S)
Port x Bit 2 Direction Set
PX3 (R/W1S)
Port x Bit 3 Direction Set
PX4 (R/W1S)
Port x Bit 4 Direction Set
PX5 (R/W1S)
Port x Bit 5 Direction Set
PX6 (R/W1S)
Port x Bit 6 Direction Set
PX7 (R/W1S)
Port x Bit 7 Direction Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-13: PORT_DIR_SET Register Diagram

Table 14-14: PORT_DIR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Direction Set. The `PORT_DIR_SET.PX15` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
14 (R/W1S)	PX14	Port x Bit 14 Direction Set. The `PORT_DIR_SET.PX14` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 14-14: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Direction Set. The `PORT_DIR_SET.PX13` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
12 (R/W1S)	PX12	Port x Bit 12 Direction Set. The `PORT_DIR_SET.PX12` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
11 (R/W1S)	PX11	Port x Bit 11 Direction Set. The `PORT_DIR_SET.PX11` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
10 (R/W1S)	PX10	Port x Bit 10 Direction Set. The `PORT_DIR_SET.PX10` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
9 (R/W1S)	PX9	Port x Bit 9 Direction Set. The `PORT_DIR_SET.PX9` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
8 (R/W1S)	PX8	Port x Bit 8 Direction Set. The `PORT_DIR_SET.PX8` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
7 (R/W1S)	PX7	Port x Bit 7 Direction Set. The `PORT_DIR_SET.PX7` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 14-14: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Direction Set. The `PORT_DIR_SET.PX6` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
5 (R/W1S)	PX5	Port x Bit 5 Direction Set. The `PORT_DIR_SET.PX5` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
4 (R/W1S)	PX4	Port x Bit 4 Direction Set. The `PORT_DIR_SET.PX4` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
3 (R/W1S)	PX3	Port x Bit 3 Direction Set. The `PORT_DIR_SET.PX3` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
2 (R/W1S)	PX2	Port x Bit 2 Direction Set. The `PORT_DIR_SET.PX2` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
1 (R/W1S)	PX1	Port x Bit 1 Direction Set. The `PORT_DIR_SET.PX1` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
0 (R/W1S)	PX0	Port x Bit 0 Direction Set. The `PORT_DIR_SET.PX0` bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Port x Function Enable Register

The `PORT_FER` register bits indicate each port bit's operating mode: general purpose I/O mode or peripheral mode. After reset, all pins default to GPIO mode. Setting a bit in the `PORT_FER` registers enables a peripheral module to take ownership of the pin. The function enable bits impact output control only. Regardless of the setting of the function enable bits, both GPIO and peripherals can still sense the pin input. After a function is enabled, it is up to the `PORT_MUX` registers as to which peripheral takes control.

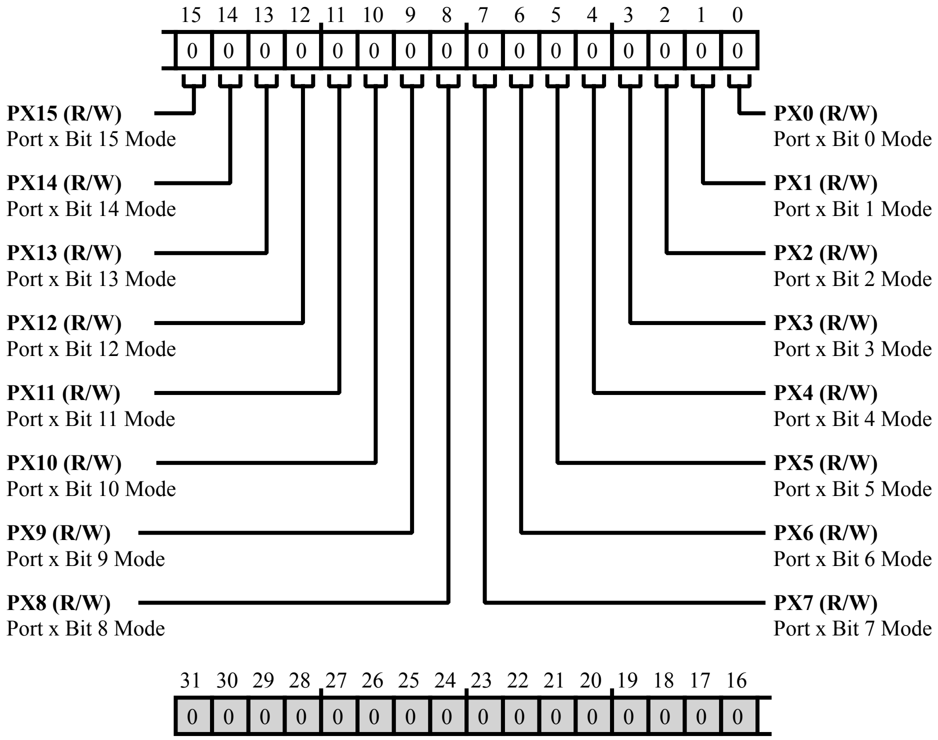

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Mode
PX14 (R/W)
Port x Bit 14 Mode
PX13 (R/W)
Port x Bit 13 Mode
PX12 (R/W)
Port x Bit 12 Mode
PX11 (R/W)
Port x Bit 11 Mode
PX10 (R/W)
Port x Bit 10 Mode
PX9 (R/W)
Port x Bit 9 Mode
PX8 (R/W)
Port x Bit 8 Mode
PX0 (R/W)
Port x Bit 0 Mode
PX1 (R/W)
Port x Bit 1 Mode
PX2 (R/W)
Port x Bit 2 Mode
PX3 (R/W)
Port x Bit 3 Mode
PX4 (R/W)
Port x Bit 4 Mode
PX5 (R/W)
Port x Bit 5 Mode
PX6 (R/W)
Port x Bit 6 Mode
PX7 (R/W)
Port x Bit 7 Mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-14: PORT_FER Register Diagram

Table 14-15: PORT_FER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Mode. The `PORT_FER.PX15` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
14 (R/W)	PX14	Port x Bit 14 Mode. The `PORT_FER.PX14` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Table 14-15: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Mode. The `PORT_FER.PX13` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
12 (R/W)	PX12	Port x Bit 12 Mode. The `PORT_FER.PX12` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
11 (R/W)	PX11	Port x Bit 11 Mode. The `PORT_FER.PX11` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
10 (R/W)	PX10	Port x Bit 10 Mode. The `PORT_FER.PX10` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
9 (R/W)	PX9	Port x Bit 9 Mode. The `PORT_FER.PX9` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
8 (R/W)	PX8	Port x Bit 8 Mode. The `PORT_FER.PX8` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
7 (R/W)	PX7	Port x Bit 7 Mode. The `PORT_FER.PX7` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Table 14-15: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	PX6	Port x Bit 6 Mode. The `PORT_FER.PX6` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
5 (R/W)	PX5	Port x Bit 5 Mode. The `PORT_FER.PX5` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
4 (R/W)	PX4	Port x Bit 4 Mode. The `PORT_FER.PX4` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
3 (R/W)	PX3	Port x Bit 3 Mode. The `PORT_FER.PX3` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
2 (R/W)	PX2	Port x Bit 2 Mode. The `PORT_FER.PX2` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
1 (R/W)	PX1	Port x Bit 1 Mode. The `PORT_FER.PX1` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
0 (R/W)	PX0	Port x Bit 0 Mode. The `PORT_FER.PX0` bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Port x Function Enable Clear Register

The `PORT_FER_CLR` register permits enabling GPIO mode for each bit and corresponding GPIO pin. Writing 1 to a bit in `PORT_FER_CLR` enables GPIO mode for the corresponding pin.

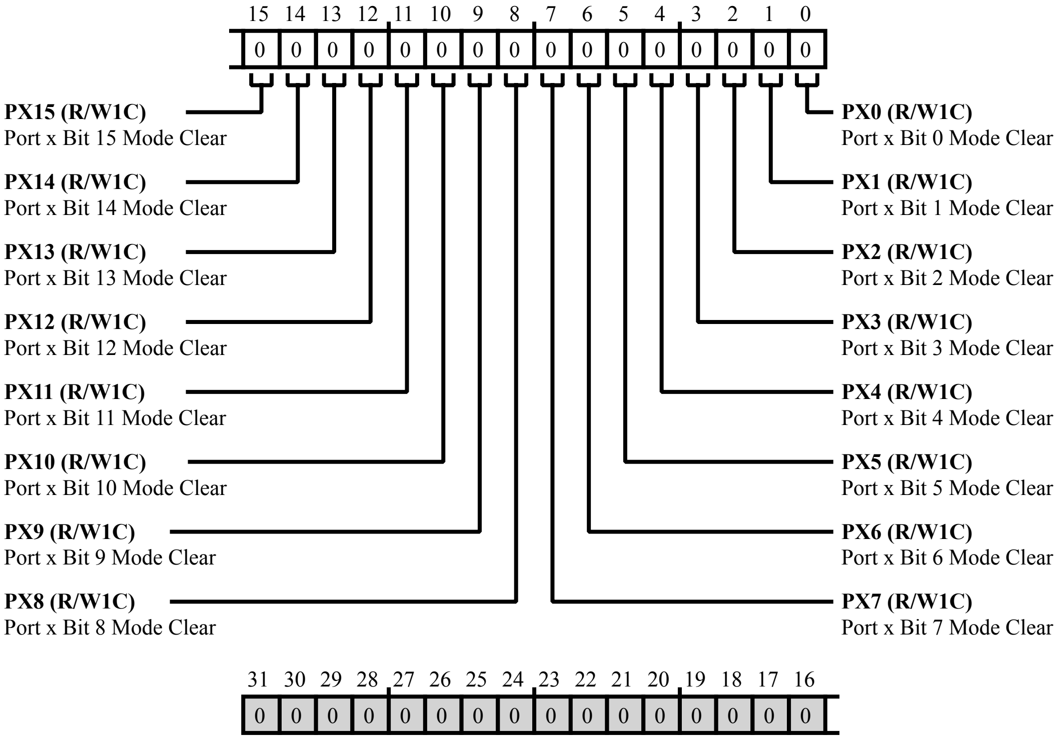

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1C)
Port x Bit 15 Mode Clear
PX14 (R/W1C)
Port x Bit 14 Mode Clear
PX13 (R/W1C)
Port x Bit 13 Mode Clear
PX12 (R/W1C)
Port x Bit 12 Mode Clear
PX11 (R/W1C)
Port x Bit 11 Mode Clear
PX10 (R/W1C)
Port x Bit 10 Mode Clear
PX9 (R/W1C)
Port x Bit 9 Mode Clear
PX8 (R/W1C)
Port x Bit 8 Mode Clear
PX0 (R/W1C)
Port x Bit 0 Mode Clear
PX1 (R/W1C)
Port x Bit 1 Mode Clear
PX2 (R/W1C)
Port x Bit 2 Mode Clear
PX3 (R/W1C)
Port x Bit 3 Mode Clear
PX4 (R/W1C)
Port x Bit 4 Mode Clear
PX5 (R/W1C)
Port x Bit 5 Mode Clear
PX6 (R/W1C)
Port x Bit 6 Mode Clear
PX7 (R/W1C)
Port x Bit 7 Mode Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-15: PORT_FER_CLR Register Diagram

Table 14-16: PORT_FER_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Mode Clear. The `PORT_FER_CLR.PX15` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
14 (R/W1C)	PX14	Port x Bit 14 Mode Clear. The `PORT_FER_CLR.PX14` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 14-16: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Mode Clear. The `PORT_FER_CLR.PX13` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
12 (R/W1C)	PX12	Port x Bit 12 Mode Clear. The `PORT_FER_CLR.PX12` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
11 (R/W1C)	PX11	Port x Bit 11 Mode Clear. The `PORT_FER_CLR.PX11` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
10 (R/W1C)	PX10	Port x Bit 10 Mode Clear. The `PORT_FER_CLR.PX10` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
9 (R/W1C)	PX9	Port x Bit 9 Mode Clear. The `PORT_FER_CLR.PX9` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
8 (R/W1C)	PX8	Port x Bit 8 Mode Clear. The `PORT_FER_CLR.PX8` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
7 (R/W1C)	PX7	Port x Bit 7 Mode Clear. The `PORT_FER_CLR.PX7` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 14-16: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	PX6	Port x Bit 6 Mode Clear. The `PORT_FER_CLR.PX6` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
5 (R/W1C)	PX5	Port x Bit 5 Mode Clear. The `PORT_FER_CLR.PX5` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
4 (R/W1C)	PX4	Port x Bit 4 Mode Clear. The `PORT_FER_CLR.PX4` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
3 (R/W1C)	PX3	Port x Bit 3 Mode Clear. The `PORT_FER_CLR.PX3` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
2 (R/W1C)	PX2	Port x Bit 2 Mode Clear. The `PORT_FER_CLR.PX2` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
1 (R/W1C)	PX1	Port x Bit 1 Mode Clear. The `PORT_FER_CLR.PX1` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
0 (R/W1C)	PX0	Port x Bit 0 Mode Clear. The `PORT_FER_CLR.PX0` bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Port x Function Enable Set Register

The `PORT_FER_SET` register permits enabling peripheral mode for each bit and corresponding GPIO pin. Writing 1 to a bit in `PORT_FER_SET` enables peripheral mode for the corresponding pin.

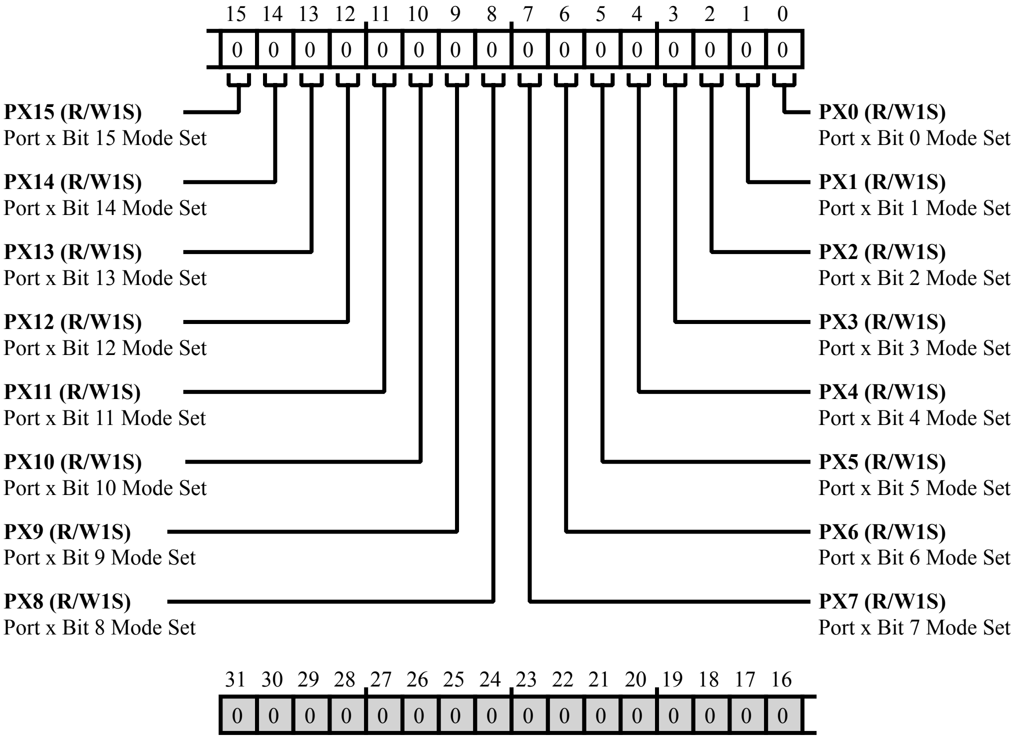

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1S)
Port x Bit 15 Mode Set
PX14 (R/W1S)
Port x Bit 14 Mode Set
PX13 (R/W1S)
Port x Bit 13 Mode Set
PX12 (R/W1S)
Port x Bit 12 Mode Set
PX11 (R/W1S)
Port x Bit 11 Mode Set
PX10 (R/W1S)
Port x Bit 10 Mode Set
PX9 (R/W1S)
Port x Bit 9 Mode Set
PX8 (R/W1S)
Port x Bit 8 Mode Set
PX0 (R/W1S)
Port x Bit 0 Mode Set
PX1 (R/W1S)
Port x Bit 1 Mode Set
PX2 (R/W1S)
Port x Bit 2 Mode Set
PX3 (R/W1S)
Port x Bit 3 Mode Set
PX4 (R/W1S)
Port x Bit 4 Mode Set
PX5 (R/W1S)
Port x Bit 5 Mode Set
PX6 (R/W1S)
Port x Bit 6 Mode Set
PX7 (R/W1S)
Port x Bit 7 Mode Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-16: PORT_FER_SET Register Diagram

Table 14-17: PORT_FER_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Mode Set. The `PORT_FER_SET.PX15` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
14 (R/W1S)	PX14	Port x Bit 14 Mode Set. The `PORT_FER_SET.PX14` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 14-17: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Mode Set. The `PORT_FER_SET.PX13` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
12 (R/W1S)	PX12	Port x Bit 12 Mode Set. The `PORT_FER_SET.PX12` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
11 (R/W1S)	PX11	Port x Bit 11 Mode Set. The `PORT_FER_SET.PX11` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
10 (R/W1S)	PX10	Port x Bit 10 Mode Set. The `PORT_FER_SET.PX10` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
9 (R/W1S)	PX9	Port x Bit 9 Mode Set. The `PORT_FER_SET.PX9` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
8 (R/W1S)	PX8	Port x Bit 8 Mode Set. The `PORT_FER_SET.PX8` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
7 (R/W1S)	PX7	Port x Bit 7 Mode Set. The `PORT_FER_SET.PX7` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 14-17: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Mode Set. The `PORT_FER_SET.PX6` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
5 (R/W1S)	PX5	Port x Bit 5 Mode Set. The `PORT_FER_SET.PX5` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
4 (R/W1S)	PX4	Port x Bit 4 Mode Set. The `PORT_FER_SET.PX4` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
3 (R/W1S)	PX3	Port x Bit 3 Mode Set. The `PORT_FER_SET.PX3` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
2 (R/W1S)	PX2	Port x Bit 2 Mode Set. The `PORT_FER_SET.PX2` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
1 (R/W1S)	PX1	Port x Bit 1 Mode Set. The `PORT_FER_SET.PX1` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
0 (R/W1S)	PX0	Port x Bit 0 Mode Set. The `PORT_FER_SET.PX0` bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Port x GPIO Input Enable Register

The PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers enable or disable input drivers, which are required for using a GPIO pin in input mode.

Writes to the PORT_INEN register affect the input drivers for all pins of the port. To set or clear specific pin drivers without impacting other pin drivers of the port, use the PORT_INEN_SET and PORT_INEN_CLR registers.

If the input is enabled, reads from the PORT_DATA, PORT_DATA_SET, or PORT_DATA_CLR registers return the state of the pins. However, the state of the output is not overwritten by the input. It is altered by software writes only. Input and output drivers can be enabled at the same time. In this case, a read of the data register returns the true value of the data register and not the pin state.

For more information, see the PORT_DATA register description and the PORT_DIR register description.

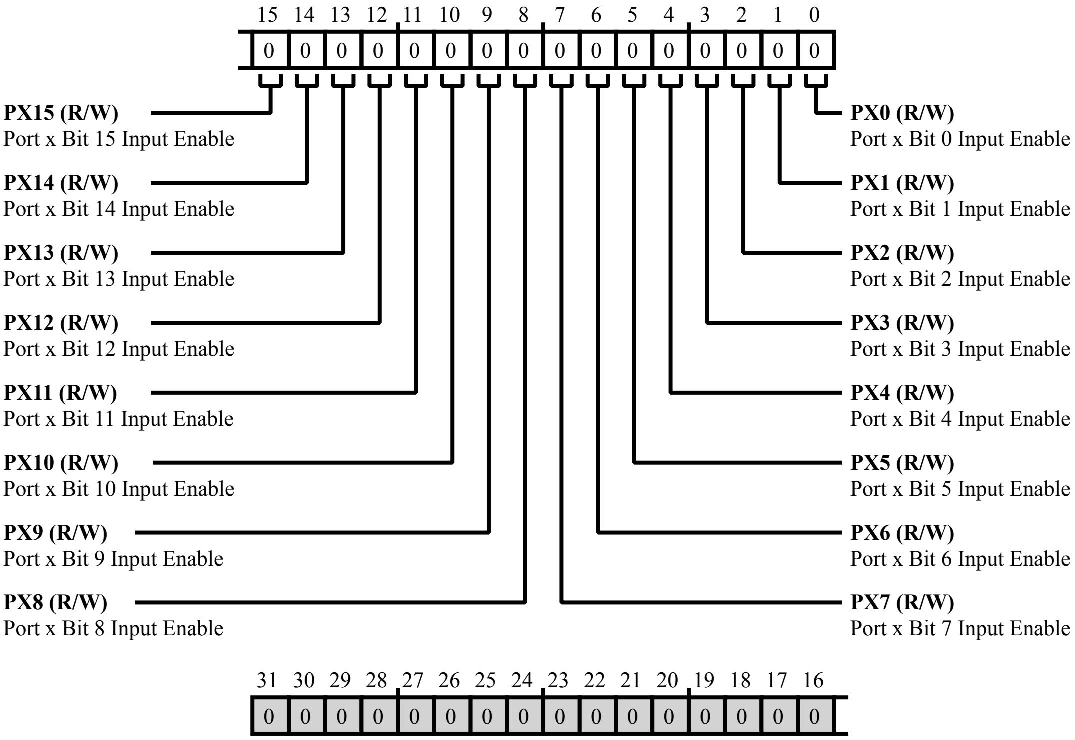

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Input Enable
PX14 (R/W)
Port x Bit 14 Input Enable
PX13 (R/W)
Port x Bit 13 Input Enable
PX12 (R/W)
Port x Bit 12 Input Enable
PX11 (R/W)
Port x Bit 11 Input Enable
PX10 (R/W)
Port x Bit 10 Input Enable
PX9 (R/W)
Port x Bit 9 Input Enable
PX8 (R/W)
Port x Bit 8 Input Enable
PX0 (R/W)
Port x Bit 0 Input Enable
PX1 (R/W)
Port x Bit 1 Input Enable
PX2 (R/W)
Port x Bit 2 Input Enable
PX3 (R/W)
Port x Bit 3 Input Enable
PX4 (R/W)
Port x Bit 4 Input Enable
PX5 (R/W)
Port x Bit 5 Input Enable
PX6 (R/W)
Port x Bit 6 Input Enable
PX7 (R/W)
Port x Bit 7 Input Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-17: PORT_INEN Register Diagram

Table 14-18: PORT_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 14-18: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
13 (R/W)	PX13	Port x Bit 13 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
12 (R/W)	PX12	Port x Bit 12 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
11 (R/W)	PX11	Port x Bit 11 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
10 (R/W)	PX10	Port x Bit 10 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
9 (R/W)	PX9	Port x Bit 9 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
8 (R/W)	PX8	Port x Bit 8 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
7 (R/W)	PX7	Port x Bit 7 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
6 (R/W)	PX6	Port x Bit 6 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
5 (R/W)	PX5	Port x Bit 5 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 14-18: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PX4	Port x Bit 4 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
3 (R/W)	PX3	Port x Bit 3 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
2 (R/W)	PX2	Port x Bit 2 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
1 (R/W)	PX1	Port x Bit 1 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
0 (R/W)	PX0	Port x Bit 0 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Port x GPIO Input Enable Clear Register

The PORT_INEN_CLR register disables the input drivers for GPIO pins. For more information, see the PORT_INEN register description.

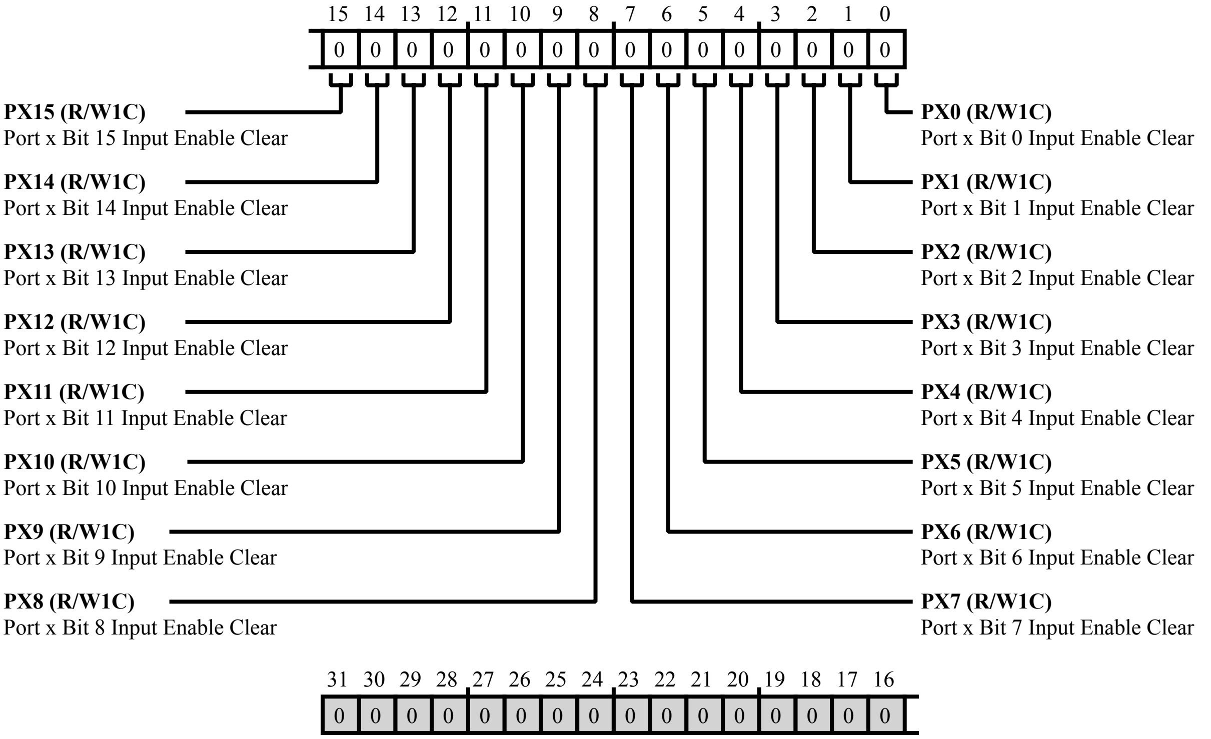

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1C)
Port x Bit 15 Input Enable Clear
PX14 (R/W1C)
Port x Bit 14 Input Enable Clear
PX13 (R/W1C)
Port x Bit 13 Input Enable Clear
PX12 (R/W1C)
Port x Bit 12 Input Enable Clear
PX11 (R/W1C)
Port x Bit 11 Input Enable Clear
PX10 (R/W1C)
Port x Bit 10 Input Enable Clear
PX9 (R/W1C)
Port x Bit 9 Input Enable Clear
PX8 (R/W1C)
Port x Bit 8 Input Enable Clear
PX0 (R/W1C)
Port x Bit 0 Input Enable Clear
PX1 (R/W1C)
Port x Bit 1 Input Enable Clear
PX2 (R/W1C)
Port x Bit 2 Input Enable Clear
PX3 (R/W1C)
Port x Bit 3 Input Enable Clear
PX4 (R/W1C)
Port x Bit 4 Input Enable Clear
PX5 (R/W1C)
Port x Bit 5 Input Enable Clear
PX6 (R/W1C)
Port x Bit 6 Input Enable Clear
PX7 (R/W1C)
Port x Bit 7 Input Enable Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-18: PORT_INEN_CLR Register Diagram

Table 14-19: PORT_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
14 (R/W1C)	PX14	Port x Bit 14 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
13 (R/W1C)	PX13	Port x Bit 13 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Table 14-19: PORT_INEN_CLR Register Fields (Continued)

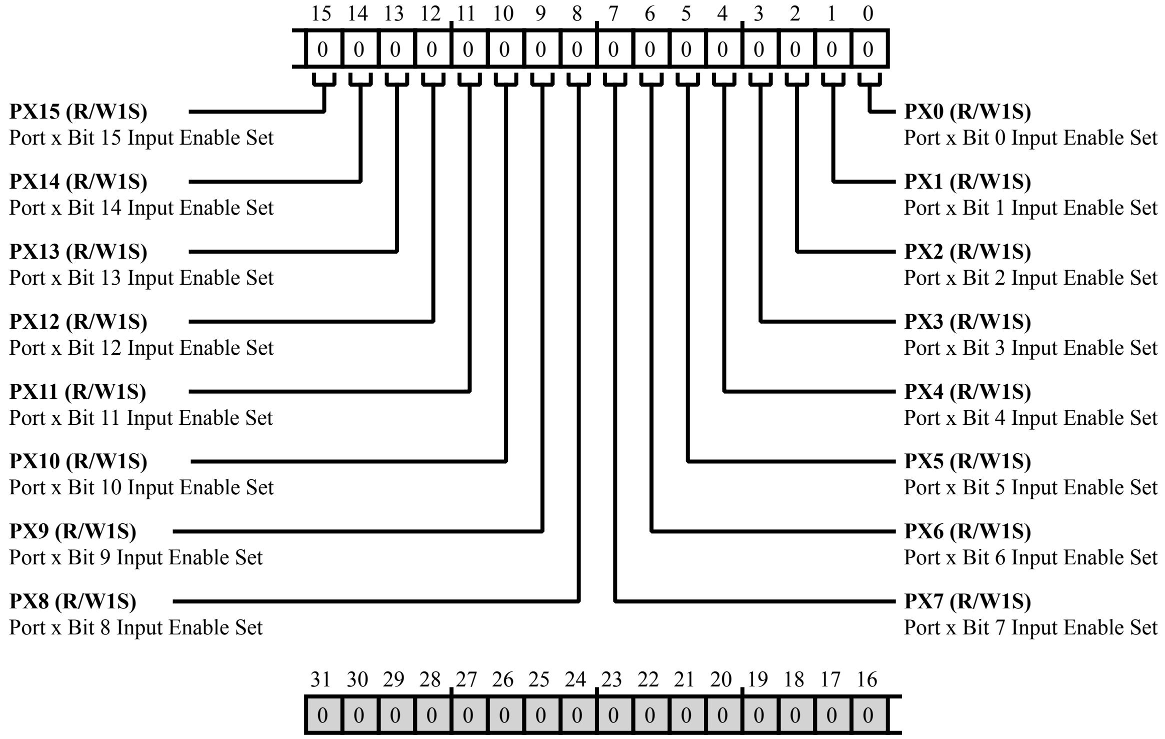
Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
11 (R/W1C)	PX11	Port x Bit 11 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
10 (R/W1C)	PX10	Port x Bit 10 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
9 (R/W1C)	PX9	Port x Bit 9 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
8 (R/W1C)	PX8	Port x Bit 8 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
7 (R/W1C)	PX7	Port x Bit 7 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
6 (R/W1C)	PX6	Port x Bit 6 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
5 (R/W1C)	PX5	Port x Bit 5 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
4 (R/W1C)	PX4	Port x Bit 4 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
3 (R/W1C)	PX3	Port x Bit 3 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Table 14-19: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
1 (R/W1C)	PX1	Port x Bit 1 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
0 (R/W1C)	PX0	Port x Bit 0 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Port x GPIO Input Enable Set Register

The `PORT_INEN_SET` register enables input drivers for GPIO pins. For more information, see the `PORT_INEN` register description.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1S)
Port x Bit 15 Input Enable Set
PX14 (R/W1S)
Port x Bit 14 Input Enable Set
PX13 (R/W1S)
Port x Bit 13 Input Enable Set
PX12 (R/W1S)
Port x Bit 12 Input Enable Set
PX11 (R/W1S)
Port x Bit 11 Input Enable Set
PX10 (R/W1S)
Port x Bit 10 Input Enable Set
PX9 (R/W1S)
Port x Bit 9 Input Enable Set
PX8 (R/W1S)
Port x Bit 8 Input Enable Set
PX0 (R/W1S)
Port x Bit 0 Input Enable Set
PX1 (R/W1S)
Port x Bit 1 Input Enable Set
PX2 (R/W1S)
Port x Bit 2 Input Enable Set
PX3 (R/W1S)
Port x Bit 3 Input Enable Set
PX4 (R/W1S)
Port x Bit 4 Input Enable Set
PX5 (R/W1S)
Port x Bit 5 Input Enable Set
PX6 (R/W1S)
Port x Bit 6 Input Enable Set
PX7 (R/W1S)
Port x Bit 7 Input Enable Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-19: PORT_INEN_SET Register Diagram

Table 14-20: PORT_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
14 (R/W1S)	PX14	Port x Bit 14 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
13 (R/W1S)	PX13	Port x Bit 13 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Table 14-20: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
11 (R/W1S)	PX11	Port x Bit 11 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
10 (R/W1S)	PX10	Port x Bit 10 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
9 (R/W1S)	PX9	Port x Bit 9 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
8 (R/W1S)	PX8	Port x Bit 8 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
7 (R/W1S)	PX7	Port x Bit 7 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
6 (R/W1S)	PX6	Port x Bit 6 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
5 (R/W1S)	PX5	Port x Bit 5 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
4 (R/W1S)	PX4	Port x Bit 4 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
3 (R/W1S)	PX3	Port x Bit 3 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Table 14-20: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
1 (R/W1S)	PX1	Port x Bit 1 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
0 (R/W1S)	PX0	Port x Bit 0 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Port x GPIO Lock Register

The PORT_LOCK register enables (unlocks) or disables (locks) write access selectively for the PORT control registers.

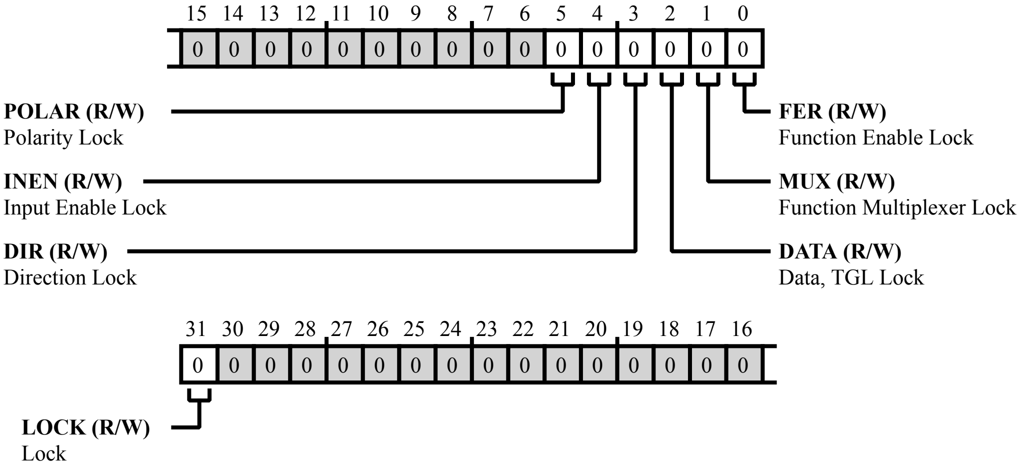

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
POLAR (R/W)
Polarity Lock
INEN (R/W)
Input Enable Lock
DIR (R/W)
Direction Lock
FER (R/W)
Function Enable Lock
MUX (R/W)
Function Multiplexer Lock
DATA (R/W)
Data, TGL Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock

Figure 14-20: PORT_LOCK Register Diagram

Table 14-21: PORT_LOCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the PORT_LOCK.LOCK bit is set, the PORT_LOCK register is read only (locked).	
		0	Unlock
		1	Lock
5 (R/W)	POLAR	Polarity Lock. The PORT_LOCK.POLAR disables write access to the PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers.	
		0	Unlock POL
		1	Lock POL
4 (R/W)	INEN	Input Enable Lock. The PORT_LOCK.INEN disables write access to the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers.	
		0	Unlock INEN
		1	Lock INEN

Table 14-21: PORT_LOCK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIR	Direction Lock. The `PORT_LOCK.DIR` disables write access to the `PORT_DIR`, `PORT_DIR_SET`, `PORT_DIR_CLR` registers.	
		0	Unlock DIR
		1	Lock DIR
2 (R/W)	DATA	Data, TGL Lock. The `PORT_LOCK.DATA` disables write access to the `PORT_DATA`, `PORT_DATA_SET`, `PORT_DATA_CLR`, and `PORT_DATA_TGL` registers.	
		0	Unlock DATA
		1	Lock DATA
1 (R/W)	MUX	Function Multiplexer Lock. The `PORT_LOCK.MUX` disables write accesses to the `PORT_MUX` register.	
		0	Unlock MUX
		1	Lock MUX
0 (R/W)	FER	Function Enable Lock. The `PORT_LOCK.FER` disables write access to the `PORT_FER`, `PORT_FER_SET`, and `PORT_FER_CLR` registers.	
		0	Unlock FER
		1	Lock FER

Port x Multiplexer Control Register

When a pin is in peripheral mode (not GPIO mode), the `PORT_MUX` register controls which peripheral takes ownership of a pin. Ports may have multiple, different peripheral functions. Two bits are required to describe every multiplexer on an individual pin-by-pin scheme. For example, bit 0 and bit 1 of the `PORT_MUX` register control the multiplexer of pin 0, bit 2 and bit 3 of `PORT_MUX` control the multiplexer of pin 1, and so on. The value of any `PORT_MUX` bit has no effect on the port pins when the associated bit in the `PORT_FER` register is 0 (selects GPIO mode). Even if a port has only one function, the `PORT_MUX` register is still present. For single function ports (no multiplexing is needed), leave the `PORT_MUX` bits at 0 (default). For all `PORT_MUX` bit fields: 00 = default/reset peripheral option, 01 = first alternate peripheral option, 10 = second alternate peripheral option, and 11 = third alternate peripheral option.

See the processor data sheet for details regarding the peripheral options associated with each port.

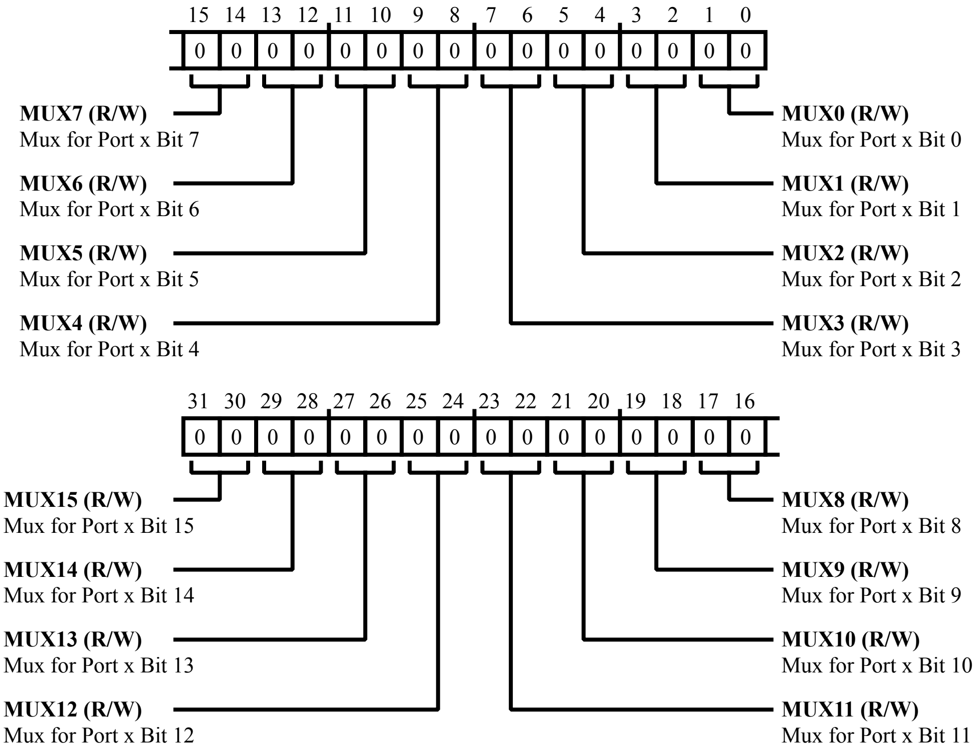

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUX7 (R/W)
Mux for Port x Bit 7
MUX6 (R/W)
Mux for Port x Bit 6
MUX5 (R/W)
Mux for Port x Bit 5
MUX4 (R/W)
Mux for Port x Bit 4
MUX0 (R/W)
Mux for Port x Bit 0
MUX1 (R/W)
Mux for Port x Bit 1
MUX2 (R/W)
Mux for Port x Bit 2
MUX3 (R/W)
Mux for Port x Bit 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUX15 (R/W)
Mux for Port x Bit 15
MUX14 (R/W)
Mux for Port x Bit 14
MUX13 (R/W)
Mux for Port x Bit 13
MUX12 (R/W)
Mux for Port x Bit 12
MUX8 (R/W)
Mux for Port x Bit 8
MUX9 (R/W)
Mux for Port x Bit 9
MUX10 (R/W)
Mux for Port x Bit 10
MUX11 (R/W)
Mux for Port x Bit 11

Figure 14-21: PORT_MUX Register Diagram

Table 14-22: PORT_MUX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MUX15	Mux for Port x Bit 15. The `PORT_MUX.MUX15` bit provides multiplexer control for port x bit 15.

Table 14-22: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:28 (R/W)	MUX14	Mux for Port x Bit 14. The `PORT_MUX.MUX14` bit provides multiplexer control for port x bit 14.
27:26 (R/W)	MUX13	Mux for Port x Bit 13. The `PORT_MUX.MUX13` bit provides multiplexer control for port x bit 13.
25:24 (R/W)	MUX12	Mux for Port x Bit 12. The `PORT_MUX.MUX12` bit provides multiplexer control for port x bit 12.
23:22 (R/W)	MUX11	Mux for Port x Bit 11. The `PORT_MUX.MUX11` bit provides multiplexer control for port x bit 11.
21:20 (R/W)	MUX10	Mux for Port x Bit 10. The `PORT_MUX.MUX10` bit provides multiplexer control for port x bit 10.
19:18 (R/W)	MUX9	Mux for Port x Bit 9. The `PORT_MUX.MUX9` bit provides multiplexer control for port x bit 9.
17:16 (R/W)	MUX8	Mux for Port x Bit 8. The `PORT_MUX.MUX8` bit provides multiplexer control for port x bit 8.
15:14 (R/W)	MUX7	Mux for Port x Bit 7. The `PORT_MUX.MUX7` bit provides multiplexer control for port x bit 7.
13:12 (R/W)	MUX6	Mux for Port x Bit 6. The `PORT_MUX.MUX6` bit provides multiplexer control for port x bit 6.
11:10 (R/W)	MUX5	Mux for Port x Bit 5. The `PORT_MUX.MUX5` bit provides multiplexer control for port x bit 5.
9:8 (R/W)	MUX4	Mux for Port x Bit 4. The `PORT_MUX.MUX4` bit provides multiplexer control for port x bit 4.
7:6 (R/W)	MUX3	Mux for Port x Bit 3. The `PORT_MUX.MUX3` bit provides multiplexer control for port x bit 3.
5:4 (R/W)	MUX2	Mux for Port x Bit 2. The `PORT_MUX.MUX2` bit provides multiplexer control for port x bit 2.
3:2 (R/W)	MUX1	Mux for Port x Bit 1. The `PORT_MUX.MUX1` bit provides multiplexer control for port x bit 1.
1:0 (R/W)	MUX0	Mux for Port x Bit 0. The `PORT_MUX.MUX0` bit provides multiplexer control for port x bit 0.

Port x GPIO Polarity Invert Register

The `PORT_POL`, `PORT_POL_SET`, and `PORT_POL_CLR` registers enable or disable inverting polarity of GPIO signals. To invert polarity of peripheral signals, use the inversion selection programming in the signal's corresponding module.

Writes to the `PORT_POL` register affect the polarity inversion selection of all pins of the port. To enable or disable polarity inversion for specific pins without impacting other pins of the port, use the `PORT_POL_SET` and `PORT_POL_CLR` registers.

Setting a bit in the `PORT_POL` register enables polarity inversion on the corresponding inversion GPIO pin, making the pin active-low or falling-edge sensitive. Clearing a bit in the `PORT_POL` register disables polarity (default state) on the corresponding GPIO pin, making it active-high or rising-edge sensitive.

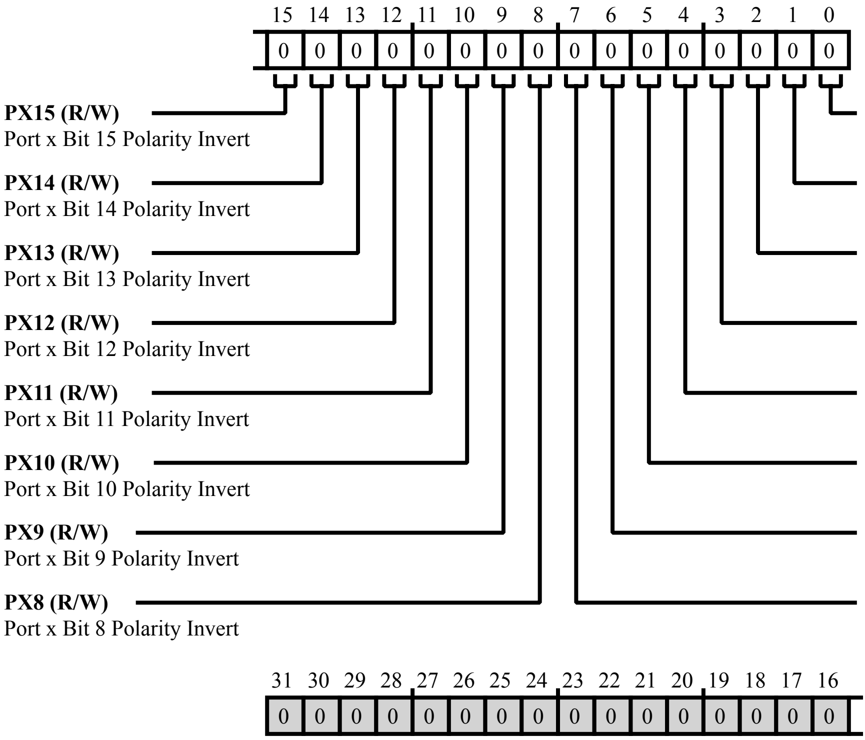

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W)
Port x Bit 15 Polarity Invert
PX14 (R/W)
Port x Bit 14 Polarity Invert
PX13 (R/W)
Port x Bit 13 Polarity Invert
PX12 (R/W)
Port x Bit 12 Polarity Invert
PX11 (R/W)
Port x Bit 11 Polarity Invert
PX10 (R/W)
Port x Bit 10 Polarity Invert
PX9 (R/W)
Port x Bit 9 Polarity Invert
PX8 (R/W)
Port x Bit 8 Polarity Invert
PX0 (R/W)
Port x Bit 0 Polarity Invert
PX1 (R/W)
Port x Bit 1 Polarity Invert
PX2 (R/W)
Port x Bit 2 Polarity Invert
PX3 (R/W)
Port x Bit 3 Polarity Invert
PX4 (R/W)
Port x Bit 4 Polarity Invert
PX5 (R/W)
Port x Bit 5 Polarity Invert
PX6 (R/W)
Port x Bit 6 Polarity Invert
PX7 (R/W)
Port x Bit 7 Polarity Invert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-22: PORT_POL Register Diagram

Table 14-23: PORT_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Polarity Invert. The `PORT_POL.PX15` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Polarity Invert. The `PORT_POL.PX14` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
13 (R/W)	PX13	Port x Bit 13 Polarity Invert. The `PORT_POL.PX13` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
12 (R/W)	PX12	Port x Bit 12 Polarity Invert. The `PORT_POL.PX12` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
11 (R/W)	PX11	Port x Bit 11 Polarity Invert. The `PORT_POL.PX11` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
10 (R/W)	PX10	Port x Bit 10 Polarity Invert. The `PORT_POL.PX10` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
9 (R/W)	PX9	Port x Bit 9 Polarity Invert. The `PORT_POL.PX9` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
8 (R/W)	PX8	Port x Bit 8 Polarity Invert. The `PORT_POL.PX8` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PX7	Port x Bit 7 Polarity Invert. The `PORT_POL.PX7` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
6 (R/W)	PX6	Port x Bit 6 Polarity Invert. The `PORT_POL.PX6` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
5 (R/W)	PX5	Port x Bit 5 Polarity Invert. The `PORT_POL.PX5` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
4 (R/W)	PX4	Port x Bit 4 Polarity Invert. The `PORT_POL.PX4` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
3 (R/W)	PX3	Port x Bit 3 Polarity Invert. The `PORT_POL.PX3` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
2 (R/W)	PX2	Port x Bit 2 Polarity Invert. The `PORT_POL.PX2` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
1 (R/W)	PX1	Port x Bit 1 Polarity Invert. The `PORT_POL.PX1` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PX0	Port x Bit 0 Polarity Invert. The `PORT_POL.PX0` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Port x GPIO Polarity Invert Clear Register

The PORT_POL_CLR register disables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

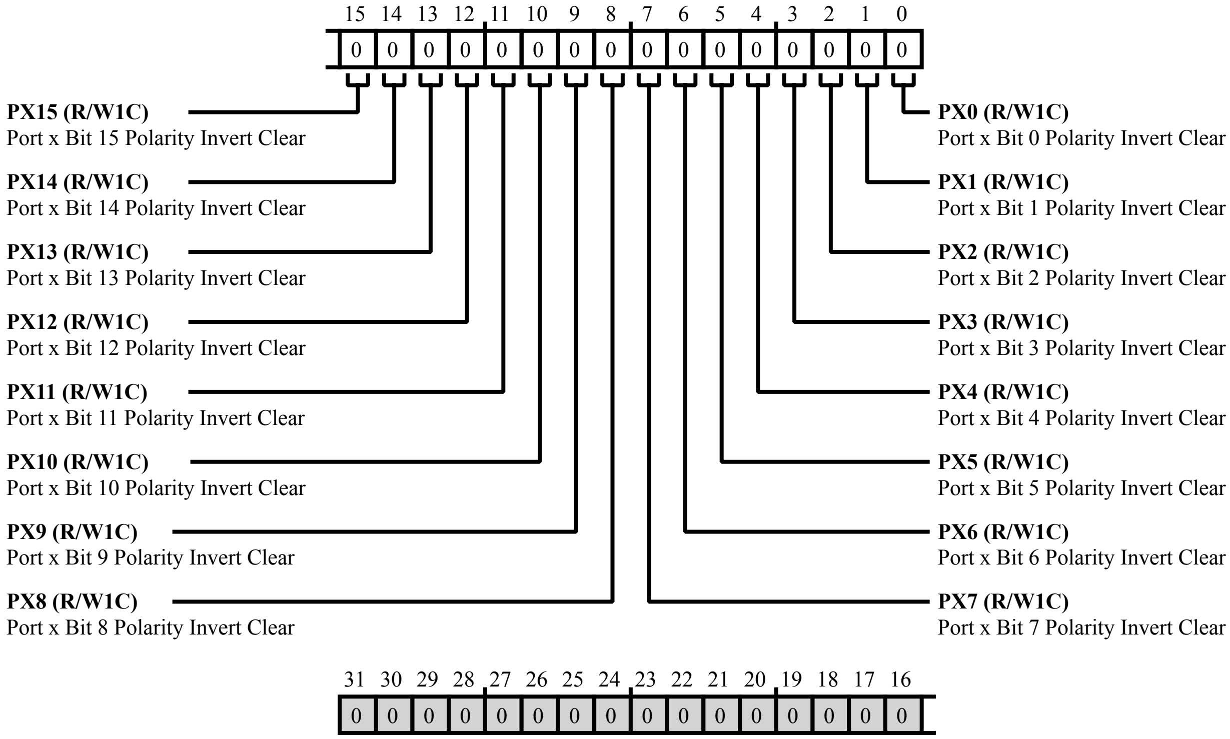

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1C)
Port x Bit 15 Polarity Invert Clear
PX14 (R/W1C)
Port x Bit 14 Polarity Invert Clear
PX13 (R/W1C)
Port x Bit 13 Polarity Invert Clear
PX12 (R/W1C)
Port x Bit 12 Polarity Invert Clear
PX11 (R/W1C)
Port x Bit 11 Polarity Invert Clear
PX10 (R/W1C)
Port x Bit 10 Polarity Invert Clear
PX9 (R/W1C)
Port x Bit 9 Polarity Invert Clear
PX8 (R/W1C)
Port x Bit 8 Polarity Invert Clear
PX0 (R/W1C)
Port x Bit 0 Polarity Invert Clear
PX1 (R/W1C)
Port x Bit 1 Polarity Invert Clear
PX2 (R/W1C)
Port x Bit 2 Polarity Invert Clear
PX3 (R/W1C)
Port x Bit 3 Polarity Invert Clear
PX4 (R/W1C)
Port x Bit 4 Polarity Invert Clear
PX5 (R/W1C)
Port x Bit 5 Polarity Invert Clear
PX6 (R/W1C)
Port x Bit 6 Polarity Invert Clear
PX7 (R/W1C)
Port x Bit 7 Polarity Invert Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-23: PORT_POL_CLR Register Diagram

Table 14-24: PORT_POL_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
14 (R/W1C)	PX14	Port x Bit 14 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
13 (R/W1C)	PX13	Port x Bit 13 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Table 14-24: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
11 (R/W1C)	PX11	Port x Bit 11 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
10 (R/W1C)	PX10	Port x Bit 10 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
9 (R/W1C)	PX9	Port x Bit 9 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
8 (R/W1C)	PX8	Port x Bit 8 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
7 (R/W1C)	PX7	Port x Bit 7 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
6 (R/W1C)	PX6	Port x Bit 6 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
5 (R/W1C)	PX5	Port x Bit 5 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
4 (R/W1C)	PX4	Port x Bit 4 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
3 (R/W1C)	PX3	Port x Bit 3 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Table 14-24: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
1 (R/W1C)	PX1	Port x Bit 1 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
0 (R/W1C)	PX0	Port x Bit 0 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Port x GPIO Polarity Invert Set Register

The `PORT_POL_SET` register enables polarity inversion for GPIO pins. For more information, see the `PORT_POL` register description.

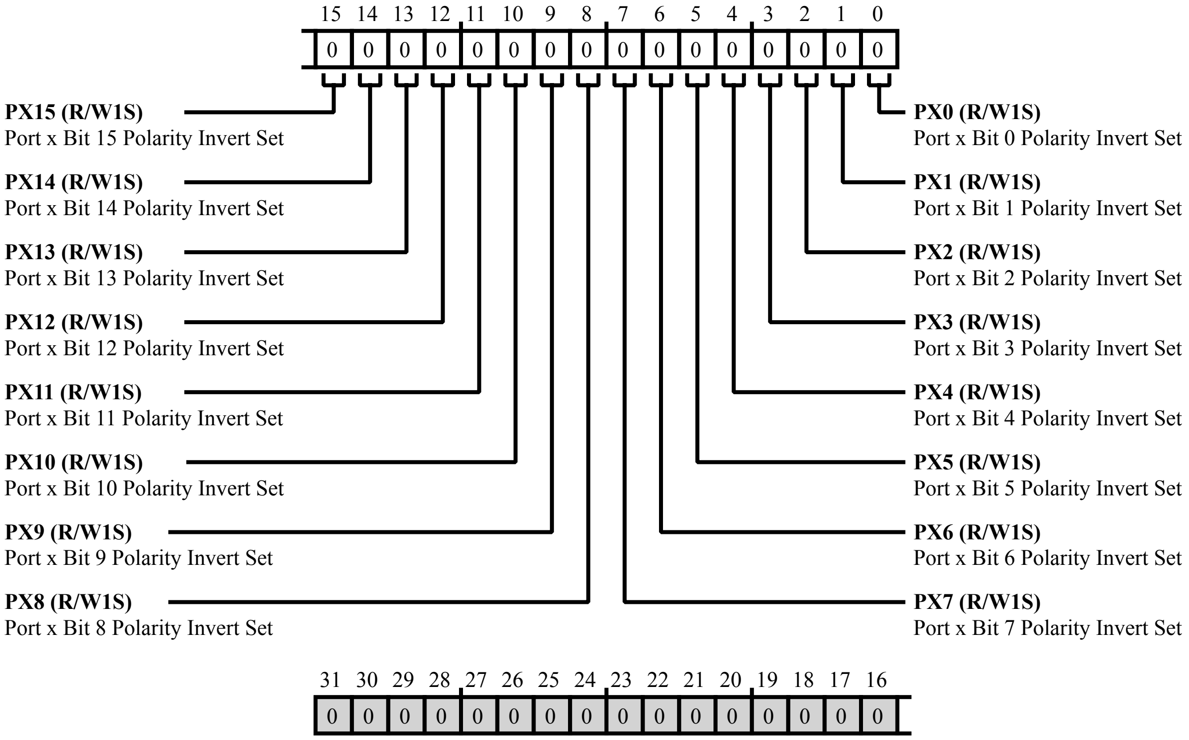

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PX15 (R/W1S)
Port x Bit 15 Polarity Invert Set
PX14 (R/W1S)
Port x Bit 14 Polarity Invert Set
PX13 (R/W1S)
Port x Bit 13 Polarity Invert Set
PX12 (R/W1S)
Port x Bit 12 Polarity Invert Set
PX11 (R/W1S)
Port x Bit 11 Polarity Invert Set
PX10 (R/W1S)
Port x Bit 10 Polarity Invert Set
PX9 (R/W1S)
Port x Bit 9 Polarity Invert Set
PX8 (R/W1S)
Port x Bit 8 Polarity Invert Set
PX0 (R/W1S)
Port x Bit 0 Polarity Invert Set
PX1 (R/W1S)
Port x Bit 1 Polarity Invert Set
PX2 (R/W1S)
Port x Bit 2 Polarity Invert Set
PX3 (R/W1S)
Port x Bit 3 Polarity Invert Set
PX4 (R/W1S)
Port x Bit 4 Polarity Invert Set
PX5 (R/W1S)
Port x Bit 5 Polarity Invert Set
PX6 (R/W1S)
Port x Bit 6 Polarity Invert Set
PX7 (R/W1S)
Port x Bit 7 Polarity Invert Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 14-24: PORT_POL_SET Register Diagram

Table 14-25: PORT_POL_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Polarity Invert Set. The `PORT_POL_SET.PX15` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
14 (R/W1S)	PX14	Port x Bit 14 Polarity Invert Set. The `PORT_POL_SET.PX14` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

Table 14-25: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Polarity Invert Set. The `PORT_POL_SET.PX13` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
12 (R/W1S)	PX12	Port x Bit 12 Polarity Invert Set. The `PORT_POL_SET.PX12` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
11 (R/W1S)	PX11	Port x Bit 11 Polarity Invert Set. The `PORT_POL_SET.PX11` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
10 (R/W1S)	PX10	Port x Bit 10 Polarity Invert Set. The `PORT_POL_SET.PX10` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
9 (R/W1S)	PX9	Port x Bit 9 Polarity Invert Set. The `PORT_POL_SET.PX9` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
8 (R/W1S)	PX8	Port x Bit 8 Polarity Invert Set. The `PORT_POL_SET.PX8` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
7 (R/W1S)	PX7	Port x Bit 7 Polarity Invert Set. The `PORT_POL_SET.PX7` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

Table 14-25: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Polarity Invert Set. The `PORT_POL_SET.PX6` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
5 (R/W1S)	PX5	Port x Bit 5 Polarity Invert Set. The `PORT_POL_SET.PX5` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
4 (R/W1S)	PX4	Port x Bit 4 Polarity Invert Set. The `PORT_POL_SET.PX4` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
3 (R/W1S)	PX3	Port x Bit 3 Polarity Invert Set. The `PORT_POL_SET.PX3` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
2 (R/W1S)	PX2	Port x Bit 2 Polarity Invert Set. The `PORT_POL_SET.PX2` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
1 (R/W1S)	PX1	Port x Bit 1 Polarity Invert Set. The `PORT_POL_SET.PX1` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
0 (R/W1S)	PX0	Port x Bit 0 Polarity Invert Set. The `PORT_POL_SET.PX0` bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

ADSP-SC58x PINT Register Descriptions

The Pin Interrupt module (PINT) contains the following registers.

Table 14-26: ADSP-SC58x PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

PINT Assign Register

The PINT_ASSIGN register controls the pin-to-interrupt request assignment in a byte-wide manner. This register consists of four control bytes that each function as a multiplexer control.

The PINT ports are subdivided into 8-bit half ports, resulting in lower and upper half 8-bit units. Using the multiplexers controlled by the PINT_ASSIGN register, the lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINT block. The upper half units can be forwarded to either byte 1 or byte 3 of the PINT block, without further restrictions.

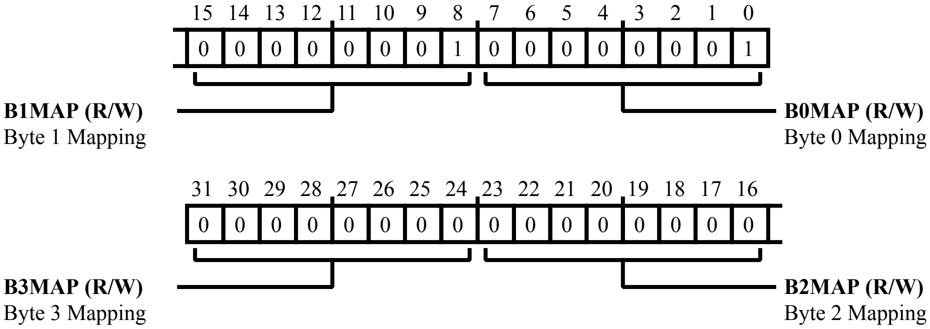

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
B1MAP (R/W)
Byte 1 Mapping
B0MAP (R/W)
Byte 0 Mapping
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B3MAP (R/W)
Byte 3 Mapping
B2MAP (R/W)
Byte 2 Mapping

Figure 14-25: PINT_ASSIGN Register Diagram

Table 14-27: PINT_ASSIGN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	B3MAP	Byte 3 Mapping.	
		0	B3MAP_PAH. Byte 3 = PA.H
		1	B3MAP_PBH. Byte 3 = PB.H
23:16 (R/W)	B2MAP	Byte 2 Mapping.	
		0	B2MAP_PAL. Byte 2 = PA.L
		1	B2MAP_PBL. Byte 2 = PB.L
15:8 (R/W)	B1MAP	Byte 1 Mapping.	
		0	B1MAP_PAH. Byte 1 = PA.H
		1	B1MAP_PBH. Byte 1 = PB.H
7:0 (R/W)	B0MAP	Byte 0 Mapping.	
		0	B0MAP_PAL. Byte 0 = PA.L
		1	B0MAP_PBL. Byte 0 = PB.L

PINT Edge Clear Register

The `PINT_EDGE_CLR` register permits selecting level-sensitive interrupts. Writing 1 to a bit in `PINT_EDGE_CLR` enables level sensitivity for the corresponding pin interrupt.

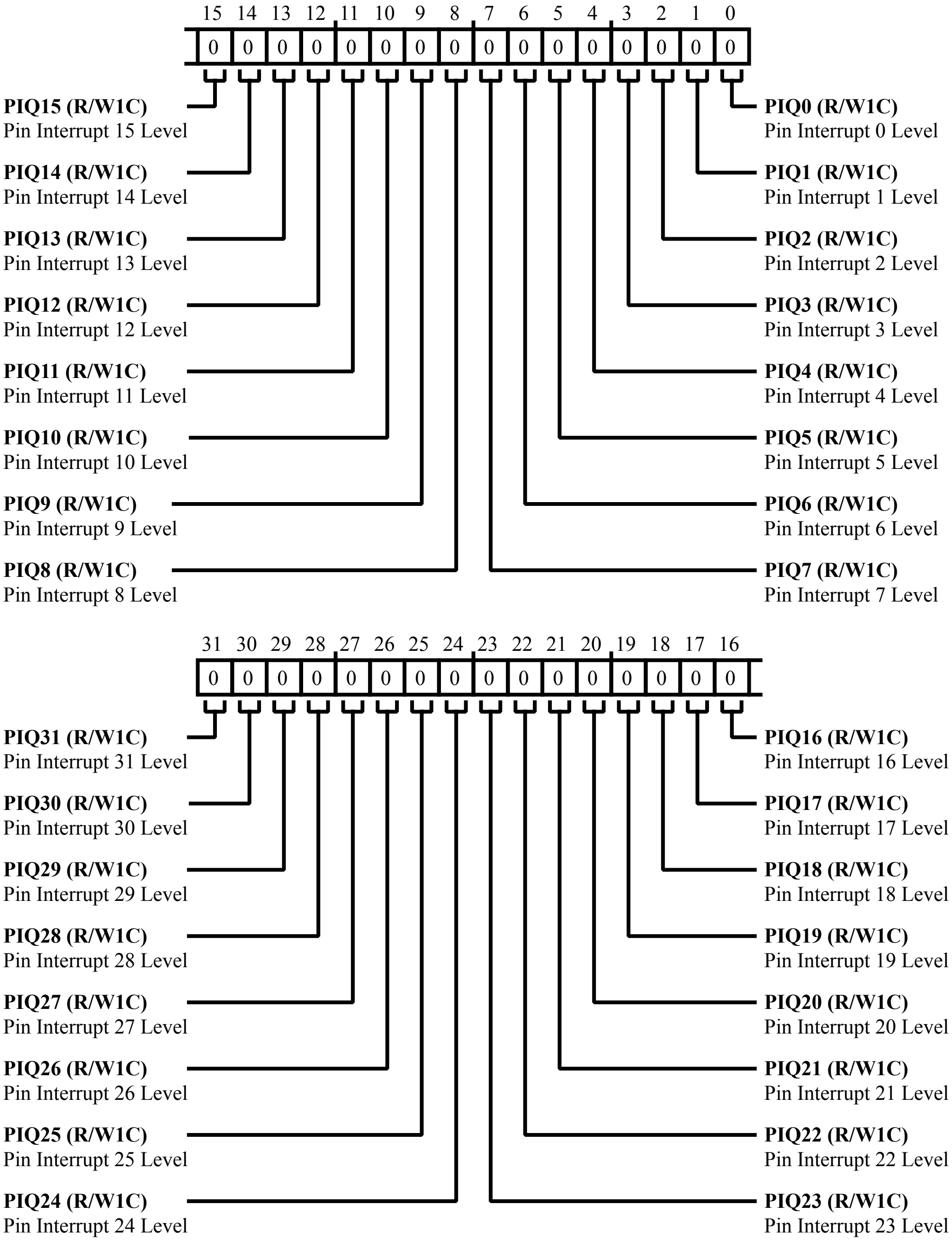

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1C)
Pin Interrupt 15 Level
PIQ14 (R/W1C)
Pin Interrupt 14 Level
PIQ13 (R/W1C)
Pin Interrupt 13 Level
PIQ12 (R/W1C)
Pin Interrupt 12 Level
PIQ11 (R/W1C)
Pin Interrupt 11 Level
PIQ10 (R/W1C)
Pin Interrupt 10 Level
PIQ9 (R/W1C)
Pin Interrupt 9 Level
PIQ8 (R/W1C)
Pin Interrupt 8 Level
PIQ0 (R/W1C)
Pin Interrupt 0 Level
PIQ1 (R/W1C)
Pin Interrupt 1 Level
PIQ2 (R/W1C)
Pin Interrupt 2 Level
PIQ3 (R/W1C)
Pin Interrupt 3 Level
PIQ4 (R/W1C)
Pin Interrupt 4 Level
PIQ5 (R/W1C)
Pin Interrupt 5 Level
PIQ6 (R/W1C)
Pin Interrupt 6 Level
PIQ7 (R/W1C)
Pin Interrupt 7 Level
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1C)
Pin Interrupt 31 Level
PIQ30 (R/W1C)
Pin Interrupt 30 Level
PIQ29 (R/W1C)
Pin Interrupt 29 Level
PIQ28 (R/W1C)
Pin Interrupt 28 Level
PIQ27 (R/W1C)
Pin Interrupt 27 Level
PIQ26 (R/W1C)
Pin Interrupt 26 Level
PIQ25 (R/W1C)
Pin Interrupt 25 Level
PIQ24 (R/W1C)
Pin Interrupt 24 Level
PIQ16 (R/W1C)
Pin Interrupt 16 Level
PIQ17 (R/W1C)
Pin Interrupt 17 Level
PIQ18 (R/W1C)
Pin Interrupt 18 Level
PIQ19 (R/W1C)
Pin Interrupt 19 Level
PIQ20 (R/W1C)
Pin Interrupt 20 Level
PIQ21 (R/W1C)
Pin Interrupt 21 Level
PIQ22 (R/W1C)
Pin Interrupt 22 Level
PIQ23 (R/W1C)
Pin Interrupt 23 Level

Figure 14-26: PINT_EDGE_CLR Register Diagram

Table 14-28: PINT_EDGE_CLR Register Fields

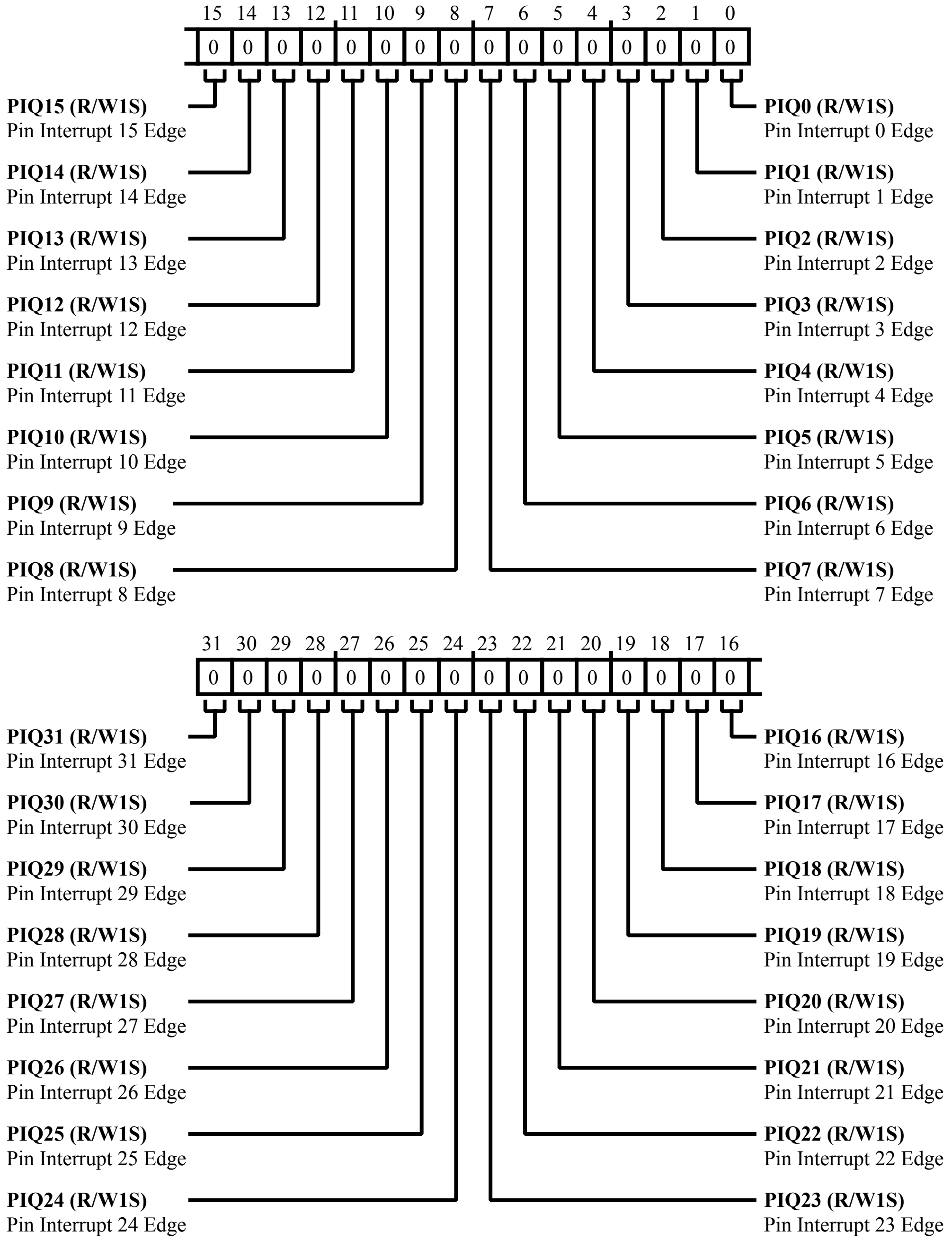
Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Level. Set the `PINT_EDGE_CLR.PIQ31` bit to enable level sensitivity.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Level. Set the `PINT_EDGE_CLR.PIQ30` bit to enable level sensitivity.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Level. Set the `PINT_EDGE_CLR.PIQ29` bit to enable level sensitivity.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Level. Set the `PINT_EDGE_CLR.PIQ28` bit to enable level sensitivity.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Level. Set the `PINT_EDGE_CLR.PIQ27` bit to enable level sensitivity.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Level. Set the `PINT_EDGE_CLR.PIQ26` bit to enable level sensitivity.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Level. Set the `PINT_EDGE_CLR.PIQ25` bit to enable level sensitivity.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Level. Set the `PINT_EDGE_CLR.PIQ24` bit to enable level sensitivity.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Level. Set the `PINT_EDGE_CLR.PIQ23` bit to enable level sensitivity.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Level. Set the `PINT_EDGE_CLR.PIQ22` bit to enable level sensitivity.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Level. Set the `PINT_EDGE_CLR.PIQ21` bit to enable level sensitivity.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Level. Set the `PINT_EDGE_CLR.PIQ20` bit to enable level sensitivity.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Level. Set the `PINT_EDGE_CLR.PIQ19` bit to enable level sensitivity.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Level. Set the `PINT_EDGE_CLR.PIQ18` bit to enable level sensitivity.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Level. Set the `PINT_EDGE_CLR.PIQ17` bit to enable level sensitivity.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Level. Set the `PINT_EDGE_CLR.PIQ16` bit to enable level sensitivity.

Table 14-28: PINT_EDGE_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Level. Set the `PINT_EDGE_CLR.PIQ15` bit to enable level sensitivity.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Level. Set the `PINT_EDGE_CLR.PIQ14` bit to enable level sensitivity.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Level. Set the `PINT_EDGE_CLR.PIQ13` bit to enable level sensitivity.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Level. Set the `PINT_EDGE_CLR.PIQ12` bit to enable level sensitivity.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Level. Set the `PINT_EDGE_CLR.PIQ11` bit to enable level sensitivity.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Level. Set the `PINT_EDGE_CLR.PIQ10` bit to enable level sensitivity.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Level. Set the `PINT_EDGE_CLR.PIQ9` bit to enable level sensitivity.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Level. Set the `PINT_EDGE_CLR.PIQ8` bit to enable level sensitivity.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Level. Set the `PINT_EDGE_CLR.PIQ7` bit to enable level sensitivity.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Level. Set the `PINT_EDGE_CLR.PIQ6` bit to enable level sensitivity.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Level. Set the `PINT_EDGE_CLR.PIQ5` bit to enable level sensitivity.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Level. Set the `PINT_EDGE_CLR.PIQ4` bit to enable level sensitivity.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Level. Set the `PINT_EDGE_CLR.PIQ3` bit to enable level sensitivity.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Level. Set the `PINT_EDGE_CLR.PIQ2` bit to enable level sensitivity.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Level. Set the `PINT_EDGE_CLR.PIQ1` bit to enable level sensitivity.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Level. Set the `PINT_EDGE_CLR.PIQ0` bit to enable level sensitivity.

PINT Edge Set Register

The `PINT_EDGE_SET` register permits selecting edge-sensitive interrupts. Writing 1 to a bit in `PINT_EDGE_SET` enables edge sensitivity for the corresponding pin interrupt.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1S)
Pin Interrupt 15 Edge
PIQ14 (R/W1S)
Pin Interrupt 14 Edge
PIQ13 (R/W1S)
Pin Interrupt 13 Edge
PIQ12 (R/W1S)
Pin Interrupt 12 Edge
PIQ11 (R/W1S)
Pin Interrupt 11 Edge
PIQ10 (R/W1S)
Pin Interrupt 10 Edge
PIQ9 (R/W1S)
Pin Interrupt 9 Edge
PIQ8 (R/W1S)
Pin Interrupt 8 Edge
PIQ0 (R/W1S)
Pin Interrupt 0 Edge
PIQ1 (R/W1S)
Pin Interrupt 1 Edge
PIQ2 (R/W1S)
Pin Interrupt 2 Edge
PIQ3 (R/W1S)
Pin Interrupt 3 Edge
PIQ4 (R/W1S)
Pin Interrupt 4 Edge
PIQ5 (R/W1S)
Pin Interrupt 5 Edge
PIQ6 (R/W1S)
Pin Interrupt 6 Edge
PIQ7 (R/W1S)
Pin Interrupt 7 Edge
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1S)
Pin Interrupt 31 Edge
PIQ30 (R/W1S)
Pin Interrupt 30 Edge
PIQ29 (R/W1S)
Pin Interrupt 29 Edge
PIQ28 (R/W1S)
Pin Interrupt 28 Edge
PIQ27 (R/W1S)
Pin Interrupt 27 Edge
PIQ26 (R/W1S)
Pin Interrupt 26 Edge
PIQ25 (R/W1S)
Pin Interrupt 25 Edge
PIQ24 (R/W1S)
Pin Interrupt 24 Edge
PIQ16 (R/W1S)
Pin Interrupt 16 Edge
PIQ17 (R/W1S)
Pin Interrupt 17 Edge
PIQ18 (R/W1S)
Pin Interrupt 18 Edge
PIQ19 (R/W1S)
Pin Interrupt 19 Edge
PIQ20 (R/W1S)
Pin Interrupt 20 Edge
PIQ21 (R/W1S)
Pin Interrupt 21 Edge
PIQ22 (R/W1S)
Pin Interrupt 22 Edge
PIQ23 (R/W1S)
Pin Interrupt 23 Edge

Figure 14-27: PINT_EDGE_SET Register Diagram

Table 14-29: PINT_EDGE_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Edge. Set the `PINT_EDGE_SET.PIQ31` bit to enable edge sensitivity.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Edge. Set the `PINT_EDGE_SET.PIQ30` bit to enable edge sensitivity.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Edge. Set the `PINT_EDGE_SET.PIQ29` bit to enable edge sensitivity.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Edge. Set the `PINT_EDGE_SET.PIQ28` bit to enable edge sensitivity.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Edge. Set the `PINT_EDGE_SET.PIQ27` bit to enable edge sensitivity.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Edge. Set the `PINT_EDGE_SET.PIQ26` bit to enable edge sensitivity.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Edge. Set the `PINT_EDGE_SET.PIQ25` bit to enable edge sensitivity.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Edge. Set the `PINT_EDGE_SET.PIQ24` bit to enable edge sensitivity.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Edge. Set the `PINT_EDGE_SET.PIQ23` bit to enable edge sensitivity.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Edge. Set the `PINT_EDGE_SET.PIQ22` bit to enable edge sensitivity.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Edge. Set the `PINT_EDGE_SET.PIQ21` bit to enable edge sensitivity.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Edge. Set the `PINT_EDGE_SET.PIQ20` bit to enable edge sensitivity.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Edge. Set the `PINT_EDGE_SET.PIQ19` bit to enable edge sensitivity.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Edge. Set the `PINT_EDGE_SET.PIQ18` bit to enable edge sensitivity.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Edge. Set the `PINT_EDGE_SET.PIQ17` bit to enable edge sensitivity.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Edge. Set the `PINT_EDGE_SET.PIQ16` bit to enable edge sensitivity.

Table 14-29: PINT_EDGE_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Edge. Set the `PINT_EDGE_SET.PIQ15` bit to enable edge sensitivity.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Edge. Set the `PINT_EDGE_SET.PIQ14` bit to enable edge sensitivity.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Edge. Set the `PINT_EDGE_SET.PIQ13` bit to enable edge sensitivity.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Edge. Set the `PINT_EDGE_SET.PIQ12` bit to enable edge sensitivity.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Edge. Set the `PINT_EDGE_SET.PIQ11` bit to enable edge sensitivity.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Edge. Set the `PINT_EDGE_SET.PIQ10` bit to enable edge sensitivity.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Edge. Set the `PINT_EDGE_SET.PIQ9` bit to enable edge sensitivity.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Edge. Set the `PINT_EDGE_SET.PIQ8` bit to enable edge sensitivity.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Edge. Set the `PINT_EDGE_SET.PIQ7` bit to enable edge sensitivity.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Edge. Set the `PINT_EDGE_SET.PIQ6` bit to enable edge sensitivity.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Edge. Set the `PINT_EDGE_SET.PIQ5` bit to enable edge sensitivity.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Edge. Set the `PINT_EDGE_SET.PIQ4` bit to enable edge sensitivity.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Edge. Set the `PINT_EDGE_SET.PIQ3` bit to enable edge sensitivity.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Edge. Set the `PINT_EDGE_SET.PIQ2` bit to enable edge sensitivity.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Edge. Set the `PINT_EDGE_SET.PIQ1` bit to enable edge sensitivity.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Edge. Set the `PINT_EDGE_SET.PIQ0` bit to enable edge sensitivity.

PINT Invert Clear Register

The `PINT_INV_CLR` register disables inverting input polarity. Writing 1 to a bit in `PINT_INV_CLR` disables an inverter for input on the corresponding pin.

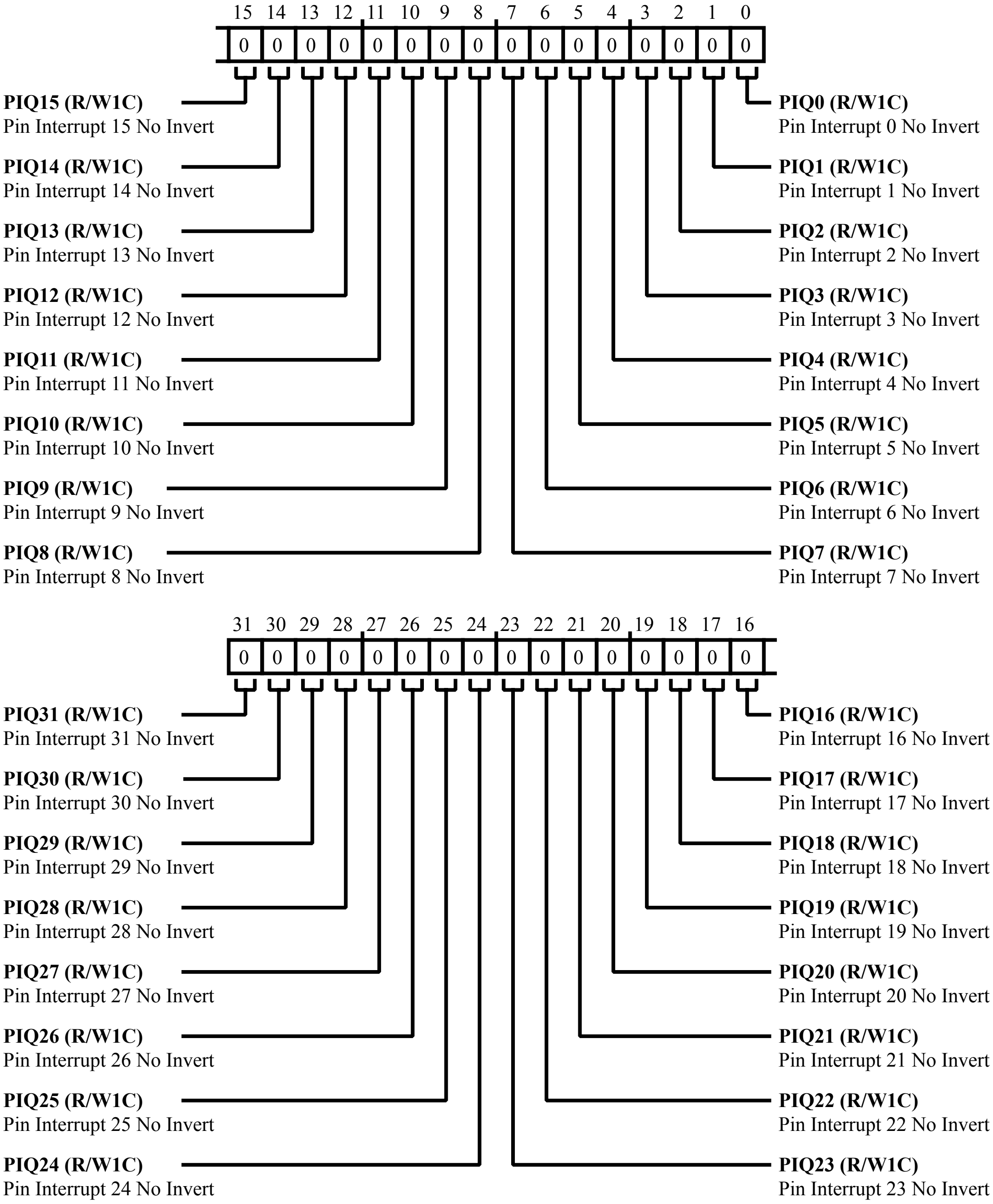

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1C)
Pin Interrupt 15 No Invert
PIQ14 (R/W1C)
Pin Interrupt 14 No Invert
PIQ13 (R/W1C)
Pin Interrupt 13 No Invert
PIQ12 (R/W1C)
Pin Interrupt 12 No Invert
PIQ11 (R/W1C)
Pin Interrupt 11 No Invert
PIQ10 (R/W1C)
Pin Interrupt 10 No Invert
PIQ9 (R/W1C)
Pin Interrupt 9 No Invert
PIQ8 (R/W1C)
Pin Interrupt 8 No Invert
PIQ0 (R/W1C)
Pin Interrupt 0 No Invert
PIQ1 (R/W1C)
Pin Interrupt 1 No Invert
PIQ2 (R/W1C)
Pin Interrupt 2 No Invert
PIQ3 (R/W1C)
Pin Interrupt 3 No Invert
PIQ4 (R/W1C)
Pin Interrupt 4 No Invert
PIQ5 (R/W1C)
Pin Interrupt 5 No Invert
PIQ6 (R/W1C)
Pin Interrupt 6 No Invert
PIQ7 (R/W1C)
Pin Interrupt 7 No Invert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1C)
Pin Interrupt 31 No Invert
PIQ30 (R/W1C)
Pin Interrupt 30 No Invert
PIQ29 (R/W1C)
Pin Interrupt 29 No Invert
PIQ28 (R/W1C)
Pin Interrupt 28 No Invert
PIQ27 (R/W1C)
Pin Interrupt 27 No Invert
PIQ26 (R/W1C)
Pin Interrupt 26 No Invert
PIQ25 (R/W1C)
Pin Interrupt 25 No Invert
PIQ24 (R/W1C)
Pin Interrupt 24 No Invert
PIQ16 (R/W1C)
Pin Interrupt 16 No Invert
PIQ17 (R/W1C)
Pin Interrupt 17 No Invert
PIQ18 (R/W1C)
Pin Interrupt 18 No Invert
PIQ19 (R/W1C)
Pin Interrupt 19 No Invert
PIQ20 (R/W1C)
Pin Interrupt 20 No Invert
PIQ21 (R/W1C)
Pin Interrupt 21 No Invert
PIQ22 (R/W1C)
Pin Interrupt 22 No Invert
PIQ23 (R/W1C)
Pin Interrupt 23 No Invert

Figure 14-28: PINT_INV_CLR Register Diagram

Table 14-30: PINT_INV_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 No Invert. Set the `PINT_INV_CLR.PIQ31` bit to disable inverted input.
30 (R/W1C)	PIQ30	Pin Interrupt 30 No Invert. Set the `PINT_INV_CLR.PIQ30` bit to disable inverted input.
29 (R/W1C)	PIQ29	Pin Interrupt 29 No Invert. Set the `PINT_INV_CLR.PIQ29` bit to disable inverted input.
28 (R/W1C)	PIQ28	Pin Interrupt 28 No Invert. Set the `PINT_INV_CLR.PIQ28` bit to disable inverted input.
27 (R/W1C)	PIQ27	Pin Interrupt 27 No Invert. Set the `PINT_INV_CLR.PIQ27` bit to disable inverted input.
26 (R/W1C)	PIQ26	Pin Interrupt 26 No Invert. Set the `PINT_INV_CLR.PIQ26` bit to disable inverted input.
25 (R/W1C)	PIQ25	Pin Interrupt 25 No Invert. Set the `PINT_INV_CLR.PIQ25` bit to disable inverted input.
24 (R/W1C)	PIQ24	Pin Interrupt 24 No Invert. Set the `PINT_INV_CLR.PIQ24` bit to disable inverted input.
23 (R/W1C)	PIQ23	Pin Interrupt 23 No Invert. Set the `PINT_INV_CLR.PIQ23` bit to disable inverted input.
22 (R/W1C)	PIQ22	Pin Interrupt 22 No Invert. Set the `PINT_INV_CLR.PIQ22` bit to disable inverted input.
21 (R/W1C)	PIQ21	Pin Interrupt 21 No Invert. Set the `PINT_INV_CLR.PIQ21` bit to disable inverted input.
20 (R/W1C)	PIQ20	Pin Interrupt 20 No Invert. Set the `PINT_INV_CLR.PIQ20` bit to disable inverted input.
19 (R/W1C)	PIQ19	Pin Interrupt 19 No Invert. Set the `PINT_INV_CLR.PIQ19` bit to disable inverted input.
18 (R/W1C)	PIQ18	Pin Interrupt 18 No Invert. Set the `PINT_INV_CLR.PIQ18` bit to disable inverted input.
17 (R/W1C)	PIQ17	Pin Interrupt 17 No Invert. Set the `PINT_INV_CLR.PIQ17` bit to disable inverted input.
16 (R/W1C)	PIQ16	Pin Interrupt 16 No Invert. Set the `PINT_INV_CLR.PIQ16` bit to disable inverted input.

Table 14-30: PINT_INV_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 No Invert. Set the `PINT_INV_CLR.PIQ15` bit to disable inverted input.
14 (R/W1C)	PIQ14	Pin Interrupt 14 No Invert. Set the `PINT_INV_CLR.PIQ14` bit to disable inverted input.
13 (R/W1C)	PIQ13	Pin Interrupt 13 No Invert. Set the `PINT_INV_CLR.PIQ13` bit to disable inverted input.
12 (R/W1C)	PIQ12	Pin Interrupt 12 No Invert. Set the `PINT_INV_CLR.PIQ12` bit to disable inverted input.
11 (R/W1C)	PIQ11	Pin Interrupt 11 No Invert. Set the `PINT_INV_CLR.PIQ11` bit to disable inverted input.
10 (R/W1C)	PIQ10	Pin Interrupt 10 No Invert. Set the `PINT_INV_CLR.PIQ10` bit to disable inverted input.
9 (R/W1C)	PIQ9	Pin Interrupt 9 No Invert. Set the `PINT_INV_CLR.PIQ9` bit to disable inverted input.
8 (R/W1C)	PIQ8	Pin Interrupt 8 No Invert. Set the `PINT_INV_CLR.PIQ8` bit to disable inverted input.
7 (R/W1C)	PIQ7	Pin Interrupt 7 No Invert. Set the `PINT_INV_CLR.PIQ7` bit to disable inverted input.
6 (R/W1C)	PIQ6	Pin Interrupt 6 No Invert. Set the `PINT_INV_CLR.PIQ6` bit to disable inverted input.
5 (R/W1C)	PIQ5	Pin Interrupt 5 No Invert. Set the `PINT_INV_CLR.PIQ5` bit to disable inverted input.
4 (R/W1C)	PIQ4	Pin Interrupt 4 No Invert. Set the `PINT_INV_CLR.PIQ4` bit to disable inverted input.
3 (R/W1C)	PIQ3	Pin Interrupt 3 No Invert. Set the `PINT_INV_CLR.PIQ3` bit to disable inverted input.
2 (R/W1C)	PIQ2	Pin Interrupt 2 No Invert. Set the `PINT_INV_CLR.PIQ2` bit to disable inverted input.
1 (R/W1C)	PIQ1	Pin Interrupt 1 No Invert. Set the `PINT_INV_CLR.PIQ1` bit to disable inverted input.
0 (R/W1C)	PIQ0	Pin Interrupt 0 No Invert. Set the `PINT_INV_CLR.PIQ0` bit to disable inverted input.

PINT Invert Set Register

The `PINT_INV_SET` register enables inverting input polarity. Writing 1 to a bit in `PINT_INV_SET` enables an inverter for input on the corresponding pin.

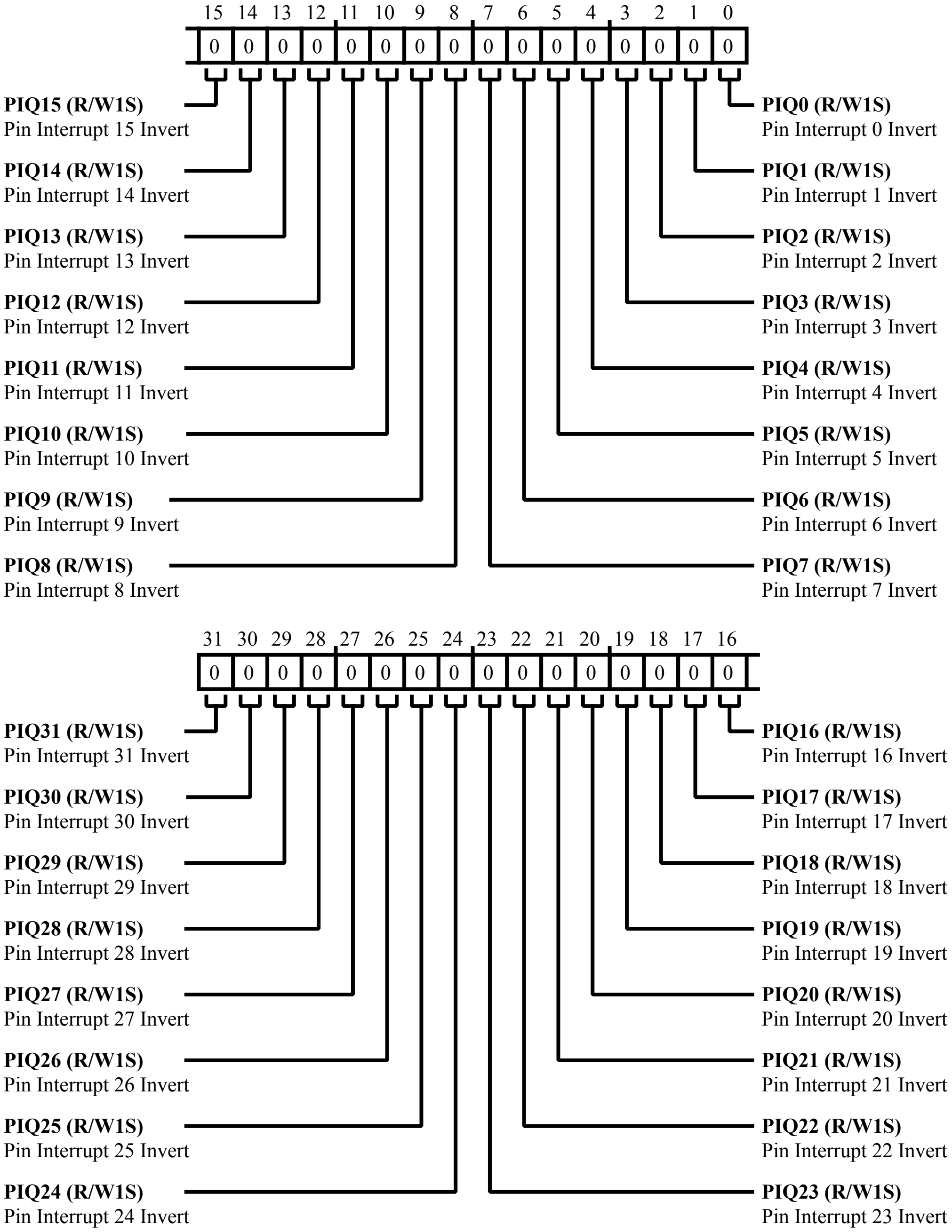

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1S)
Pin Interrupt 15 Invert
PIQ14 (R/W1S)
Pin Interrupt 14 Invert
PIQ13 (R/W1S)
Pin Interrupt 13 Invert
PIQ12 (R/W1S)
Pin Interrupt 12 Invert
PIQ11 (R/W1S)
Pin Interrupt 11 Invert
PIQ10 (R/W1S)
Pin Interrupt 10 Invert
PIQ9 (R/W1S)
Pin Interrupt 9 Invert
PIQ8 (R/W1S)
Pin Interrupt 8 Invert
PIQ0 (R/W1S)
Pin Interrupt 0 Invert
PIQ1 (R/W1S)
Pin Interrupt 1 Invert
PIQ2 (R/W1S)
Pin Interrupt 2 Invert
PIQ3 (R/W1S)
Pin Interrupt 3 Invert
PIQ4 (R/W1S)
Pin Interrupt 4 Invert
PIQ5 (R/W1S)
Pin Interrupt 5 Invert
PIQ6 (R/W1S)
Pin Interrupt 6 Invert
PIQ7 (R/W1S)
Pin Interrupt 7 Invert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1S)
Pin Interrupt 31 Invert
PIQ30 (R/W1S)
Pin Interrupt 30 Invert
PIQ29 (R/W1S)
Pin Interrupt 29 Invert
PIQ28 (R/W1S)
Pin Interrupt 28 Invert
PIQ27 (R/W1S)
Pin Interrupt 27 Invert
PIQ26 (R/W1S)
Pin Interrupt 26 Invert
PIQ25 (R/W1S)
Pin Interrupt 25 Invert
PIQ24 (R/W1S)
Pin Interrupt 24 Invert
PIQ16 (R/W1S)
Pin Interrupt 16 Invert
PIQ17 (R/W1S)
Pin Interrupt 17 Invert
PIQ18 (R/W1S)
Pin Interrupt 18 Invert
PIQ19 (R/W1S)
Pin Interrupt 19 Invert
PIQ20 (R/W1S)
Pin Interrupt 20 Invert
PIQ21 (R/W1S)
Pin Interrupt 21 Invert
PIQ22 (R/W1S)
Pin Interrupt 22 Invert
PIQ23 (R/W1S)
Pin Interrupt 23 Invert

Figure 14-29: PINT_INV_SET Register Diagram

Table 14-31: PINT_INV_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Invert. Set the `PINT_INV_SET.PIQ31` bit to enable inverted input.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Invert. Set the `PINT_INV_SET.PIQ30` bit to enable inverted input.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Invert. Set the `PINT_INV_SET.PIQ29` bit to enable inverted input.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Invert. Set the `PINT_INV_SET.PIQ28` bit to enable inverted input.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Invert. Set the `PINT_INV_SET.PIQ27` bit to enable inverted input.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Invert. Set the `PINT_INV_SET.PIQ26` bit to enable inverted input.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Invert. Set the `PINT_INV_SET.PIQ25` bit to enable inverted input.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Invert. Set the `PINT_INV_SET.PIQ24` bit to enable inverted input.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Invert. Set the `PINT_INV_SET.PIQ23` bit to enable inverted input.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Invert. Set the `PINT_INV_SET.PIQ22` bit to enable inverted input.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Invert. Set the `PINT_INV_SET.PIQ21` bit to enable inverted input.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Invert. Set the `PINT_INV_SET.PIQ20` bit to enable inverted input.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Invert. Set the `PINT_INV_SET.PIQ19` bit to enable inverted input.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Invert. Set the `PINT_INV_SET.PIQ18` bit to enable inverted input.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Invert. Set the `PINT_INV_SET.PIQ17` bit to enable inverted input.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Invert. Set the `PINT_INV_SET.PIQ16` bit to enable inverted input.

Table 14-31: PINT_INV_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Invert. Set the `PINT_INV_SET.PIQ15` bit to enable inverted input.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Invert. Set the `PINT_INV_SET.PIQ14` bit to enable inverted input.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Invert. Set the `PINT_INV_SET.PIQ13` bit to enable inverted input.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Invert. Set the `PINT_INV_SET.PIQ12` bit to enable inverted input.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Invert. Set the `PINT_INV_SET.PIQ11` bit to enable inverted input.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Invert. Set the `PINT_INV_SET.PIQ10` bit to enable inverted input.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Invert. Set the `PINT_INV_SET.PIQ9` bit to enable inverted input.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Invert. Set the `PINT_INV_SET.PIQ8` bit to enable inverted input.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Invert. Set the `PINT_INV_SET.PIQ7` bit to enable inverted input.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Invert. Set the `PINT_INV_SET.PIQ6` bit to enable inverted input.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Invert. Set the `PINT_INV_SET.PIQ5` bit to enable inverted input.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Invert. Set the `PINT_INV_SET.PIQ4` bit to enable inverted input.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Invert. Set the `PINT_INV_SET.PIQ3` bit to enable inverted input.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Invert. Set the `PINT_INV_SET.PIQ2` bit to enable inverted input.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Invert. Set the `PINT_INV_SET.PIQ1` bit to enable inverted input.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Invert. Set the `PINT_INV_SET.PIQ0` bit to enable inverted input.

PINT Latch Register

The `PINT_LATCH` register indicates the interrupt latch status for pin interrupts. When set, an interrupt request is latched. When cleared, there is no interrupt request latched.

Both the `PINT_REQ` and `PINT_LATCH` registers indicate whether an interrupt request is latched on the respective pin. The `PINT_LATCH` register is a latch that operates regardless of the interrupt masks. Bits of the `PINT_REQ` register depend on the mask register. The `PINT_REQ` register is a logical AND of the `PINT_LATCH` register and the interrupt mask.

Having two separate registers here enables the user to interrogate certain pins in polling mode while others work in interrupt mode. The `PINT_LATCH` registers can be used for edge detection or pin activity detection.

Both registers have W1C behavior. Writing a 1 to either register clears the respective bits in both registers. For interrupt operation, the user may prefer to W1C the `PINT_REQ` register (address still loaded in Px pointer). In polling mode, it might be cleaner to W1C the `PINT_LATCH` register.

Whether in edge-sensitive mode or level-sensitive mode, `PINT_LATCH` bits are never cleared by hardware except at system reset. Even in level-sensitive mode, the `PINT_LATCH` register functions as latch.

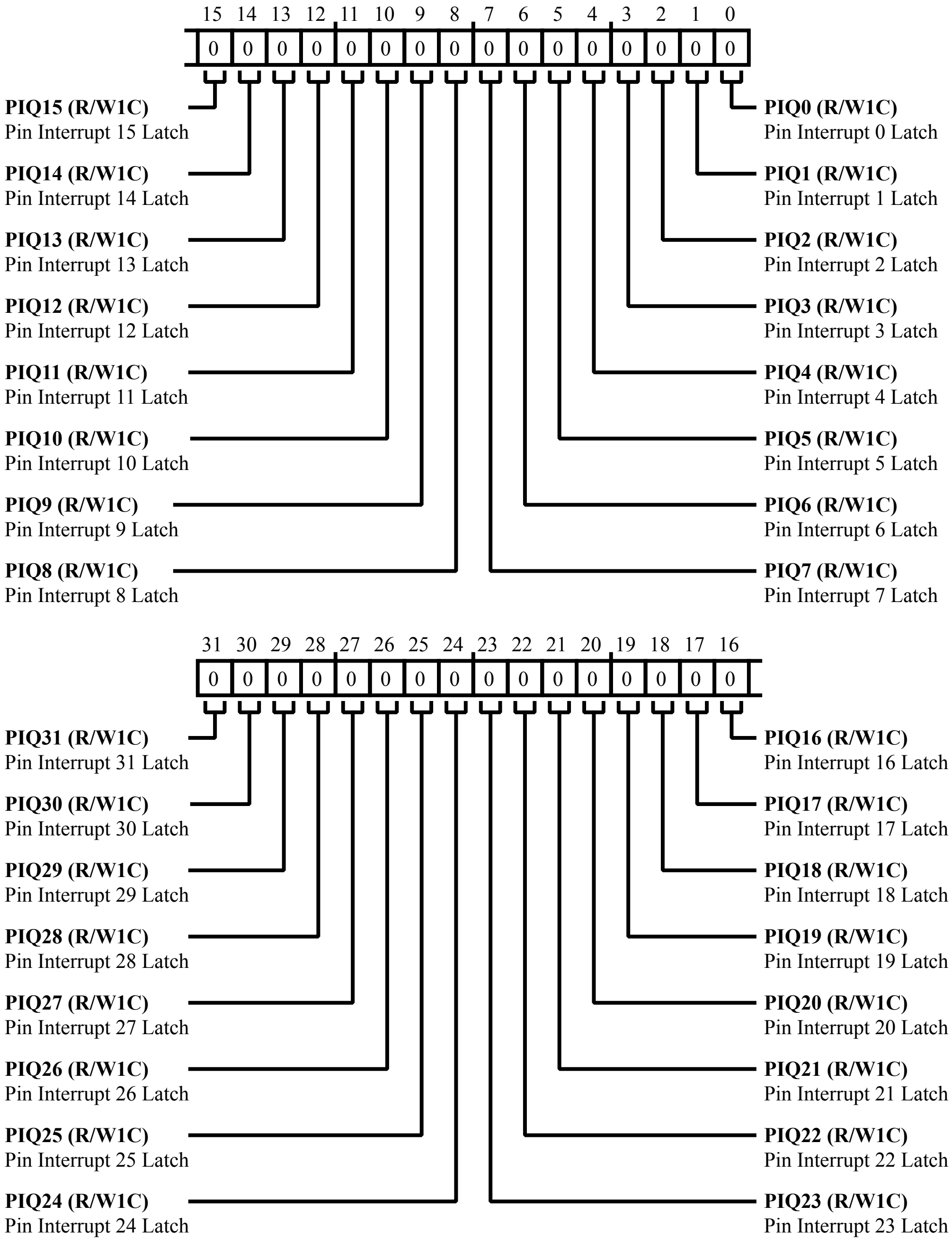

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1C)
Pin Interrupt 15 Latch
PIQ14 (R/W1C)
Pin Interrupt 14 Latch
PIQ13 (R/W1C)
Pin Interrupt 13 Latch
PIQ12 (R/W1C)
Pin Interrupt 12 Latch
PIQ11 (R/W1C)
Pin Interrupt 11 Latch
PIQ10 (R/W1C)
Pin Interrupt 10 Latch
PIQ9 (R/W1C)
Pin Interrupt 9 Latch
PIQ8 (R/W1C)
Pin Interrupt 8 Latch
PIQ0 (R/W1C)
Pin Interrupt 0 Latch
PIQ1 (R/W1C)
Pin Interrupt 1 Latch
PIQ2 (R/W1C)
Pin Interrupt 2 Latch
PIQ3 (R/W1C)
Pin Interrupt 3 Latch
PIQ4 (R/W1C)
Pin Interrupt 4 Latch
PIQ5 (R/W1C)
Pin Interrupt 5 Latch
PIQ6 (R/W1C)
Pin Interrupt 6 Latch
PIQ7 (R/W1C)
Pin Interrupt 7 Latch
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1C)
Pin Interrupt 31 Latch
PIQ30 (R/W1C)
Pin Interrupt 30 Latch
PIQ29 (R/W1C)
Pin Interrupt 29 Latch
PIQ28 (R/W1C)
Pin Interrupt 28 Latch
PIQ27 (R/W1C)
Pin Interrupt 27 Latch
PIQ26 (R/W1C)
Pin Interrupt 26 Latch
PIQ25 (R/W1C)
Pin Interrupt 25 Latch
PIQ24 (R/W1C)
Pin Interrupt 24 Latch
PIQ16 (R/W1C)
Pin Interrupt 16 Latch
PIQ17 (R/W1C)
Pin Interrupt 17 Latch
PIQ18 (R/W1C)
Pin Interrupt 18 Latch
PIQ19 (R/W1C)
Pin Interrupt 19 Latch
PIQ20 (R/W1C)
Pin Interrupt 20 Latch
PIQ21 (R/W1C)
Pin Interrupt 21 Latch
PIQ22 (R/W1C)
Pin Interrupt 22 Latch
PIQ23 (R/W1C)
Pin Interrupt 23 Latch

Figure 14-30: PINT_LATCH Register Diagram

Table 14-32: PINT_LATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Latch. If the `PINT_LATCH.PIQ31` bit is set, the request is latched.

Table 14-32: PINT_LATCH Register Fields (Continued)

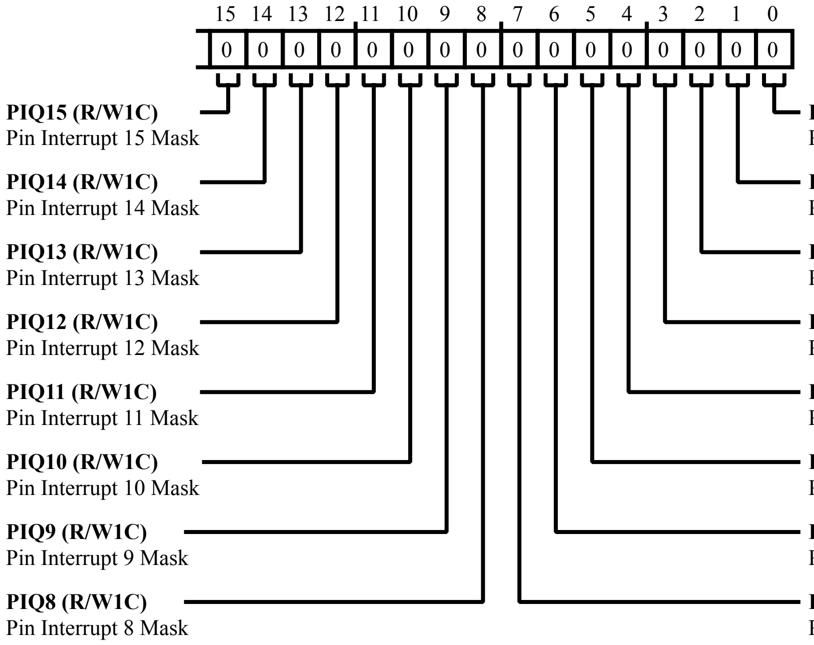
Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Latch. If the `PINT_LATCH.PIQ30` bit is set, the request is latched.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Latch. If the `PINT_LATCH.PIQ29` bit is set, the request is latched.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Latch. If the `PINT_LATCH.PIQ28` bit is set, the request is latched.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Latch. If the `PINT_LATCH.PIQ27` bit is set, the request is latched.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Latch. If the `PINT_LATCH.PIQ26` bit is set, the request is latched.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Latch. If the `PINT_LATCH.PIQ25` bit is set, the request is latched.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Latch. If the `PINT_LATCH.PIQ24` bit is set, the request is latched.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Latch. If the `PINT_LATCH.PIQ23` bit is set, the request is latched.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Latch. If the `PINT_LATCH.PIQ22` bit is set, the request is latched.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Latch. If the `PINT_LATCH.PIQ21` bit is set, the request is latched.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Latch. If the `PINT_LATCH.PIQ20` bit is set, the request is latched.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Latch. If the `PINT_LATCH.PIQ19` bit is set, the request is latched.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Latch. If the `PINT_LATCH.PIQ18` bit is set, the request is latched.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Latch. If the `PINT_LATCH.PIQ17` bit is set, the request is latched.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Latch. If the `PINT_LATCH.PIQ16` bit is set, the request is latched.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Latch. If the `PINT_LATCH.PIQ15` bit is set, the request is latched.

Table 14-32: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Latch. If the `PINT_LATCH.PIQ14` bit is set, the request is latched.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Latch. If the `PINT_LATCH.PIQ13` bit is set, the request is latched.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Latch. If the `PINT_LATCH.PIQ12` bit is set, the request is latched.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Latch. If the `PINT_LATCH.PIQ11` bit is set, the request is latched.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Latch. If the `PINT_LATCH.PIQ10` bit is set, the request is latched.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Latch. If the `PINT_LATCH.PIQ9` bit is set, the request is latched.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Latch. If the `PINT_LATCH.PIQ8` bit is set, the request is latched.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Latch. If the `PINT_LATCH.PIQ7` bit is set, the request is latched.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Latch. If the `PINT_LATCH.PIQ6` bit is set, the request is latched.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Latch. If the `PINT_LATCH.PIQ5` bit is set, the request is latched.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Latch. If the `PINT_LATCH.PIQ4` bit is set, the request is latched.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Latch. If the `PINT_LATCH.PIQ3` bit is set, the request is latched.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Latch. If the `PINT_LATCH.PIQ2` bit is set, the request is latched.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Latch. If the `PINT_LATCH.PIQ1` bit is set, the request is latched.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Latch. If the `PINT_LATCH.PIQ0` bit is set, the request is latched.

PINT Mask Clear Register

The `PINT_MSK_CLR` register permits masking (disabling) of interrupt requests. Writing 1 to a bit in `PINT_MSK_CLR` masks the corresponding pin interrupt.


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1C)
Pin Interrupt 15 Mask
PIQ14 (R/W1C)
Pin Interrupt 14 Mask
PIQ13 (R/W1C)
Pin Interrupt 13 Mask
PIQ12 (R/W1C)
Pin Interrupt 12 Mask
PIQ11 (R/W1C)
Pin Interrupt 11 Mask
PIQ10 (R/W1C)
Pin Interrupt 10 Mask
PIQ9 (R/W1C)
Pin Interrupt 9 Mask
PIQ8 (R/W1C)
Pin Interrupt 8 Mask
PIQ0 (R/W1C)
Pin Interrupt 0 Mask
PIQ1 (R/W1C)
Pin Interrupt 1 Mask
PIQ2 (R/W1C)
Pin Interrupt 2 Mask
PIQ3 (R/W1C)
Pin Interrupt 3 Mask
PIQ4 (R/W1C)
Pin Interrupt 4 Mask
PIQ5 (R/W1C)
Pin Interrupt 5 Mask
PIQ6 (R/W1C)
Pin Interrupt 6 Mask
PIQ7 (R/W1C)
Pin Interrupt 7 Mask

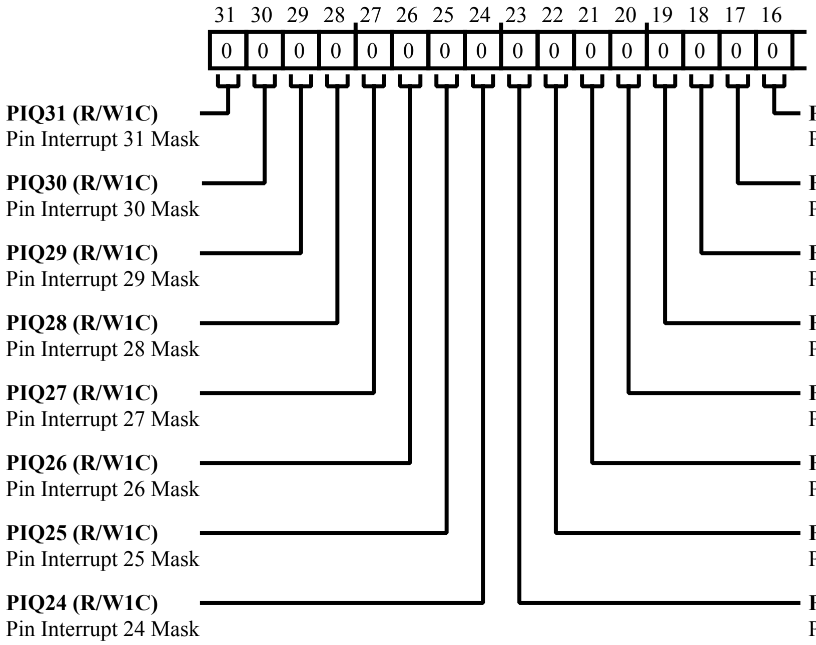

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1C)
Pin Interrupt 31 Mask
PIQ30 (R/W1C)
Pin Interrupt 30 Mask
PIQ29 (R/W1C)
Pin Interrupt 29 Mask
PIQ28 (R/W1C)
Pin Interrupt 28 Mask
PIQ27 (R/W1C)
Pin Interrupt 27 Mask
PIQ26 (R/W1C)
Pin Interrupt 26 Mask
PIQ25 (R/W1C)
Pin Interrupt 25 Mask
PIQ24 (R/W1C)
Pin Interrupt 24 Mask
PIQ16 (R/W1C)
Pin Interrupt 16 Mask
PIQ17 (R/W1C)
Pin Interrupt 17 Mask
PIQ18 (R/W1C)
Pin Interrupt 18 Mask
PIQ19 (R/W1C)
Pin Interrupt 19 Mask
PIQ20 (R/W1C)
Pin Interrupt 20 Mask
PIQ21 (R/W1C)
Pin Interrupt 21 Mask
PIQ22 (R/W1C)
Pin Interrupt 22 Mask
PIQ23 (R/W1C)
Pin Interrupt 23 Mask

Figure 14-31: PINT_MSK_CLR Register Diagram

Table 14-33: PINT_MSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Mask. Set the `PINT_MSK_CLR.PIQ31` bit to disable the interrupt.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Mask. Set the `PINT_MSK_CLR.PIQ30` bit to disable the interrupt.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Mask. Set the `PINT_MSK_CLR.PIQ29` bit to disable the interrupt.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Mask. Set the `PINT_MSK_CLR.PIQ28` bit to disable the interrupt.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Mask. Set the `PINT_MSK_CLR.PIQ27` bit to disable the interrupt.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Mask. Set the `PINT_MSK_CLR.PIQ26` bit to disable the interrupt.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Mask. Set the `PINT_MSK_CLR.PIQ25` bit to disable the interrupt.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Mask. Set the `PINT_MSK_CLR.PIQ24` bit to disable the interrupt.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Mask. Set the `PINT_MSK_CLR.PIQ23` bit to disable the interrupt.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Mask. Set the `PINT_MSK_CLR.PIQ22` bit to disable the interrupt.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Mask. Set the `PINT_MSK_CLR.PIQ21` bit to disable the interrupt.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Mask. Set the `PINT_MSK_CLR.PIQ20` bit to disable the interrupt.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Mask. Set the `PINT_MSK_CLR.PIQ19` bit to disable the interrupt.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Mask. Set the `PINT_MSK_CLR.PIQ18` bit to disable the interrupt.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Mask. Set the `PINT_MSK_CLR.PIQ17` bit to disable the interrupt.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Mask. Set the `PINT_MSK_CLR.PIQ16` bit to disable the interrupt.

Table 14-33: PINT_MSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Mask. Set the `PINT_MSK_CLR.PIQ15` bit to disable the interrupt.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Mask. Set the `PINT_MSK_CLR.PIQ14` bit to disable the interrupt.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Mask. Set the `PINT_MSK_CLR.PIQ13` bit to disable the interrupt.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Mask. Set the `PINT_MSK_CLR.PIQ12` bit to disable the interrupt.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Mask. Set the `PINT_MSK_CLR.PIQ11` bit to disable the interrupt.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Mask. Set the `PINT_MSK_CLR.PIQ10` bit to disable the interrupt.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Mask. Set the `PINT_MSK_CLR.PIQ9` bit to disable the interrupt.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Mask. Set the `PINT_MSK_CLR.PIQ8` bit to disable the interrupt.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Mask. Set the `PINT_MSK_CLR.PIQ7` bit to disable the interrupt.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Mask. Set the `PINT_MSK_CLR.PIQ6` bit to disable the interrupt.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Mask. Set the `PINT_MSK_CLR.PIQ5` bit to disable the interrupt.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Mask. Set the `PINT_MSK_CLR.PIQ4` bit to disable the interrupt.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Mask. Set the `PINT_MSK_CLR.PIQ3` bit to disable the interrupt.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Mask. Set the `PINT_MSK_CLR.PIQ2` bit to disable the interrupt.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Mask. Set the `PINT_MSK_CLR.PIQ1` bit to disable the interrupt.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Mask. Set the `PINT_MSK_CLR.PIQ0` bit to disable the interrupt.

PINT Mask Set Register

The `PINT_MSK_SET` register permits unmasking (enabling) of interrupt requests. Writing 1 to a bit in `PINT_MSK_SET` unmasks the corresponding pin interrupt.

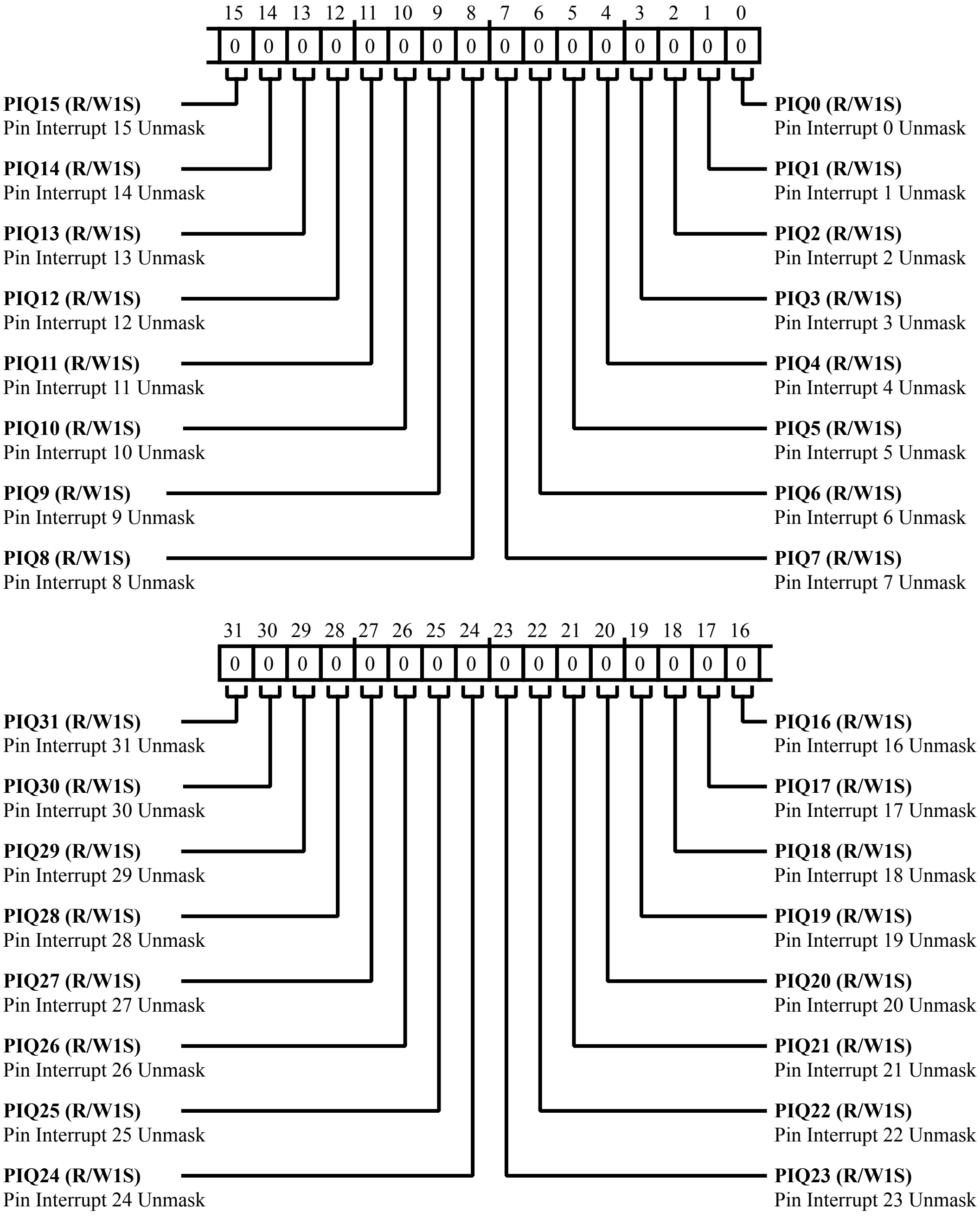

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1S)
Pin Interrupt 15 Unmask
PIQ14 (R/W1S)
Pin Interrupt 14 Unmask
PIQ13 (R/W1S)
Pin Interrupt 13 Unmask
PIQ12 (R/W1S)
Pin Interrupt 12 Unmask
PIQ11 (R/W1S)
Pin Interrupt 11 Unmask
PIQ10 (R/W1S)
Pin Interrupt 10 Unmask
PIQ9 (R/W1S)
Pin Interrupt 9 Unmask
PIQ8 (R/W1S)
Pin Interrupt 8 Unmask
PIQ0 (R/W1S)
Pin Interrupt 0 Unmask
PIQ1 (R/W1S)
Pin Interrupt 1 Unmask
PIQ2 (R/W1S)
Pin Interrupt 2 Unmask
PIQ3 (R/W1S)
Pin Interrupt 3 Unmask
PIQ4 (R/W1S)
Pin Interrupt 4 Unmask
PIQ5 (R/W1S)
Pin Interrupt 5 Unmask
PIQ6 (R/W1S)
Pin Interrupt 6 Unmask
PIQ7 (R/W1S)
Pin Interrupt 7 Unmask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1S)
Pin Interrupt 31 Unmask
PIQ30 (R/W1S)
Pin Interrupt 30 Unmask
PIQ29 (R/W1S)
Pin Interrupt 29 Unmask
PIQ28 (R/W1S)
Pin Interrupt 28 Unmask
PIQ27 (R/W1S)
Pin Interrupt 27 Unmask
PIQ26 (R/W1S)
Pin Interrupt 26 Unmask
PIQ25 (R/W1S)
Pin Interrupt 25 Unmask
PIQ24 (R/W1S)
Pin Interrupt 24 Unmask
PIQ16 (R/W1S)
Pin Interrupt 16 Unmask
PIQ17 (R/W1S)
Pin Interrupt 17 Unmask
PIQ18 (R/W1S)
Pin Interrupt 18 Unmask
PIQ19 (R/W1S)
Pin Interrupt 19 Unmask
PIQ20 (R/W1S)
Pin Interrupt 20 Unmask
PIQ21 (R/W1S)
Pin Interrupt 21 Unmask
PIQ22 (R/W1S)
Pin Interrupt 22 Unmask
PIQ23 (R/W1S)
Pin Interrupt 23 Unmask

Figure 14-32: PINT_MSK_SET Register Diagram

Table 14-34: PINT_MSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Unmask. Set the `PINT_MSK_SET.PIQ31` bit to enable the interrupt.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Unmask. Set the `PINT_MSK_SET.PIQ30` bit to enable the interrupt.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Unmask. Set the `PINT_MSK_SET.PIQ29` bit to enable the interrupt.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Unmask. Set the `PINT_MSK_SET.PIQ28` bit to enable the interrupt.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Unmask. Set the `PINT_MSK_SET.PIQ27` bit to enable the interrupt.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Unmask. Set the `PINT_MSK_SET.PIQ26` bit to enable the interrupt.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Unmask. Set the `PINT_MSK_SET.PIQ25` bit to enable the interrupt.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Unmask. Set the `PINT_MSK_SET.PIQ24` bit to enable the interrupt.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Unmask. Set the `PINT_MSK_SET.PIQ23` bit to enable the interrupt.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Unmask. Set the `PINT_MSK_SET.PIQ22` bit to enable the interrupt.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Unmask. Set the `PINT_MSK_SET.PIQ21` bit to enable the interrupt.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Unmask. Set the `PINT_MSK_SET.PIQ20` bit to enable the interrupt.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Unmask. Set the `PINT_MSK_SET.PIQ19` bit to enable the interrupt.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Unmask. Set the `PINT_MSK_SET.PIQ18` bit to enable the interrupt.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Unmask. Set the `PINT_MSK_SET.PIQ17` bit to enable the interrupt.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Unmask. Set the `PINT_MSK_SET.PIQ16` bit to enable the interrupt.

Table 14-34: PINT_MSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Unmask. Set the `PINT_MSK_SET.PIQ15` bit to enable the interrupt.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Unmask. Set the `PINT_MSK_SET.PIQ14` bit to enable the interrupt.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Unmask. Set the `PINT_MSK_SET.PIQ13` bit to enable the interrupt.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Unmask. Set the `PINT_MSK_SET.PIQ12` bit to enable the interrupt.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Unmask. Set the `PINT_MSK_SET.PIQ11` bit to enable the interrupt.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Unmask. Set the `PINT_MSK_SET.PIQ10` bit to enable the interrupt.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Unmask. Set the `PINT_MSK_SET.PIQ9` bit to enable the interrupt.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Unmask. Set the `PINT_MSK_SET.PIQ8` bit to enable the interrupt.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Unmask. Set the `PINT_MSK_SET.PIQ7` bit to enable the interrupt.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Unmask. Set the `PINT_MSK_SET.PIQ6` bit to enable the interrupt.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Unmask. Set the `PINT_MSK_SET.PIQ5` bit to enable the interrupt.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Unmask. Set the `PINT_MSK_SET.PIQ4` bit to enable the interrupt.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Unmask. Set the `PINT_MSK_SET.PIQ3` bit to enable the interrupt.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Unmask. Set the `PINT_MSK_SET.PIQ2` bit to enable the interrupt.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Unmask. Set the `PINT_MSK_SET.PIQ1` bit to enable the interrupt.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Unmask. Set the `PINT_MSK_SET.PIQ0` bit to enable the interrupt.

PINT Pin State Register

When a half port is assigned to a byte in any PINT block, the state of the eight pins (regardless of GPIO or function, input or output) can be seen in the `PINT_PINSTATE` register. While neither input nor output drivers of the pin are enabled, reads of the pin state in `PINT_PINSTATE` return zero. The `PINT_PINSTATE` register reports the inverted state of the pin if the signal inverter is activated by the `PINT_INV_SET` register. The inverter can be enabled on an individual bit-by-bit basis. Every bit in the `PINT_INV_SET` and `PINT_INV_CLR` register pair represents a pin signal.

The pin interrupt pin state registers enable the service routine to read the current state of the pin without reading from GPIO space. If there was an edge-sensitive interrupt, the service routine can check whether the state of the pin is still high or turned low.

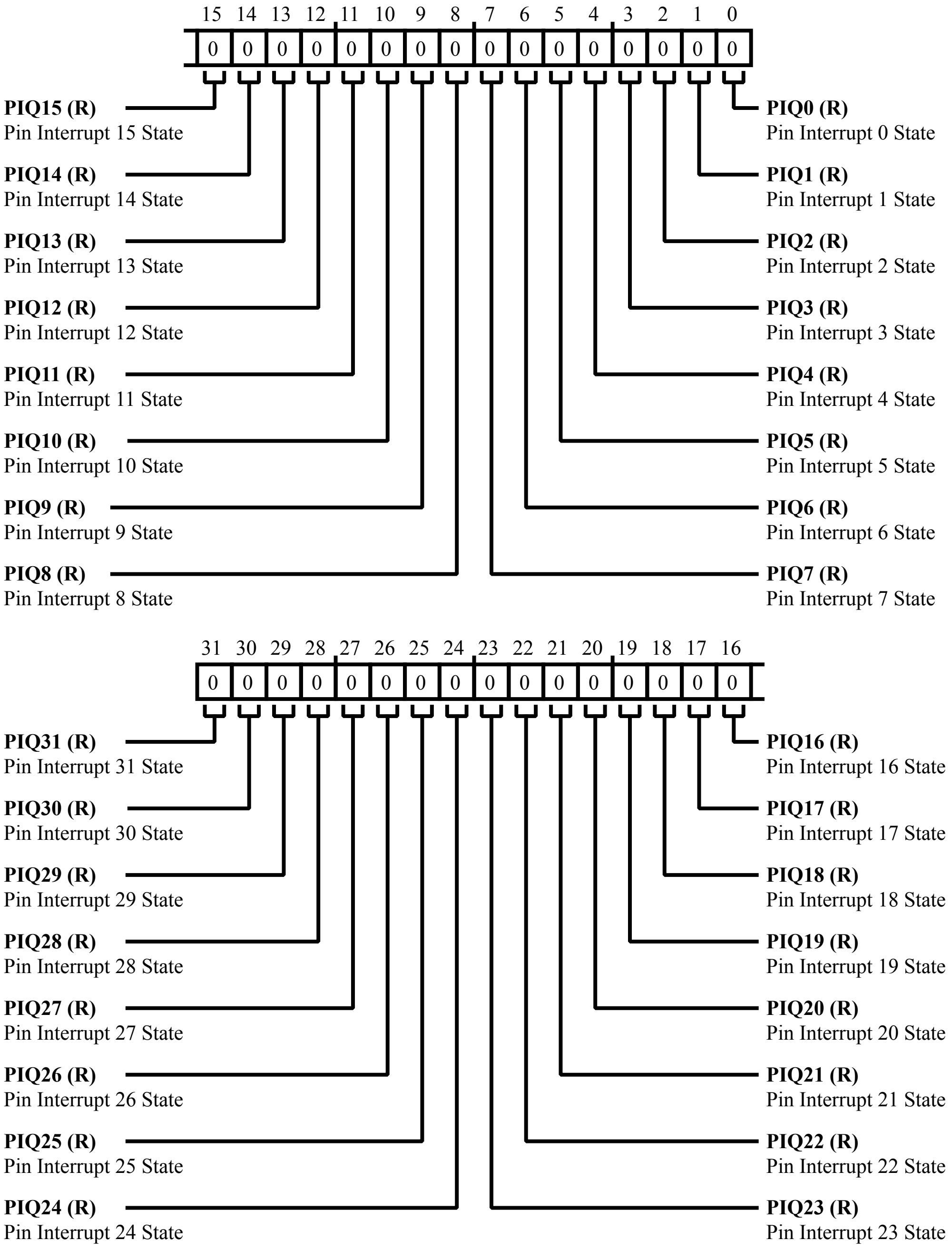

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R)
Pin Interrupt 15 State
PIQ14 (R)
Pin Interrupt 14 State
PIQ13 (R)
Pin Interrupt 13 State
PIQ12 (R)
Pin Interrupt 12 State
PIQ11 (R)
Pin Interrupt 11 State
PIQ10 (R)
Pin Interrupt 10 State
PIQ9 (R)
Pin Interrupt 9 State
PIQ8 (R)
Pin Interrupt 8 State
PIQ0 (R)
Pin Interrupt 0 State
PIQ1 (R)
Pin Interrupt 1 State
PIQ2 (R)
Pin Interrupt 2 State
PIQ3 (R)
Pin Interrupt 3 State
PIQ4 (R)
Pin Interrupt 4 State
PIQ5 (R)
Pin Interrupt 5 State
PIQ6 (R)
Pin Interrupt 6 State
PIQ7 (R)
Pin Interrupt 7 State
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R)
Pin Interrupt 31 State
PIQ30 (R)
Pin Interrupt 30 State
PIQ29 (R)
Pin Interrupt 29 State
PIQ28 (R)
Pin Interrupt 28 State
PIQ27 (R)
Pin Interrupt 27 State
PIQ26 (R)
Pin Interrupt 26 State
PIQ25 (R)
Pin Interrupt 25 State
PIQ24 (R)
Pin Interrupt 24 State
PIQ16 (R)
Pin Interrupt 16 State
PIQ17 (R)
Pin Interrupt 17 State
PIQ18 (R)
Pin Interrupt 18 State
PIQ19 (R)
Pin Interrupt 19 State
PIQ20 (R)
Pin Interrupt 20 State
PIQ21 (R)
Pin Interrupt 21 State
PIQ22 (R)
Pin Interrupt 22 State
PIQ23 (R)
Pin Interrupt 23 State

Figure 14-33: PINT_PINSTATE Register Diagram

Table 14-35: PINT_PINSTATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	PIQ31	Pin Interrupt 31 State. A read of the `PINT_PINSTATE.PIQ31` bit returns the pin state.

Table 14-35: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	PIQ30	Pin Interrupt 30 State. A read of the `PINT_PINSTATE.PIQ30` bit returns the pin state.
29 (R/NW)	PIQ29	Pin Interrupt 29 State. A read of the `PINT_PINSTATE.PIQ29` bit returns the pin state.
28 (R/NW)	PIQ28	Pin Interrupt 28 State. A read of the `PINT_PINSTATE.PIQ28` bit returns the pin state.
27 (R/NW)	PIQ27	Pin Interrupt 27 State. A read of the `PINT_PINSTATE.PIQ27` bit returns the pin state.
26 (R/NW)	PIQ26	Pin Interrupt 26 State. A read of the `PINT_PINSTATE.PIQ26` bit returns the pin state.
25 (R/NW)	PIQ25	Pin Interrupt 25 State. A read of the `PINT_PINSTATE.PIQ25` bit returns the pin state.
24 (R/NW)	PIQ24	Pin Interrupt 24 State. A read of the `PINT_PINSTATE.PIQ24` bit returns the pin state.
23 (R/NW)	PIQ23	Pin Interrupt 23 State. A read of the `PINT_PINSTATE.PIQ23` bit returns the pin state.
22 (R/NW)	PIQ22	Pin Interrupt 22 State. A read of the `PINT_PINSTATE.PIQ22` bit returns the pin state.
21 (R/NW)	PIQ21	Pin Interrupt 21 State. A read of the `PINT_PINSTATE.PIQ21` bit returns the pin state.
20 (R/NW)	PIQ20	Pin Interrupt 20 State. A read of the `PINT_PINSTATE.PIQ20` bit returns the pin state.
19 (R/NW)	PIQ19	Pin Interrupt 19 State. A read of the `PINT_PINSTATE.PIQ19` bit returns the pin state.
18 (R/NW)	PIQ18	Pin Interrupt 18 State. A read of the `PINT_PINSTATE.PIQ18` bit returns the pin state.
17 (R/NW)	PIQ17	Pin Interrupt 17 State. A read of the `PINT_PINSTATE.PIQ17` bit returns the pin state.
16 (R/NW)	PIQ16	Pin Interrupt 16 State. A read of the `PINT_PINSTATE.PIQ16` bit returns the pin state.
15 (R/NW)	PIQ15	Pin Interrupt 15 State. A read of the `PINT_PINSTATE.PIQ15` bit returns the pin state.

Table 14-35: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PIQ14	Pin Interrupt 14 State. A read of the `PINT_PINSTATE.PIQ14` bit returns the pin state.
13 (R/NW)	PIQ13	Pin Interrupt 13 State. A read of the `PINT_PINSTATE.PIQ13` bit returns the pin state.
12 (R/NW)	PIQ12	Pin Interrupt 12 State. A read of the `PINT_PINSTATE.PIQ12` bit returns the pin state.
11 (R/NW)	PIQ11	Pin Interrupt 11 State. A read of the `PINT_PINSTATE.PIQ11` bit returns the pin state.
10 (R/NW)	PIQ10	Pin Interrupt 10 State. A read of the `PINT_PINSTATE.PIQ10` bit returns the pin state.
9 (R/NW)	PIQ9	Pin Interrupt 9 State. A read of the `PINT_PINSTATE.PIQ9` bit returns the pin state.
8 (R/NW)	PIQ8	Pin Interrupt 8 State. A read of the `PINT_PINSTATE.PIQ8` bit returns the pin state.
7 (R/NW)	PIQ7	Pin Interrupt 7 State. A read of the `PINT_PINSTATE.PIQ7` bit returns the pin state.
6 (R/NW)	PIQ6	Pin Interrupt 6 State. A read of the `PINT_PINSTATE.PIQ6` bit returns the pin state.
5 (R/NW)	PIQ5	Pin Interrupt 5 State. A read of the `PINT_PINSTATE.PIQ5` bit returns the pin state.
4 (R/NW)	PIQ4	Pin Interrupt 4 State. A read of the `PINT_PINSTATE.PIQ4` bit returns the pin state.
3 (R/NW)	PIQ3	Pin Interrupt 3 State. A read of the `PINT_PINSTATE.PIQ3` bit returns the pin state.
2 (R/NW)	PIQ2	Pin Interrupt 2 State. A read of the `PINT_PINSTATE.PIQ2` bit returns the pin state.
1 (R/NW)	PIQ1	Pin Interrupt 1 State. A read of the `PINT_PINSTATE.PIQ1` bit returns the pin state.
0 (R/NW)	PIQ0	Pin Interrupt 0 State. A read of the `PINT_PINSTATE.PIQ0` bit returns the pin state.

PINT Request Register

The `PINT_REQ` register indicates the interrupt request status for pin interrupts. When set, an interrupt request is pending. When cleared, there is no interrupt request pending.

Both the `PINT_REQ` and `PINT_LATCH` registers indicate whether an interrupt request is latched on the respective pin. The `PINT_LATCH` register is a latch that operates regardless of the interrupt masks. Bits of the `PINT_REQ` register depend on the mask register. The `PINT_REQ` register is a logical AND of the `PINT_LATCH` register and the interrupt mask.

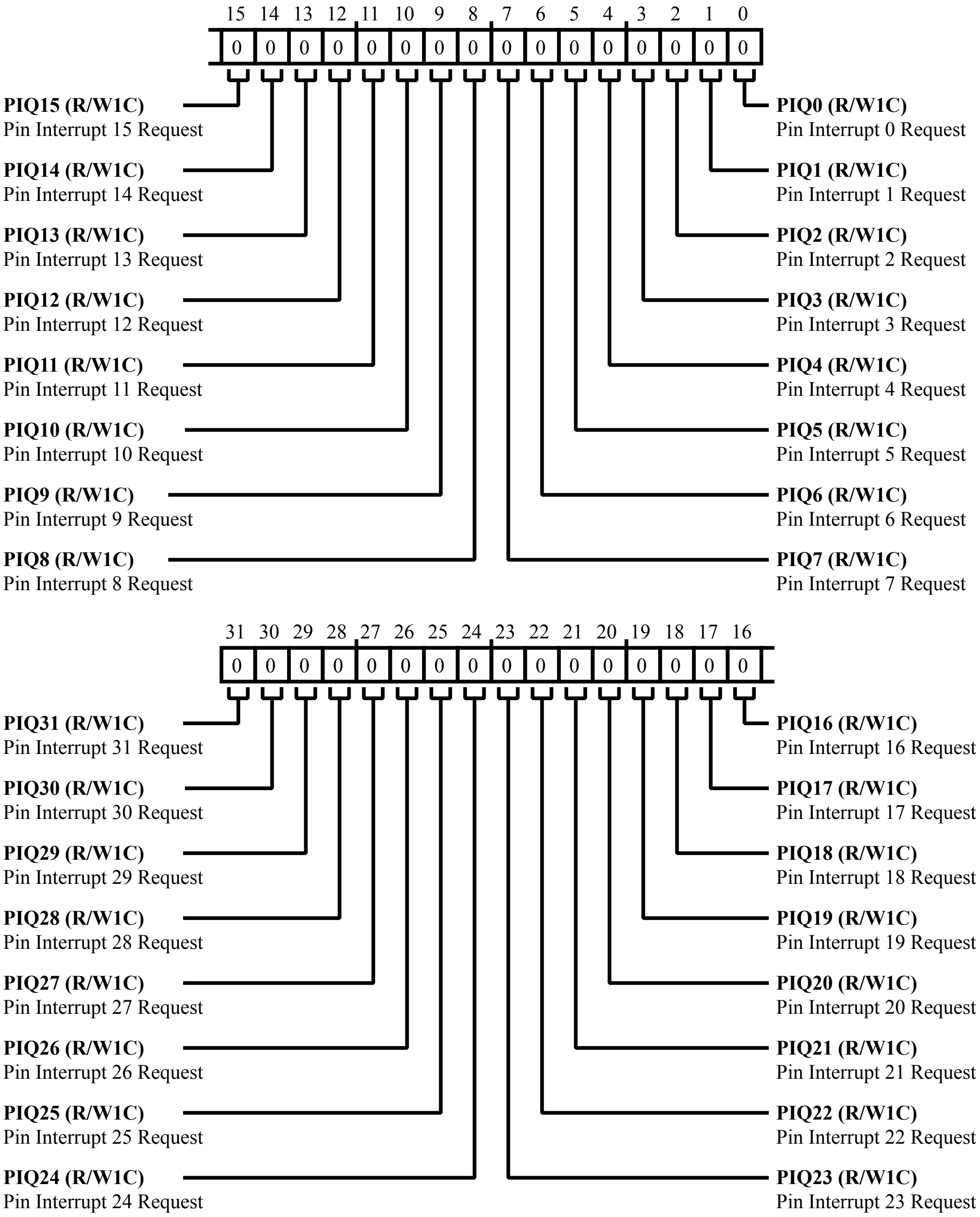

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ15 (R/W1C)
Pin Interrupt 15 Request
PIQ14 (R/W1C)
Pin Interrupt 14 Request
PIQ13 (R/W1C)
Pin Interrupt 13 Request
PIQ12 (R/W1C)
Pin Interrupt 12 Request
PIQ11 (R/W1C)
Pin Interrupt 11 Request
PIQ10 (R/W1C)
Pin Interrupt 10 Request
PIQ9 (R/W1C)
Pin Interrupt 9 Request
PIQ8 (R/W1C)
Pin Interrupt 8 Request
PIQ0 (R/W1C)
Pin Interrupt 0 Request
PIQ1 (R/W1C)
Pin Interrupt 1 Request
PIQ2 (R/W1C)
Pin Interrupt 2 Request
PIQ3 (R/W1C)
Pin Interrupt 3 Request
PIQ4 (R/W1C)
Pin Interrupt 4 Request
PIQ5 (R/W1C)
Pin Interrupt 5 Request
PIQ6 (R/W1C)
Pin Interrupt 6 Request
PIQ7 (R/W1C)
Pin Interrupt 7 Request
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIQ31 (R/W1C)
Pin Interrupt 31 Request
PIQ30 (R/W1C)
Pin Interrupt 30 Request
PIQ29 (R/W1C)
Pin Interrupt 29 Request
PIQ28 (R/W1C)
Pin Interrupt 28 Request
PIQ27 (R/W1C)
Pin Interrupt 27 Request
PIQ26 (R/W1C)
Pin Interrupt 26 Request
PIQ25 (R/W1C)
Pin Interrupt 25 Request
PIQ24 (R/W1C)
Pin Interrupt 24 Request
PIQ16 (R/W1C)
Pin Interrupt 16 Request
PIQ17 (R/W1C)
Pin Interrupt 17 Request
PIQ18 (R/W1C)
Pin Interrupt 18 Request
PIQ19 (R/W1C)
Pin Interrupt 19 Request
PIQ20 (R/W1C)
Pin Interrupt 20 Request
PIQ21 (R/W1C)
Pin Interrupt 21 Request
PIQ22 (R/W1C)
Pin Interrupt 22 Request
PIQ23 (R/W1C)
Pin Interrupt 23 Request

Figure 14-34: PINT_REQ Register Diagram

Table 14-36: PINT_REQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Request. If the `PINT_REQ.PIQ31` bit is set, a request is pending.

Table 14-36: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Request. If the `PINT_REQ.PIQ30` bit is set, a request is pending.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Request. If the `PINT_REQ.PIQ29` bit is set, a request is pending.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Request. If the `PINT_REQ.PIQ28` bit is set, a request is pending.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Request. If the `PINT_REQ.PIQ27` bit is set, a request is pending.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Request. If the `PINT_REQ.PIQ26` bit is set, a request is pending.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Request. If the `PINT_REQ.PIQ25` bit is set, a request is pending.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Request. If the `PINT_REQ.PIQ24` bit is set, a request is pending.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Request. If the `PINT_REQ.PIQ23` bit is set, a request is pending.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Request. If the `PINT_REQ.PIQ22` bit is set, a request is pending.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Request. If the `PINT_REQ.PIQ21` bit is set, a request is pending.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Request. If the `PINT_REQ.PIQ20` bit is set, a request is pending.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Request. If the `PINT_REQ.PIQ19` bit is set, a request is pending.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Request. If the `PINT_REQ.PIQ18` bit is set, a request is pending.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Request. If the `PINT_REQ.PIQ17` bit is set, a request is pending.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Request. If the `PINT_REQ.PIQ16` bit is set, a request is pending.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Request. If the `PINT_REQ.PIQ15` bit is set, a request is pending.

Table 14-36: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Request. If the `PINT_REQ.PIQ14` bit is set, a request is pending.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Request. If the `PINT_REQ.PIQ13` bit is set, a request is pending.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Request. If the `PINT_REQ.PIQ12` bit is set, a request is pending.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Request. If the `PINT_REQ.PIQ11` bit is set, a request is pending.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Request. If the `PINT_REQ.PIQ10` bit is set, a request is pending.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Request. If the `PINT_REQ.PIQ9` bit is set, a request is pending.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Request. If the `PINT_REQ.PIQ8` bit is set, a request is pending.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Request. If the `PINT_REQ.PIQ7` bit is set, a request is pending.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Request. If the `PINT_REQ.PIQ6` bit is set, a request is pending.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Request. If the `PINT_REQ.PIQ5` bit is set, a request is pending.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Request. If the `PINT_REQ.PIQ4` bit is set, a request is pending.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Request. If the `PINT_REQ.PIQ3` bit is set, a request is pending.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Request. If the `PINT_REQ.PIQ2` bit is set, a request is pending.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Request. If the `PINT_REQ.PIQ1` bit is set, a request is pending.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Request. If the `PINT_REQ.PIQ0` bit is set, a request is pending.

ADSP-SC58x PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 14-37: ADSP-SC58x PADS Register List

Name	Description
`PADS_DAI0_IE`	DAI0 Port Input Enable Control Register
`PADS_DAI1_IE`	DAI1 Port Input Enable Control Register
`PADS_PCFG0`	Peripheral PAD Configuration0 Register

DAI0 Port Input Enable Control Register

The `PADS_DAI0_IE` register configures input enable control of the DAI0 (20 pins) pads. If =0 implies input buffer disable and if =1 implies enable.

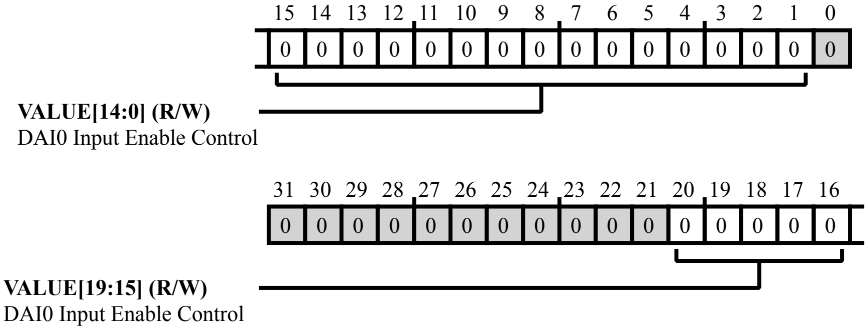

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[14:0] (R/W)
DAI0 Input Enable Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[19:15] (R/W)
DAI0 Input Enable Control

Figure 14-35: PADS_DAI0_IE Register Diagram

Table 14-38: PADS_DAI0_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20:1 (R/W)	VALUE	DAI0 Input Enable Control.

DAI1 Port Input Enable Control Register

The `PADS_DAI1_IE` register configures input enable control of the DAI1 (20 pins) pads. If =0 implies input buffer disable and if =1 implies enable.

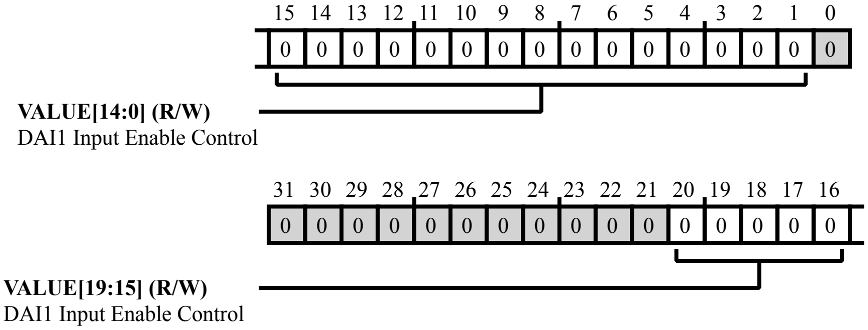

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[14:0] (R/W)
DAI1 Input Enable Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[19:15] (R/W)
DAI1 Input Enable Control

Figure 14-36: PADS_DAI1_IE Register Diagram

Table 14-39: PADS_DAI1_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
20:1 (R/W)	VALUE	DAI1 Input Enable Control.

Peripheral PAD Configuration0 Register

The `PADS_PCFG0` register provides several configuration options for the pads and multiplexing for peripherals.

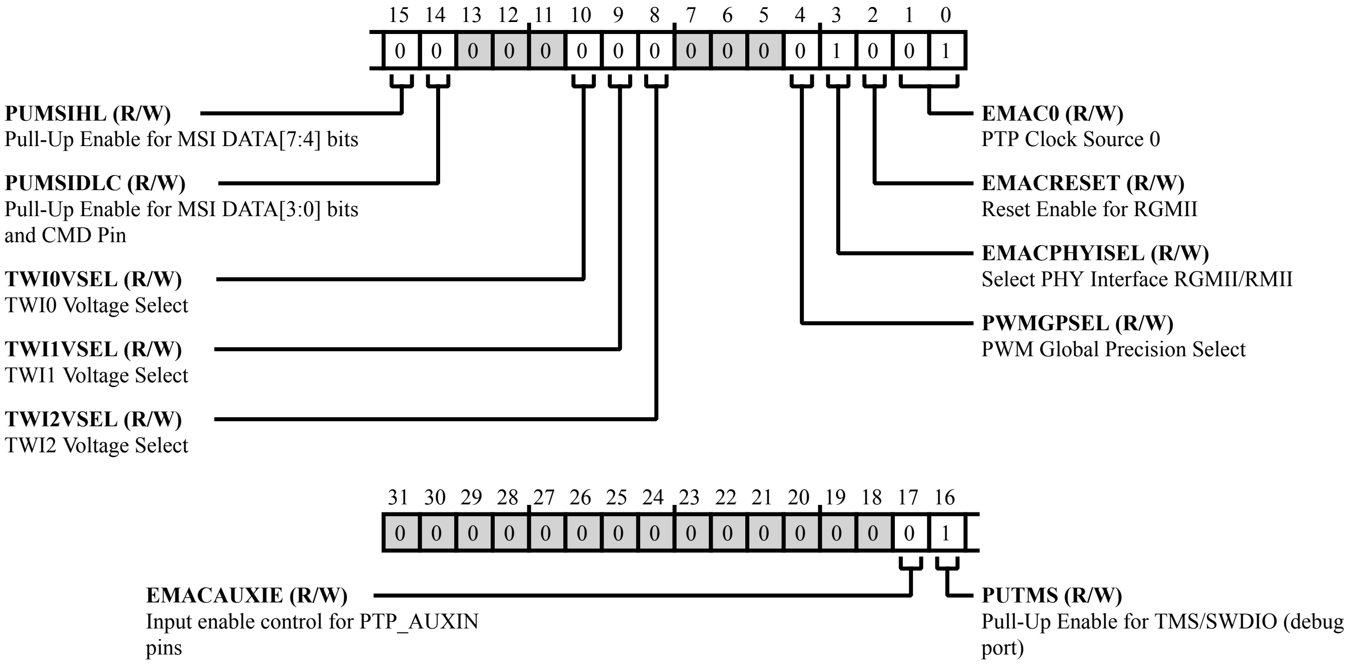

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
PUMSIHL (R/W)
Pull-Up Enable for MSI DATA[7:4] bits
PUMSIDLC (R/W)
Pull-Up Enable for MSI DATA[3:0] bits and CMD Pin
TWI0VSEL (R/W)
TWI0 Voltage Select
TWI1VSEL (R/W)
TWI1 Voltage Select
TWI2VSEL (R/W)
TWI2 Voltage Select
EMAC0 (R/W)
PTP Clock Source 0
EMACRESET (R/W)
Reset Enable for RGMII
EMACPHYISEL (R/W)
Select PHY Interface RGMII/RMII
PWMGPSEL (R/W)
PWM Global Precision Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
EMACAUXIE (R/W)
Input enable control for PTP_AUXIN pins
PUTMS (R/W)
Pull-Up Enable for TMS/SWDIO (debug port)

Figure 14-37: PADS_PCFG0 Register Diagram

Table 14-40: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	EMACAUXIE	Input enable control for PTP_AUXIN pins.	
		0	Disable input
		1	Enable input
16 (R/W)	PUTMS	Pull-Up Enable for TMS/SWDIO (debug port).	
		0	Disable pull-up
		1	Enable pull-up
15 (R/W)	PUMSIHL	Pull-Up Enable for MSI DATA[7:4] bits.	
		0	Disable pull-up
		1	Enable pull-up
14 (R/W)	PUMSIDLC	Pull-Up Enable for MSI DATA[3:0] bits and CMD Pin.	
		0	Disable pull-up
		1	Enable pull-up

Table 14-40: PADS_PCFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	TWI0VSEL	TWI0 Voltage Select. The `PADS_PCFG0.TWI0VSEL` bit selects the drive/tolerate voltage for the `TWI_SCL` and `TWI_SDA` pins for TWI0. By default this bit is cleared (=0, 3.3V).	
		0	TWI0 voltage is 3.3V
		1	TWI0 voltage is 5.0V
9 (R/W)	TWI1VSEL	TWI1 Voltage Select. The `PADS_PCFG0.TWI1VSEL` bit selects the drive/tolerate voltage for the `TWI_SCL` and `TWI_SDA` pins for TWI1. By default this bit is cleared (=0, 3.3V).	
		0	TWI1 voltage is 3.3V
		1	TWI1 voltage is 5V
8 (R/W)	TWI2VSEL	TWI2 Voltage Select. The `PADS_PCFG0.TWI2VSEL` bit selects the drive/tolerate voltage for the `TWI_SCL` and `TWI_SDA` pins for TWI2. By default this bit is cleared (=0, 3.3V).	
		0	TWI2 voltage is 3.3V
		1	TWI2 voltage is 5V
4 (R/W)	PWMGPSEL	PWM Global Precision Select. The `PADS_PCFG0.PWMGPSEL` bit selects between mixed precision and full precision on the PWM output.	
		0	Mixed precision on High vs Low outputs
		1	Heightened precision on High and Low outputs
3 (R/W)	EMACPHYISEL	Select PHY Interface RGMII/RMII.	
		0	RMII interface
		1	RGMII interface
2 (R/W)	EMACRESET	Reset Enable for RGMII. The `PADS_PCFG0.EMACRESET` bit asserts the reset on the RGMII interface To select PHY interface (RGMII or RMII), set the EMACPHYISEL bit as required and then set `PADS_PCFG0.EMACRESET`.	
		0	RGMII reset is asserted
		1	RGMII reset is de-asserted

Table 14-40: PADS_PCFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	EMAC0	PTP Clock Source 0. The `PADS_PCFG0.EMAC0` selects the clock source for the PTP Block in EMAC0.	
		0	EMAC0_RMII CLK
		1	SCLK
		2	External Clock
		3	SCLK

15 Link Port (LP)

Link ports allow the processor to connect to other processors or peripheral link ports using a simple communication protocol for high-speed parallel data transfer. This peripheral allows various I/O peripheral interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes.

The link ports of the processor support 8-bit wide data transfers. The link port pins are multiplexed in the GPIO ports. For information on processor multiplexing, see the data sheet for the specific processor.

Link ports can operate independently and simultaneously, allowing glueless high-speed connectivity of up to four external processors.

LP Features

All link ports are identical in their design and have the following common features.

- Bidirectional ports with eight data signals (`LP0_D0` - `LP0_D7`), an acknowledge signal (`LP0_ACK`), and a clock signal (`LP0_CLK`).
- Provide high-speed, point-to-point data transfers to other processors, allowing different types of interconnections between multiple processors.
- Pack data into 32-bit words. The processor can directly read this data or transfer it through DMA to or from on-chip memory.
- Support for data buffering through a 2-deep FIFO for transmit and a 4-deep FIFO for receive.
- Programmable clock and acknowledge based handshake mechanism for efficient communication.
- A dedicated DMA channel.

LP Functional Description

This section provides a description of the link port, including a list of its registers and a functional block diagram.

ADSP-SC58x LP Register List

The Link Port LP is an 8-bit wide parallel port that can connect to another processor's LP or another LP-compatible device. This port allows a variety of interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes. A set of registers governs LP operations. For more information on LP functionality, see the LP register descriptions.

Table 15-1: ADSP-SC58x LP Register List

Name	Description
`LP_CTL`	Control Register
`LP_DIV`	Clock Divider Value Register
`LP_RX`	Receive Buffer Register
`LP_STAT`	Status Register
`LP_TX`	Transmit Buffer Register
`LP_TXIN_SHDW`	Shadow Input Transmit Buffer Register
`LP_TXOUT_SHDW`	Shadow Output Transmit Buffer Register

ADSP-SC58x LP Interrupt List

Table 15-2: ADSP-SC58x LP Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
77	LP0_DMA	LP0 DMA Channel		30
78	LP0_STAT	LP0 Status		
79	LP1_DMA	LP1 DMA Channel		36
80	LP1_STAT	LP1 Status		
208	LP0_DMA_ERR	LP0 DMA Data Error		
209	LP1_DMA_ERR	LP1 DMA Data Error		

ADSP-SC58x LP Trigger List

Table 15-3: ADSP-SC58x LP Trigger List Masters

Trigger ID	Name	Description	Sensitivity
64	`LP0_DMA`	LP0 DMA Channel	
65	`LP1_DMA`	LP1 DMA Channel	

Table 15-4: ADSP-SC58x LP Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
48	`LP0_DMA`	LP0 DMA Channel	Pulse
49	`LP1_DMA`	LP1 DMA Channel	Pulse

ADSP-SC58x LP DMA Channel List

Table 15-5: ADSP-SC58x LP DMA Channel List

DMA ID	DMA Channel Name	Description
DMA30	LP0_DMA	LP0 DMA Channel
DMA36	LP1_DMA	LP1 DMA Channel

Block Diagram

The ***Link Port Block Diagram*** shows the block diagram of a link port.

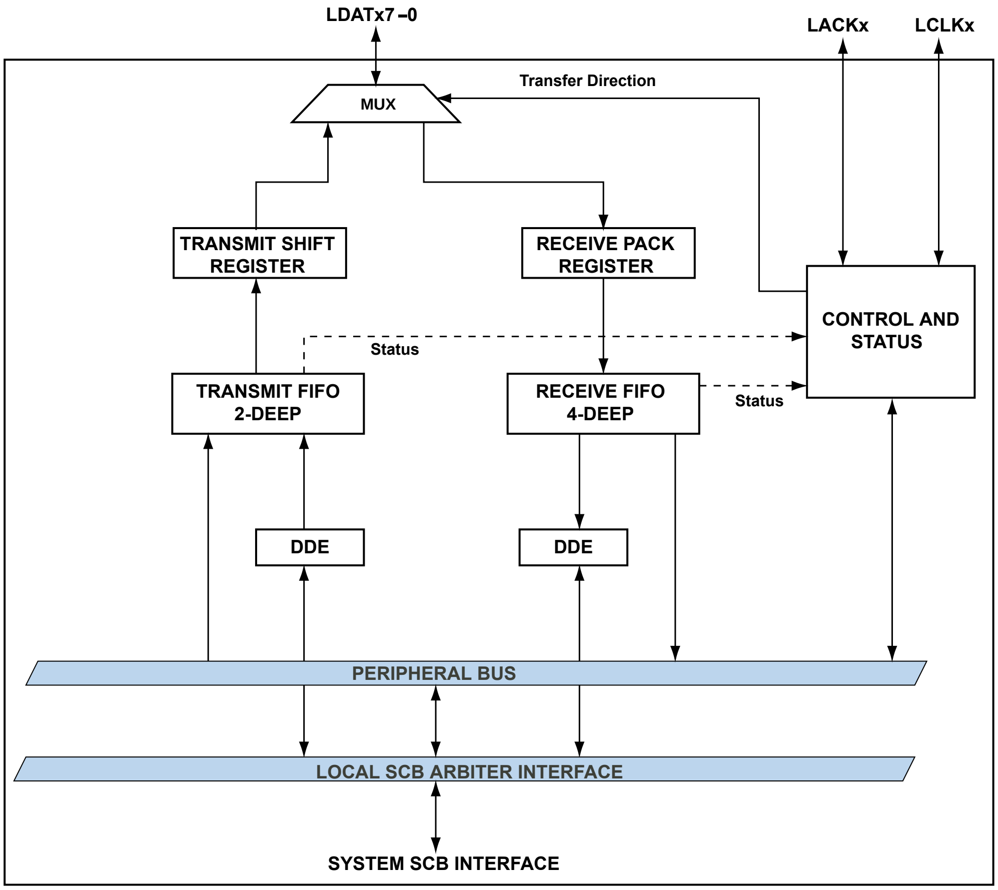

LDATx7 –0
LACKx
LCLKx
Transfer Direction
MUX
TRANSMIT SHIFT REGISTER
RECEIVE PACK REGISTER
CONTROL AND STATUS
Status
TRANSMIT FIFO 2-DEEP
RECEIVE FIFO 4-DEEP
Status
DDE
DDE
PERIPHERAL BUS
LOCAL SCB ARBITER INTERFACE
SYSTEM SCB INTERFACE

Figure 15-1: Link Port Block Diagram

External Connections

As shown in the ***Link Port Pin Connections*** figure, a link port has eight data lines (`LP_D0` – `LP_D7`), a clock line (`LP_CLK`), and an acknowledge line (`LP_ACK`). A link port can act as either a transmitter or a receiver but not both at the same time.

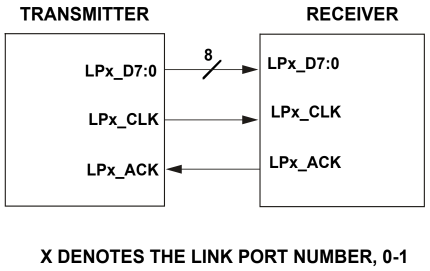


Figure 15-2: Link Port Pin Connections

Use external pull-downs for the `LP_CLK` and `LP_ACK` pins so that the link port can enable the transmitter and receiver, irrespective of the state of the other.

Internal Blocks

As shown in the block diagram, the link ports have independent modules for transmit and receive. If enabled as transmitter, the link port uses a 2 deep 32-bit FIFO. If enabled as a receiver, the port uses a 4 deep 32-bit FIFO. The core MMR access bus interfaces with these FIFOs. The distributed DMA engines (DDE) use the system cross bar (SCB) interface to access the FIFO. The link port uses the `LP_CTL.TRAN` bit to determine whether the module is enabled for transmit or receive operation.

Architectural Concepts

The following sections describe the architectural concepts of the link port.

- Link Port Protocol
- FIFO Buffers
- Handshake for Link Port Enable Process
- Clocking
- Multi-Processor Connectivity

Link Port Protocol

A link port transmitted word consists of 4 bytes and the communication proceeds as follows.

1. The transmitter asserts the link port clock (`LP_CLK`) with each byte of data. The receiver uses the falling edge of `LP_CLK` driven by the transmitter to latch the byte.
2. When the receiver is ready to accept another word in the receive buffer it asserts the acknowledge signal, `LP_ACK`.
3. The transmitter samples `LP_ACK` driven by the receiver at the beginning of each word transmission. If `LP_ACK` is deasserted, then the transmitter does not transmit the next word.
4. The transmitter leaves `LP_CLK` high and continues to drive the first byte of the next word until `LP_ACK` is asserted.

5. When this assertion occurs, the transmitter drives `LP_CLK` low. The transmission of the next word starts. If the transmit buffer is empty, `LP_CLK` remains low until the buffer refills, regardless of the state of `LP_ACK`.

The `LP_ACK` signal can deassert when it anticipates that the buffer could fill. The receiver reasserts the `LP_ACK` signal as soon as the internal DMA grant signal has occurred or the core reads the receive buffer. Either of these actions frees a buffer location.

NOTE: The `LP_ACK` signal inhibits transmission of the next word and not of the current byte.

The `LP_ACK` signal provides a handshake between the receiver and transmitter in the following configurations.

- When configured as a transmitter, the port drives both the data and the clock while `LP_ACK` is three-stated. In this mode, `LP_CLK` is always synchronous with CDU0_CLKO4.
- When configured as a receiver, the link port drives the acknowledge signal and the data and clock lines are three-stated. In this case, the external `LP_CLK` signal can either be synchronous or asynchronous with CDU0_CLKO4.
- When the link port is disabled, the data, clock, and acknowledge signals are three-stated.

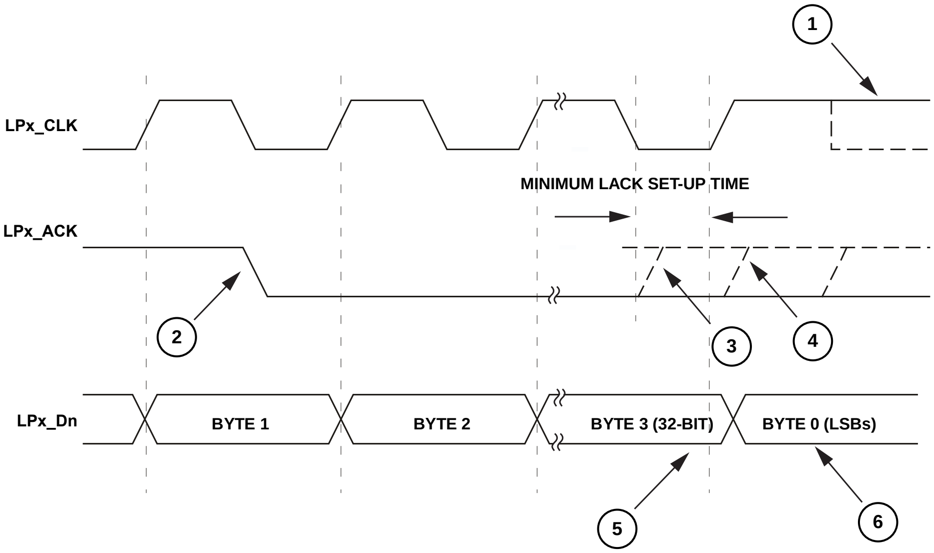


Figure 15-3: Link Port Communication and Handshake Waveform

The following list describes the stages shown in the ***Link Port Communication and Handshake Waveform*** figure.

1. `LP_CLK` stays high at byte 0 when `LP_ACK` is sampled low on the previous `LP_CLK` rising edge. `LP_CLK` high indicates a stall.
2. The `LP_ACK` signal can deassert after byte 0.
3. The `LP_ACK` signal reasserts as soon as the link buffer is not full (depending on Rx FIFO conditions).
4. The transmitter samples `LP_ACK` to determine whether to transmit the next word.

5. The receiver accepts the remaining word even if `LP_ACK` is deasserted. The transmitter does not send the following word.
6. Transmission of data for next word is held until `LP_ACK` is asserted.

The transmitter samples the `LP_ACK` signal. If the signal is high, the transmitter gives out the falling edges of `LP_CLK` for data sampling. The `LP_ACK` signal is first sampled at the rising edge of CDU0_CLKO4. One more CDU0_CLKO4 stage synchronizes the signal further. This synchronized signal is given to the subsequent logic. The `LP_CLK` falling edge is aligned with CDU0_CLKO4 falling edge in a 1:1 clock ratio mode and with the *SCLK* rising edge for the rest of the clock ratios. The following figures explain how the synchronization is maintained between the `LP_ACK` and `LP_CLK` signals.

In the following figure, synchronizing time is guaranteed to be 1.5 CDU0_CLKO4 cycles.

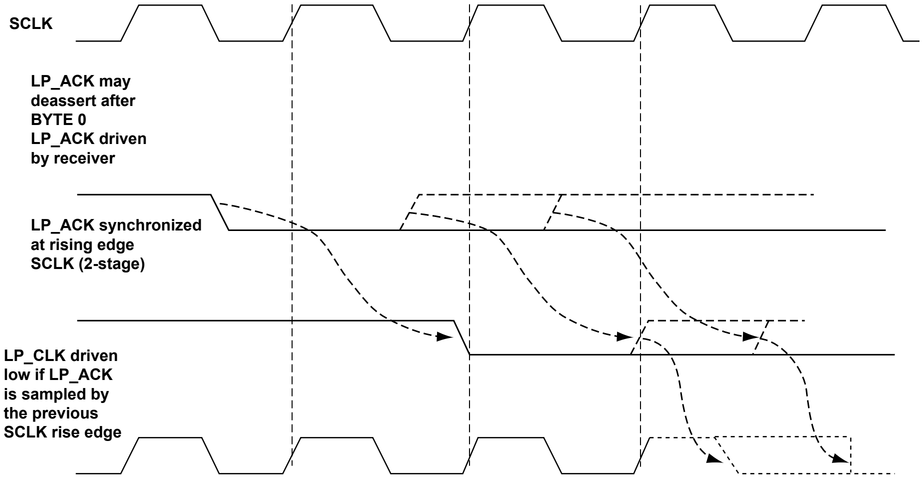

SCLK
LP_ACK may deassert after BYTE 0 LP_ACK driven by receiver
LP_ACK synchronized at rising edge SCLK (2-stage)
LP_CLK driven low if LP_ACK is sampled by the previous SCLK rise edge

Figure 15-4: `LP_ACK` Synchronization for SCLK:`LP_CLK`=1:1

In the following figure, synchronizing time is guaranteed to be 2 CDU0_CLKO4 cycles.

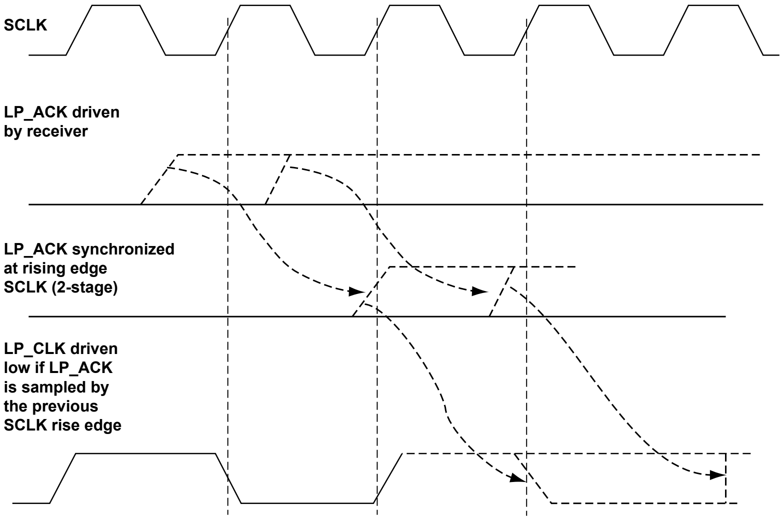

SCLK
LP_ACK driven by receiver
LP_ACK synchronized at rising edge SCLK (2-stage)
LP_CLK driven low if LP_ACK is sampled by the previous SCLK rise edge

Figure 15-5: `LP_ACK` Synchronization for SCLK: `LP_CLK`=1:2, 1:4 and Up

The link port uses the value programmed in the `LP_DIV` register at the transmitter to determine the frequency of the link port clock (`LP_CLK`). However, the signal appearing on the `LP_CLK` pin is also dependent on the status of the `LP_ACK` pin driven by the receiver. The ***Relationship Between Internal Link Port Clock and Link Port Clock at the Pins*** figure shows this relationship.

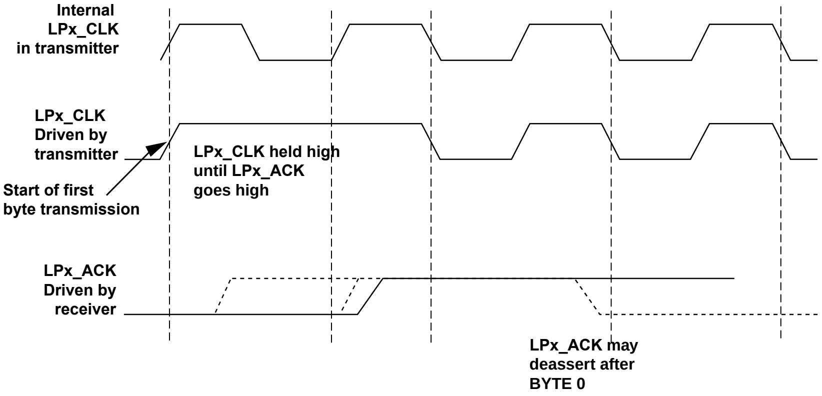

Internal LPx_CLK in transmitter
LPx_CLK Driven by transmitter
Start of first byte transmission
LPx_CLK held high until LPx_ACK goes high
LPx_ACK Driven by receiver
LPx_ACK may deassert after BYTE 0

Figure 15-6: Relationship Between Internal Link Port Clock and Link Port Clock at the Pins

FIFO Buffers

When a link port is configured as transmitter, the link port uses a 2-deep FIFO buffer. A shift register unpacks the single 32-bit word to four 8-bit data bytes. As the FIFO has space for more data, the link port makes a new DMA request. If the FIFO becomes empty, the `LP_CLK` signal is deasserted. The core can access FIFO through the `LP_TX` register.

The core or DMA makes three writes (2-stage FIFO and 1 shift register) to the transmit buffer before it signals a full condition. The link port uses the `LP_STAT.FFST` bit field to reflect the status of the FIFO but not the shift register full or empty condition. However, the program can poll the `LP_STAT.LPBS` bit to discover whether the link port is driving data from the shift register to the pins. The `LP_STAT.LPBS` bit is also set when receiver has held off transmission by driving `LP_ACK` low.

NOTE: When the 2-deep FIFO and the output shift-register overflow, any further write to the link port buffer overwrites the input stage of the FIFO.

NOTE: The core can also read the transmit FIFO through the data `LP_TX` register.

NOTE: If the transmitter is disabled while performing writes to the transmit FIFO, a FIFO full condition is signaled after two writes.

The transmit buffer registers have shadow registers. Using these shadow registers, both stages of the 2-deep FIFO can be read without updating the status registers. The `LP_TXIN_SHDW` register corresponds to the input stage of the FIFO. The `LP_TXOUT_SHDW` register corresponds to the output stage of the FIFO as shown in the ***Transmit FIFO path*** figure.

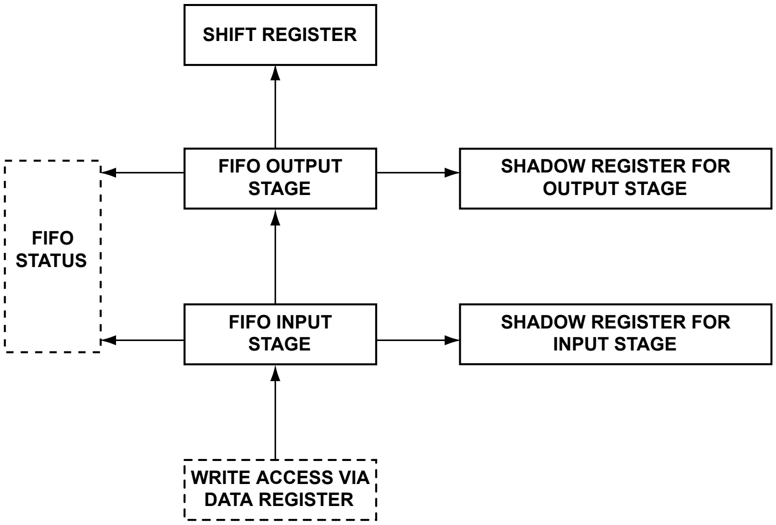

SHIFT REGISTER
FIFO OUTPUT STAGE
SHADOW REGISTER FOR OUTPUT STAGE
FIFO STATUS
FIFO INPUT STAGE
SHADOW REGISTER FOR INPUT STAGE
WRITE ACCESS VIA DATA REGISTER

Figure 15-7: Transmit FIFO path

When a link port is configured as receiver, data transfers to the core or DMA from the full 4-deep receive FIFO. An internal packing register packs data to 32 bits. Four reads can occur from the receive buffer by the core or DMA before it signals an empty condition. The link port uses the `LP_STAT.FFST` bits to reflect the status of the 4-deep read buffer FIFO. The core can access this FIFO through the `LP_RX` register.

NOTE: When receive FIFO overflows (`LP_STAT.ROVF` bit=1), any further data from the transmitter is lost. Only the data retained in the receive FIFO can be retrieved further.

The receiver drives the `LP_ACK` output signal low, after the first byte of data for the next-to-last empty slot (in the 4-deep FIFO) is received. This functionality prevents data loss due to the transmitter starting transmission of the next word before the `LP_ACK` signal reaches the transmitter. (The timing is due to the larger delay in synchronization.) This functionality guarantees that even after allowing for the extra synchronization cycle in the transmitter and receiver, there is no overflow in the receive FIFO. The ***LACK Generation Based on Receive FIFO Status*** figure shows how FIFO slots influence the acknowledge signal generation. The grayed sections show received data. The white sections show empty locations where the decision to pull `LP_ACK` high is taken.

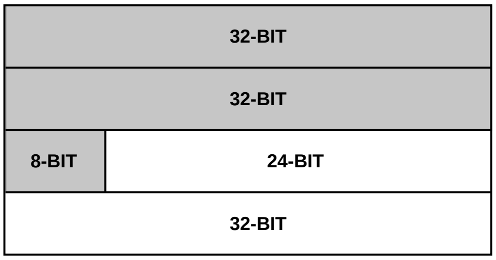


Figure 15-8: LACK Generation Based on Receive FIFO Status

> **NOTE:** The link port uses a 4-deep receive FIFO only under a worst case situation, as mentioned. In all other cases, respond as if the FIFO has only a 3-deep stage. The `LP_ACK` signal is pulled high before the last stage of the FIFO.

The link port has memory-mapped buffers for both receive and transmit operations. A JTAG-based emulator can read the FIFO which can cause unexpected problems in data transfers. This activity can only happen during an emulation event (typically hitting a breakpoint or single-stepping). The emulator issues core reads through JTAG. To work around this issue, see the tools documentation for more information.

Handshake for Link Port Enable Process

In a link port-based system, the transmitter and the receiver can be enabled at different times. Use external pull-downs for the `LP_CLK` and `LP_ACK` signals.

If the receiver is enabled before the transmitter, the external pull-down holds the `LP_CLK` signal of the transmitter low. The receiver is held off. The receiver can wait for a rising edge on the `LP_CLK` signal to assert its receive service request interrupt. This rising edge occurs only when transmitter starts driving the first data on to the bus, after the application enables it.

If the transmitter is enabled before the receiver, the external pull-down holds the `LP_ACK` signal of the receiver low. Transmission is held off. Refer to the ***Enable the Transmitter Before the Receiver*** figure. The transmitter can wait for a rising edge on the `LP_ACK` signal to assert its transmit service request interrupt. This rising edge is asserted as soon as the receiver is enabled after the hardware drives the `LP_ACK` high.

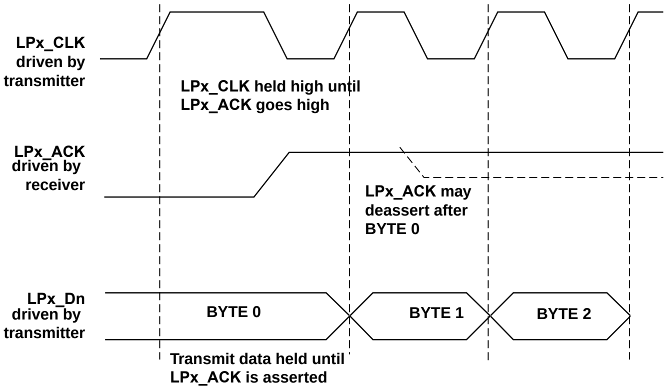

LPx_CLK driven by transmitter
LPx_CLK held high until LPx_ACK goes high
LPx_ACK driven by receiver
LPx_ACK may deassert after BYTE 0
LPx_Dn driven by transmitter
BYTE 0
BYTE 1
BYTE 2
Transmit data held until LPx_ACK is asserted

Figure 15-9: Enable the Transmitter Before the Receiver

NOTE: Service request interrupts or status are asserted only when the link port (receiver or transmitter) is disabled.

Clocking

The link port clock (`LP_CLK`) is derived from the internal system clock (CDU0_CLKO4). The link port clock to system clock ratio can be configured in the `LP_DIV` register. This value applies to the transmitter only. The receiver can operate at any asynchronous frequency up to the maximum frequency, independent of the ratio programmed. The following formula describes the relationship between the frequency of the link port clock, the CDU0_CLKO4 frequency, and the `LP_DIV` value.

- $f_{LP\_CLK} = f_{SCLK} <$ or $= f_{LP\_CLK\text{-}MAX}$ if DIV = 0
- $f_{LP\_CLK} = f_{SCLK}/(2 \times \text{DIV})$ if DIV > 0

Where: f_{LP_CLK} = link clock frequency, $f_{LP\_CLK\text{-}MAX}$ = link clock maximum frequency, and f_{SCLK} = system clock frequency.

While programming the `LP_DIV` register to select the clock ratio, ensure that the `LP_CLK` frequency does not exceed the maximum frequency supported for the device. For example, if the CDU0_CLKO4 frequency is 125 MHz and the limit for `LP_CLK` operation is 83 MHz, `LP_DIV` must be greater than or equal to 1. The resulting `LP_CLK` frequency is less than or equal to 83 MHz. For supported frequencies, see the product-specific data sheet.

NOTE: The link ports on this processor have a flexible system clock assignment where their CDU0_CLKO4 is derived from the CLKO8 output of the CDU. For information on clock programming, see CDU Programming Model.

Multi-Processor Connectivity

Link ports can operate independently, allowing glueless connection with external processors. Link ports have dedicated DMA channels, allowing independent data transfers. The following group of figures shows some examples of different bus connection topology that can be used in multi-processor system design. The inter-connection methods are not limited to these examples.

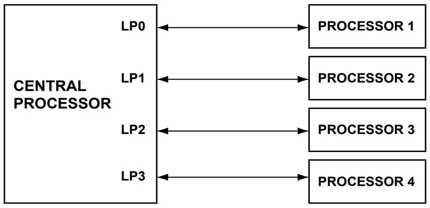

CENTRAL PROCESSOR
LP0
LP1
LP2
LP3
PROCESSOR 1
PROCESSOR 2
PROCESSOR 3
PROCESSOR 4

Figure 15-10: Central Processor-Based Model

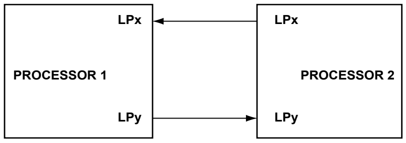

PROCESSOR 1
LPx
LPy
PROCESSOR 2
LPx
LPy

Figure 15-11: Link Port Full-Duplex Transfer Model

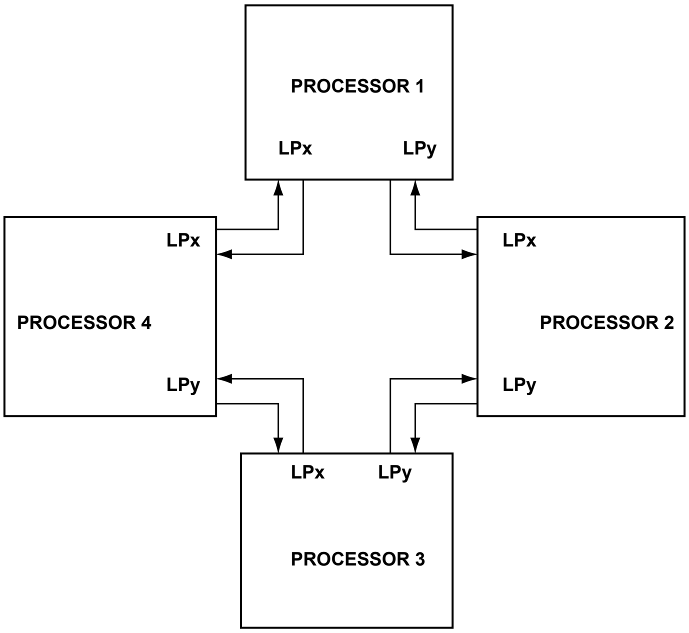

PROCESSOR 1
LPx
LPy
PROCESSOR 4
LPx
LPy
PROCESSOR 2
LPx
LPy
LPx
LPy
PROCESSOR 3

Figure 15-12: Link Port Ring Model

The link port protocol does not include built-in support for multiple masters. However, there can be situations where multiple devices try to become the bus master at the same time. Multi-master conflicts can be resolved using token passing. In token passing, the token is a software flag that passes between processors.

At reset, the token is set to reside in the link port of one device, making it the master and the transmitter. When a receiver (slave) wants to become the master, it can assert its `LP_ACK` signal to get the attention of the master. The master knows, through the software protocol, whether to respond with actual data or whether the token is requested. If the master wishes to give up the token, it can send back a user-defined token release word and thereafter clear its token flag.

Simultaneously, the slave sets its token and can thereafter transmit. The token release word can be any user-defined value. Because the transmitter and receiver expect a code word, this word does not need to be exclusive of normal data transmission. If the master wishes to give up the token, it can send back a user-defined token release word and thereafter clear its token flag. Simultaneously, the slave examines the data sent back and if it is the token release word, the slave sets its token, and can thereafter transmit.

The link port protocol includes handshake mechanism to inform the other end of transfer (transmit or receive) of an enable instance. However, it does not support handshakes to inform a disable instance, while a chunk of data transfers. The application must assume the disabled state of the other end, and take appropriate action.

For example, in a multi-processing environment, a receiver did not read its full FIFO for an extended time due to internal bus arbitrations. The transmitter can require software or a peripheral timer-based timeout to inform the application that the `LP_ACK` signal is low for an extended time period.

LP Operating Modes

The link port does not have particular modes of operation, as the peripheral is based on a simple protocol. The following sections explain the data transfer modes, using the core and using DMA.

- Core Data Transfers
- DMA Data Transfers

LP Data Transfer Modes

This section describes link port DMA and core data transfers.

Core Data Transfers

If DMA is disabled for a link port buffer, the processor core can write or read internal FIFO buffers as a memory-mapped register through the MMR access bus. In order to avoid FIFO overflow or underflow, the core can access the FIFO registers in one of the two following ways.

1. Access link port registers using an interrupt service routine (ISR) mapped to the data request interrupt of the link port. The interrupt request remains high only if the FIFO is accessible (if the FIFO is not full in transmit mode and not empty in receive mode).
2. Poll the FIFO status bits of the `LP_STAT` register. Write to the transmit FIFO if not full or read from the receive FIFO if not empty.

DMA Data Transfers

Dedicated DMA channels are available for each link port. DMA-related activity is explained in the following steps.

1. Data Receive – Once the DMA channel and link port module are configured and enabled, the external device begins writing data to the FIFO through the data pins of the link port. The FIFO detects this activity and in turn sends a DMA request. After the request is granted, the DMA transfer progresses until the FIFO is empty.
2. Data Transmit – Once the DMA channel and link port module are configured and enabled, setting the `LP_CTL.EN` bit automatically asserts a DMA request when the transmit FIFO is empty. After the request is granted, DMA fills the FIFO. The external device begins reading data from the FIFO through the data pins of

the link port. The FIFO detects that there is room in the buffer and asserts another DMA request, continuing the process.

LP Event Control

This section describes how the link port uses interrupts and status signals.

Interrupt Signals

Each link port has two dedicated interrupt lines registered with the system event controller—a data request interrupt and a status interrupt. Data request interrupts are asserted based on FIFO conditions for data transfer. Status interrupts are asserted when a service request status or an overflow status is set. The following list explains each of these interrupts.

- *Data Request Interrupt.* Asserted if the FIFO is not full in transmission mode and the FIFO is not empty in reception mode. This functionality serves as a core triggered interrupt in non-DMA mode and as the DMA interrupt request in DMA mode. Generation of this interrupt is based on the `LP_STAT.FFST` (status bit of the link port buffer).
- *Link Port Transmit Service Request Interrupt (LTRQ)*. Allow a disabled link port to generate an interrupt when an external access is attempted. When a link port is configured as transmitter, the transmit service request interrupt is enabled by setting the `LP_CTL.TRQMSK` bit. When set, an external receiver can indicate to the disabled transmitter that it must receive data through the connected link port. The receiver does so by driving a high level on the `LP_ACK` line. When the `LP_ACK` of the disabled transmitter link port is detected high, a `LP_STAT.LTRQ` interrupt is generated. The transmitter can enable itself for data transfer with the receiver. The link port needs a pull-down on `LP_ACK` for this feature to function properly.
- *Link Port Receive Service Request Interrupt (LRRQ)*. When a link port is configured as receiver, this interrupt is enabled by setting the `LP_CTL.RRQMSK` bit. When set, an external transmitter can indicate to the disabled receiver that it must receive data through the connected link port. The transmitter does so by driving out the first data. When the `LP_CLK` of the disabled receiver link port is detected high, a `LP_STAT.LRRQ` interrupt is generated. The receiver can further enable itself for data transfer with the transmitter. The link port needs a pull-down on the `LP_CLK` signal for this feature to function properly.
- *Link Port Receive Overflow Interrupt (LPOVF)*. Generated when the receiver FIFO overflows and is enabled by setting the `LP_CTL.ROVFMSK` bit. This interrupt can happen if the transmitter continues to transmit data even though the receiver has deasserted `LP_ACK` signal causing the receive FIFO to overflow.

Enabling Link Port Interrupts

A data request interrupt is fed to the system event controller directly and can be controlled separately from the application.

To mask the interrupt, set the mask bits in `LP_CTL` register corresponding to service interrupts and the overflow interrupt. These interrupts are OR'ed and fed to the SIC as a single `LP_STAT` interrupt. These interrupts are latched and stored in the associated bits of `LP_STAT` register. If an `LP_STAT` interrupt occurs, in the ISR, programs can read the `LP_STAT` register bits to determine the type of interrupt. These bits are write-one-to-clear (W1C); writing one to the bit resets the bit and disables the corresponding interrupt.

Status and Error Signals

This section explains the various status signals in the `LP_STAT` register.

- *Transfer Status signals*. The link port uses the bus status bit (`LP_STAT.LPBS`) to give the status of the bus condition (busy or idle), when the link port is configured as transmitter. The `LP_STAT.LPBS` is high if the link port drives data into the link port pins. Programs can poll this bit after polling the `LP_STAT.FFST` bit to disable the link port safely.

  The link buffer status (`LP_STAT.FFST`) field directly indicates the status of the FIFO (including empty or full conditions) during data transfer. Software can poll this field in the `LP_STAT` register before writing to the FIFO (in case of transmission) or reading from the FIFO (in case of reception). The `LP_STAT.FFST` bit is automatically cleared when the link port is disabled.

- *Transfer Request Status signals*. The link port uses the receive request status (`LP_STAT.LRRQ`) bit to indicate that an external receiver wants to receive data (in case the link port is a disabled transmitter). The link port uses the transmit request status (`LP_STAT.LTRQ`) bit to indicate that an external transmitter wants to send data (in case the link port is a disabled receiver). Software can poll these bits to enable the transmitter or receiver accordingly.

- *Error Status signals*. In receive mode 32-bit data is received in four chunks of 8-bit data. This data is then packed to a single 32-bit data before loading the FIFO. The link buffer error status (`LP_STAT.LPACK`) bit is high during this packing process and goes low after packing.

  The link port overflow status (`LP_STAT.ROVF`) bit is set when the receive FIFO overflows. This event can occur if the transmitter continues to transmit data even though the receiver has deasserted `LP_ACK` causing the receiver FIFO to overflow.

LP Programming Model

The following sections provide information on configuring the operating mode and enabling the link ports.

- Setting Up a DMA Transmit Operation
- Setting Up a DMA Receive Operation
- Setting Up a Core Transmit Operation
- Setting Up a Core Receive Operation

Setting Up a DMA Transmit Operation

This following procedure describes the typical steps for configuring the link ports in DMA transmit mode.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.

2. Install interrupt handlers for DMA and for transfer status (service request interrupt).
3. Configure the link port to transmit by setting the `LP_CTL` bit and enable the transmit request interrupt mask by setting the `LP_CTL.TRQMSK` bit.
4. Program the link port clock divider by writing a value to the `LP_DIV` register.
5. If using DMA stop mode or auto buffer mode, program the appropriate DMA registers.

   ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Stop/Auto, `DMA_CFG.PSIZE` =1, `DMA_CFG.MSIZE` =4, interrupt generation and memory read).
6. Wait for the link port receiver (connected externally) to be enabled. The application can wait for the transmit service request interrupt to assert.
7. Clear the transmit service request interrupt status by writing 1 to the `LP_STAT.LTRQ` bit.
8. Enable DMA by setting the `DMA_CFG.EN` bit.
9. Enable the link port by setting the `LP_CTL.EN` bit.
10. Wait for DMA to assert a transfer completion interrupt.
11. Clear the DMA interrupt source by writing 1 to the `DMA_STAT.IRQDONE` bit.

Setting Up a DMA Receive Operation

This section describes the typical steps for using the link ports in DMA receive mode.

1. Enable the link port pins in GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Install interrupt handlers for DMA and for transfer status (service request interrupt).
3. Configure the link port for reception (clear the `LP_CTL.TRAN` bit) and enable the receive request interrupt mask by setting the `LP_CTL.RRQMSK` bit.
4. If using DMA stop mode or auto buffer mode, program the DMA registers.

   ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Stop/Auto, `DMA_CFG.PSIZE=1`, `DMA_CFG.MSIZE=4`, interrupt generation and memory write).
5. If using DMA array mode or list mode, create DMA configuration data structures filled with components.

   ADDITIONAL INFORMATION: An example configuration is: `DMA_ADDRSTART`, `DMA_XCNT`, `DMA_XMOD`, and `DMA_CFG` registers (Array/List, `DMA_CFG.PSIZE` =1, `DMA_CFG.MSIZE` =4, interrupt generation, memory write and fetch =4/5) and `DMA_DSCPTR_NXT` register (if list mode). Further, program DMA configuration register (Array/List, `DMA_CFG.PSIZE` =1, `DMA_CFG.MSIZE` =4, Memory Write and Fetch =4/5) and program the `DMA_DSCPTR_NXT` register (if list mode).

6. Wait for the link port transmitter (connected externally) to be enabled with subsequent data transmission. The application can wait for the receive service request interrupt to assert.
7. Clear the receive service request interrupt status by writing 1 to the `LP_STAT.LRRQ` bit.
8. Enable DMA by setting the `DMA_CFG.EN` bit.
9. Enable the link port by setting the `LP_CTL.EN` bit.
10. Wait for DMA to assert the transfer complete interrupt.
11. Clear the DMA interrupt source by writing 1 to the `DMA_STAT.IRQDONE` bit of the DMA status register.

Setting Up a Core Transmit Operation

This section describes the typical steps for using the link ports in processor core based transmission.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Install interrupt handlers for data transfer and for transfer status (service request interrupt). The interrupt handlers for data transfer are the same source or ID as the DMA interrupt line in the SEC.
3. Configure the link port for transmission by setting the `LP_CTL.TRAN` bit) and enable the transmit request interrupt mask by setting the `LP_CTL.TRQMSK` bit).
4. Program the link port clock divider by writing a value in to the `LP_DIV` register.
5. Wait for the link port receiver (connected externally) to be enabled. The application can wait for a transmit service request interrupt to assert.
6. Clear the transmit service request interrupt status by writing 1 to the `LP_STAT.LTRQ` bit.
7. Enable the link port by setting the `LP_CTL.EN` bit.
8. The data request interrupt is asserted whenever there is free space in the FIFO. The application can write to the `LP_TX` register based on the FIFO conditions (half or empty) reflected in the `LP_STAT.FFST` bit field.

Setting Up a Core Receive Operation

This section describes the typical steps for using the link ports in processor core-based reception.

1. Enable the link port pins in the GPIO port mux using the appropriate `PORT_FER` and `PORT_MUX` registers.
2. Install interrupt handlers for data transfer and for transfer status (service request interrupt). The interrupt handlers for data transfer are the same source or ID as the DMA interrupt line in the SEC).
3. Configure link port for reception (clear `LP_CTL.TRAN` bit). Enable the receive request interrupt mask bit (set `LP_CTL.RRQMSK`).
4. Wait for the link port transmit (connected externally) to be enabled with subsequent transmission of data. The application can wait for receive service request interrupt to be asserted.

5. Clear the receive service request interrupt status by writing 1 to the `LP_STAT.LRRQ` bit.
6. Enable the link port by setting the `LP_CTL.EN` bit.
7. The data request interrupt is asserted whenever there is free space in the FIFO. The application can read from the `LP_RX` register based on the FIFO conditions (1 or 2 or 3 data available) which is reflected in the `LP_STAT.FFST` bit field.

ADSP-SC58x LP Register Descriptions

Link Port (LP) contains the following registers.

Table 15-6: ADSP-SC58x LP Register List

Name	Description
`LP_CTL`	Control Register
`LP_DIV`	Clock Divider Value Register
`LP_RX`	Receive Buffer Register
`LP_STAT`	Status Register
`LP_TX`	Transmit Buffer Register
`LP_TXIN_SHDW`	Shadow Input Transmit Buffer Register
`LP_TXOUT_SHDW`	Shadow Output Transmit Buffer Register

Control Register

The `LP_CTL` register provides LP interrupt masking, selection of transfer direction, and link port enable.

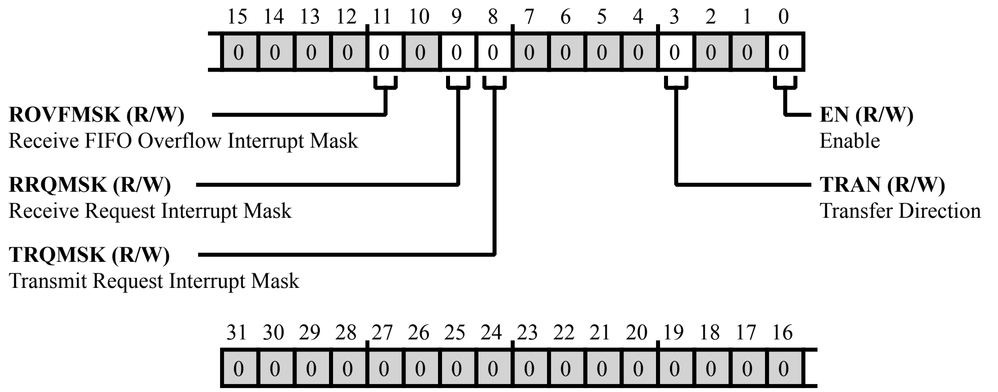

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROVFMSK (R/W)
Receive FIFO Overflow Interrupt Mask
RRQMSK (R/W)
Receive Request Interrupt Mask
TRQMSK (R/W)
Transmit Request Interrupt Mask
EN (R/W)
Enable
TRAN (R/W)
Transfer Direction
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-13: LP_CTL Register Diagram

Table 15-7: LP_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ROVFMSK	Receive FIFO Overflow Interrupt Mask. Receive FIFO Overflow Interrupt Mask	
		0	Mask Disable Receive FIFO Overflow Interrupt
		1	Unmask Enable Receive FIFO Overflow Interrupt
9 (R/W)	RRQMSK	Receive Request Interrupt Mask. Link Port Receive Request Mask	
		0	Mask Disable Receive Request interrupt.
		1	Unmask Enable Receive Request interrupt.
8 (R/W)	TRQMSK	Transmit Request Interrupt Mask. Link Port Transmit Request Mask	
		0	Mask Disable Transmit Request interrupt.
		1	Unmask Enable Transmit Request interrupt.
3 (R/W)	TRAN	Transfer Direction. The `LP_CTL.TRAN` bit selects the transfer direction as transmit (if set) or receive (if cleared) for link buffer.	
		0	Receive Direction transfer is receive
		1	Transmit Direction transfer is transmit

Table 15-7: LP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The `LP_CTL.EN` enables or disables the link port. When the processor disables the port (`LP_CTL.EN` transitions from high to low), the processor clears the corresponding `LP_STAT` bits.	
		0	Disable Disable linkport
		1	Enable linkport Enable linkport

Clock Divider Value Register

The `LP_DIV` register selects the divisor for ratio between the internal LP clock (LCLK) and system clock (CDU0_CLKO8). This programming is applicable only for the transmitter. The receiver can operate at any asynchronous frequency up to the maximum frequency independent of the ratio programmed.

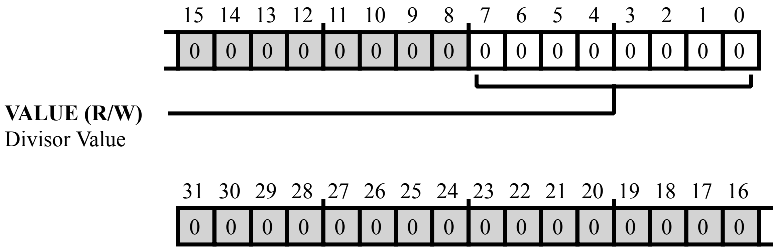

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Divisor Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-14: LP_DIV Register Diagram

Table 15-8: LP_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Divisor Value. The `LP_DIV.VALUE` bits select the clock divider (relating the LP' internally generated clock (LCLK) to the system clock (CDU0_CLKO8). The `LP_DIV.VALUE` should be programmed prior to LP enable. For `LP_DIV.VALUE` = 0, LCLK = CDU0_CLKO8 For `LP_DIV.VALUE` = xxxxxxxx, LCLK = CDU0_CLKO8 / (2 x DIV)

Receive Buffer Register

The `LP_RX` register buffers the receive data flow through the LP. The receive buffer is a four-location deep FIFO. In the receive buffer, data is transferred to the core or DMA from the receive FIFO where an internal register does the packing. This packing register is not software accessible. For more information on LP buffer features and operations, see the LP functional description.

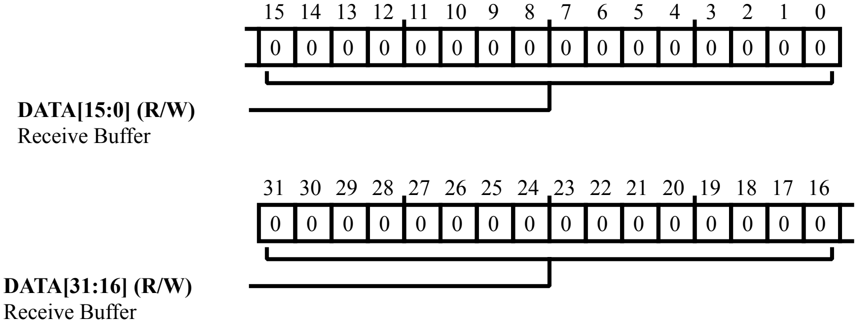

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
Receive Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
Receive Buffer

Figure 15-15: LP_RX Register Diagram

Table 15-9: LP_RX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Receive Buffer.

Status Register

The `LP_STAT` register provides status information on link port interrupts, FIFO, buses, and receive/transmit requests.

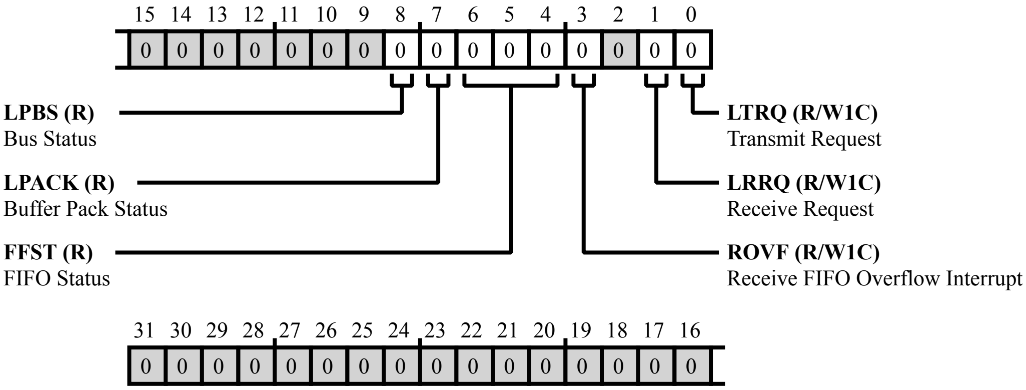

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LPBS (R)
Bus Status
LPACK (R)
Buffer Pack Status
FFST (R)
FIFO Status
LTRQ (R/W1C)
Transmit Request
LRRQ (R/W1C)
Receive Request
ROVF (R/W1C)
Receive FIFO Overflow Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-16: LP_STAT Register Diagram

Table 15-10: LP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	LPBS	Bus Status. The `LP_STAT.LPBS` bit indicates the LPDAT bus status. `LP_STAT.LPBS` is kept high if data is being driven by the link port into the `LP_D[n]` pins.	
		0	Bus is Idle Link Port Bus is idle
		1	Bus Busy Link Port Bus is busy
7 (R/NW)	LPACK	Buffer Pack Status. The `LP_STAT.LPACK` bit indicates packing status. In receive mode, 32-bit data is received in 4 blocks of 8-bit data. Then, the data is packed to get a single 32-bit data before loading the FIFO. The `LP_STAT.LPACK` bit is high during this packing process and goes low after packing. In transmit mode, 32-bit data in the FIFO is unpacked to 4 blocks of 8-bit data before sending. The `LP_STAT.LPACK` is high during unpacking.	
		0	Packing Complete Packing done
		1	Packing Incomplete Packing is in progress
6:4 (R/NW)	FFST	FIFO Status. The `LP_STAT.FFST` bits indicate the FIFO status. These bits are cleared when the LP is disabled.	
		0	TX - Empty; RX - Empty Link buffer (TX OR RX) empty

Table 15-10: LP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	TX - Reserved ; RX - Has 1 data word RX has 1 word of data. TX reserved
		2	TX - Reserved; RX - Has 2 data words RX has 2 word of data. TX reserved.
		3	TX - Reserved; RX - Has 3 data words RX has 3 word of data. TX reserved.
		4	TX - One Word; RX -Has 4 data words RX has 4 word of data. TX 1 word of data.
		5	Reserved
		6	TX - FIFO Full; RX - Reserved RX reserved. TX reserved.
		7	Reserved
3 (R/W1C)	ROVF	Receive FIFO Overflow Interrupt. This interrupt is generated when the receiver FIFO overflows. This overflow may happen if the transmitter continues to transmit data even though the receiver has de-asserted the `LP_ACK` pin.	
1 (R/W1C)	LRRQ	Receive Request. The LP generates this interrupt when the `LP_CLK` pin of a disabled link port (the receiver) is forced high by another link port (the transmitter).	
0 (R/W1C)	LTRQ	Transmit Request. The LP generates this interrupt when the `LP_ACK` pin of a disabled link port (the transmitter) is forced high by another link port (the receiver).	

Transmit Buffer Register

The LP_TX register buffers the transmit data flow through the LP. The transmit buffer is two words deep. In the transmit buffer, the input stage of the FIFO is used to accept core data or DMA data from internal memory, and the data is transferred to the link port interface from the output stage of the FIFO. The output stage performs the unpacking in the transmit buffer. The least significant byte is transmitted first. As each word is unpacked and transmitted, the next location in FIFO becomes available and a new DMA request is made if DMA is enabled. If the register becomes empty, the LP asserts the `LP_CLK` signal. For more information on LP buffer features and operations, see the LP functional description.

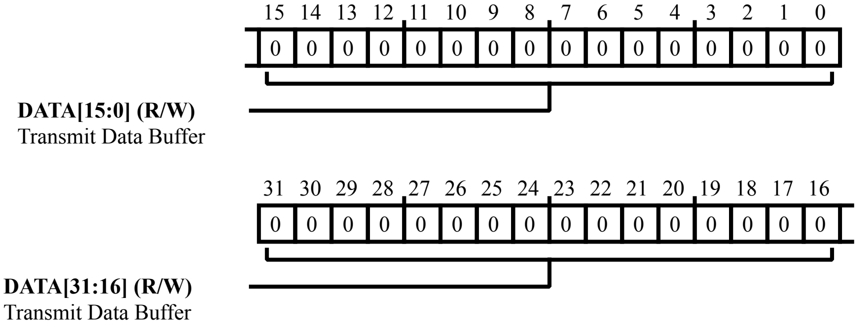

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
Transmit Data Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
Transmit Data Buffer

Figure 15-17: LP_TX Register Diagram

Table 15-11: LP_TX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit Data Buffer.

Shadow Input Transmit Buffer Register

The `LP_TXIN_SHDW` register contains the same data as the input stage of the transmit buffer. Read of this shadow transmit buffer does not update the `LP_STAT` register.

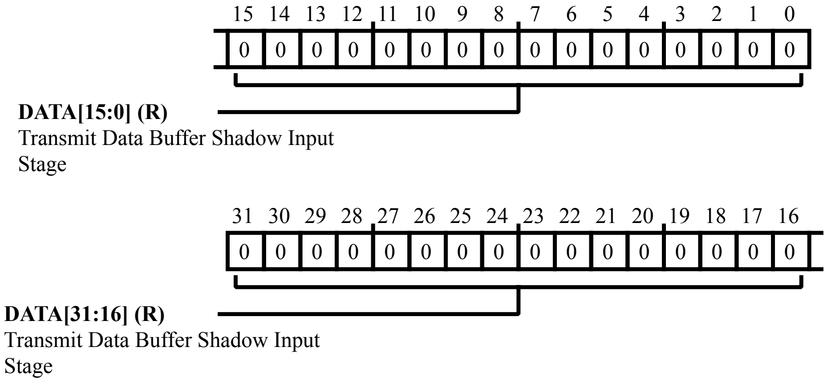

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R)
Transmit Data Buffer Shadow Input Stage
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R)
Transmit Data Buffer Shadow Input Stage

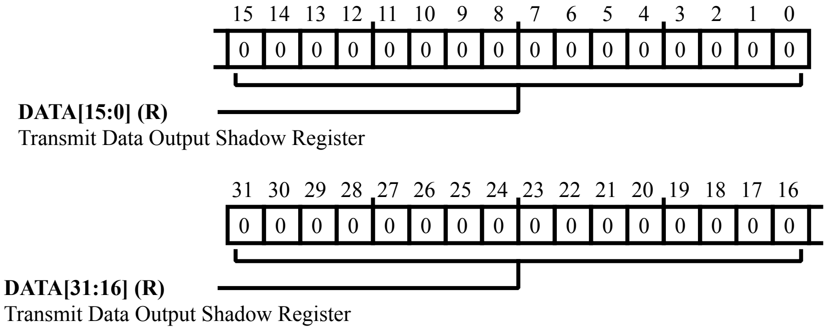
Figure 15-18: LP_TXIN_SHDW Register Diagram

Table 15-12: LP_TXIN_SHDW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Transmit Data Buffer Shadow Input Stage.

Shadow Output Transmit Buffer Register

The `LP_TXOUT_SHDW` register contains the same data as the output stage of the transmit buffer. Read of this shadow transmit buffer does not update the `LP_STAT` register.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R)
Transmit Data Output Shadow Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R)
Transmit Data Output Shadow Register

Figure 15-19: LP_TXOUT_SHDW Register Diagram

Table 15-13: LP_TXOUT_SHDW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Transmit Data Output Shadow Register.

16 Serial Peripheral Interface (SPI)

The serial peripheral interface is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

With a range of configurable options, the SPI ports provide a glueless hardware interface with other SPI-compatible devices in master mode, slave mode, and multimaster environments. The SPI peripheral includes programmable baud rates, clock phase, and clock polarity. The peripheral can operate in a multimaster environment by interfacing with several other devices, acting as either a master device or a slave device. In a multimaster environment, the SPI peripheral uses open-drain outputs to avoid data bus contention. The flow control features enable slow slave devices to interface with fast master devices by providing an SPI ready pin which flexibly controls the transfers.

NOTE: All SPI peripherals (SPI0, SPI1 and SPI2) on the ADSP-SC58x/ADSP-2158x processors operate on the SCLK1_0 domain. The term SCLK in this chapter is a generic reference to SCLK1_0. For more details on SCLK1_0 programming refer the *Clock Generation Unit (CGU)* chapter.

SPI Features

The SPI module supports the following features:

- Full-duplex, synchronous serial interface
- 8, 16-bit and 32-bit word sizes
- Programmable baud rate, clock phase, and polarity
- Programmable inter-frame latency
- Flow control
- Support for Fast and DIOM modes
- Quad and memory-mapped modes are supported by SPI2 only
- Independent receive and transmit DMA channels

- Burst transfer mode for non-DMA write accesses

SPI Functional Description

This section provides information on the function of the SPI module.

Shift register functionality

The SPI is essentially a shift register that serially transmits and receives data bits to or from other SPI devices. During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line synchronizes shifting and sampling of the information on the two serial data lines.

Master slave functionality

During a data transfer, one SPI system acts as the link master which controls the data flow. The other system acts as the slave, which has data shifted into and out of it by the master. Different devices can take turn being masters, and one master can simultaneously shift data into multiple slaves (broadcast mode). However, only one slave can drive its output to write data back to the master at any given time. This rule must be enforced in the broadcast mode. Several slaves can be selected to receive data from the master in this mode. But only one slave can be enabled to send data back to the master.

Enhanced operating modes

SPI supports enhanced modes of operation like fast mode, DIOM, and Quad-SPI, and optional flow control. In fast mode, received data is sampled on the transmit edge instead of the standard receive edge, thus enabling a full-cycle path for the received data. In DIOM, both MOSI and MISO are configured as input or output pins, and 2 bits are shifted in or out on each receive or transmit edge. In Quad-SPI mode, SPI_D3:0 are configured as input or output pins and 4 bits are shifted in or out on each receive or transmit edge. A slower slave can use flow control to stall a faster master device.

Single and multi-master use

The SPI can be used in a single master as well as multi-master environment. The `SPI_MOSI`, `SPI_MISO`, and the `SPI_CLK` signals are all tied together in both configurations. SPI transmission and reception can be enabled simultaneously or individually, depending on `SPI_RXCTL` and `SPI_TXCTL` settings. In broadcast mode, several slaves can be enabled to receive, but only one slave must be in transmit mode and driving the `SPI_MISO` line.

ADSP-SC58x SPI Register List

The Serial Peripheral Interface (SPI) provides a full-duplex, synchronous serial interface, which supports both master/slave modes and multi-master environments. The SPI's baud rate and clock phase/polarities are programmable, and it has integrated DMA channels for both transmit and receive data streams. A set of registers governs SPI operations. For more information on SPI functionality, see the SPI register descriptions.

Table 16-1: ADSP-SC58x SPI Register List

Name	Description
`SPI_CLK`	Clock Rate Register
`SPI_CTL`	Control Register
`SPI_DLY`	Delay Register
`SPI_ILAT`	Masked Interrupt Condition Register
`SPI_ILAT_CLR`	Masked Interrupt Clear Register
`SPI_IMSK`	Interrupt Mask Register
`SPI_IMSK_CLR`	Interrupt Mask Clear Register
`SPI_IMSK_SET`	Interrupt Mask Set Register
`SPI_MMRDH`	Memory Mapped Read Header (Only on SPI2)
`SPI_MMTOP`	SPI Memory Top Address (Only on SPI2)
`SPI_RFIFO`	Receive FIFO Data Register
`SPI_RWC`	Received Word Count Register
`SPI_RWCR`	Received Word Count Reload Register
`SPI_RXCTL`	Receive Control Register
`SPI_SLVSEL`	Slave Select Register
`SPI_STAT`	Status Register
`SPI_TFIFO`	Transmit FIFO Data Register
`SPI_TWC`	Transmitted Word Count Register
`SPI_TWCR`	Transmitted Word Count Reload Register
`SPI_TXCTL`	Transmit Control Register

ADSP-SC58x SPI Interrupt List

Table 16-2: ADSP-SC58x SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
68	SPI2_TXDMA	SPI2 TX DMA Channel	Level	26
69	SPI2_RXDMA	SPI2 RX DMA Channel	Level	27
70	SPI2_STAT	SPI2 Status	Level	
71	SPI2_ERR	SPI2 Error	Level	
106	SPI0_TXDMA	SPI0 TX DMA Channel	Level	22
107	SPI0_RXDMA	SPI0 RX DMA Channel	Level	23

Table 16-2: ADSP-SC58x SPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
108	SPI0_STAT	SPI0 Status	Level	
109	SPI0_ERR	SPI0 Error	Level	
110	SPI1_TXDMA	SPI1 TX DMA Channel	Level	24
111	SPI1_RXDMA	SPI1 RX DMA Channel	Level	25
112	SPI1_STAT	SPI1 Status	Level	
113	SPI1_ERR	SPI1 Error	Level	
188	SPI2_TXDMA_ERR	SPI2 TX DMA Channel Error	Level	
189	SPI2_RXDMA_ERR	SPI2 RX DMA Channel Error	Level	
198	SPI0_TXDMA_ERR	SPI0 TX DMA Channel Error	Level	
199	SPI0_RXDMA_ERR	SPI0 RX DMA Channel Error	Level	
200	SPI1_TXDMA_ERR	SPI1 TX DMA Channel Error	Level	
201	SPI1_RXDMA_ERR	SPI1 RX DMA Channel Error	Level	

ADSP-SC58x SPI Trigger List

Table 16-3: ADSP-SC58x SPI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
40	`SPI0_TXDMA`	SPI0 TX DMA Channel	Edge
41	`SPI0_RXDMA`	SPI0 RX DMA Channel	Edge
42	`SPI1_TXDMA`	SPI1 TX DMA Channel	Edge
43	`SPI1_RXDMA`	SPI1 RX DMA Channel	Edge
44	`SPI2_TXDMA`	SPI2 TX DMA Channel	Edge
45	`SPI2_RXDMA`	SPI2 RX DMA Channel	Edge

Table 16-4: ADSP-SC58x SPI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
33	`SPI0_TXDMA`	SPI0 TX DMA Channel	Pulse
34	`SPI0_RXDMA`	SPI0 RX DMA Channel	Pulse
35	`SPI1_TXDMA`	SPI1 TX DMA Channel	Pulse
36	`SPI1_RXDMA`	SPI1 RX DMA Channel	Pulse
37	`SPI2_TXDMA`	SPI2 TX DMA Channel	Pulse

Table 16-4: ADSP-SC58x SPI Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
38	`SPI2_RXDMA`	SPI2 RX DMA Channel	Pulse

ADSP-SC58x SPI DMA Channel List

Table 16-5: ADSP-SC58x SPI DMA Channel List

DMA ID	DMA Channel Name	Description
DMA22	SPI0_TXDMA	SPI0 TX DMA Channel
DMA23	SPI0_RXDMA	SPI0 RX DMA Channel
DMA24	SPI1_TXDMA	SPI1 TX DMA Channel
DMA25	SPI1_RXDMA	SPI1 RX DMA Channel
DMA26	SPI2_TXDMA	SPI2 TX DMA Channel
DMA27	SPI2_RXDMA	SPI2 RX DMA Channel

SPI Block Diagram

The ***SPI Controller Block Diagram*** illustrates the blocks of the SPI module. The module is comprised of three primary parts:

- SPI core contains the receive and transmit FIFOs and their associated shift registers
- Control blocks contain the synchronizer and logic to control the data flow through the data pipelines
- Register block

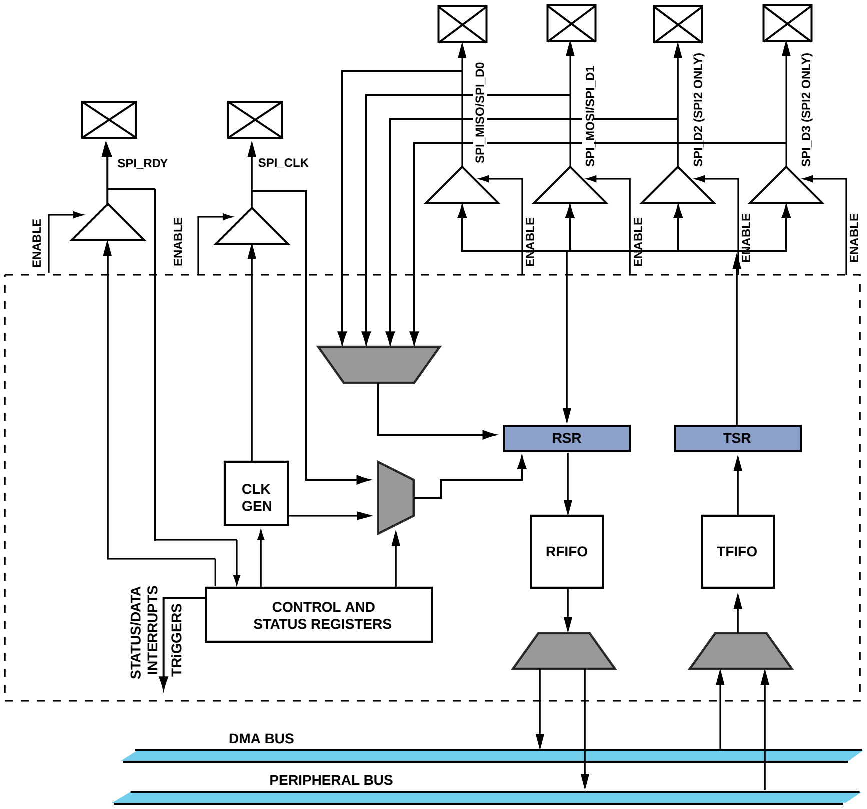

SPI_RDY
SPI_CLK
ENABLE
ENABLE
SPI_MISO/SPI_D0
SPI_MOSI/SPI_D1
SPI_D2 (SPI2 ONLY)
SPI_D3 (SPI2 ONLY)
ENABLE
ENABLE
ENABLE
ENABLE
RSR
TSR
CLK GEN
RFIFO
TFIFO
CONTROL AND STATUS REGISTERS
STATUS/DATA
INTERRUPTS
TRIGGERS
DMA BUS
PERIPHERAL BUS

Figure 16-1: SPI Controller Block Diagram, Quad Mode

Transfer Protocol

The SPI module implements two channels that are independent of each other. The SPI module uses the `SPI_RXCTL` and `SPI_TXCTL` dedicated control registers to control these channels. Except in dual and quad modes, SPI can enable and use both channels simultaneously.

The SPI protocol supports four different combinations of serial clock phase and polarity. These combinations are selected through the `SPI_CTL.CPOL` and `SPI_CTL.CPHA` bits.

The ***SPI Transfer Protocol*** figures demonstrate the two basic transfer formats as defined by the CPHA bit. Two waveforms are shown for `SPI_CLK`; one for `SPI_CTL.CPOL`=0 and the other for `SPI_CTL.CPOL`=1. The diagrams can be interpreted as master or slave timing diagrams since the `SPI_CLK`, `SPI_MISO`, and `SPI_MOSI` pins are directly connected between the master and the slave. The `SPI_MISO` signal is the output from the slave (slave transmission), and the `SPI_MOSI` signal is the output from the master (master transmission). The master generates the `SPI_CLK` signal. The `SPI_SS` signal is the slave device select input to the slave from the master. The diagrams represent an 8-bit transfer (`SPI_CTL.SIZE`=0) with the MSB first (`SPI_CTL.LSBF`=0). Any combination of the `SPI_CTL.SIZE` and `SPI_CTL.LSBF` bits is permissible. For example, a 16-bit transfer with the LSB first is another possible configuration.

The clock polarity and the clock phase could be identical for the master device and the slave device involved in the communication link. The transfer format from the master can be changed between transfers to adjust for various requirements of a slave device.

The SPI module uses the `SPI_CTL.ASSEL` bit to determine when the SPI hardware or software control the `SPI_SEL[n]` line. When `SPI_CTL.ASSEL`=1, the slave select line must be set to the polarity set in the `SPI_CTL.SELST` field between each serial transfer. The actual behavior of `SPI_SEL[n]` also depends on the parameters programmed into the `SPI_DLY` register. The SPI hardware logic automatically controls this functionality. When `SPI_CTL.ASSEL`=0, `SPI_SEL[n]` can either remain active between successive transfers or be inactive. The software must control this activity through manipulation of the `SPI_SLVSEL` register.

The ***SPI Transfer Protocol*** pair of figures illustrates the case when `SPI_CTL.ASSEL` = 1 and the `SPI_SEL[n]` line is inactive between frames. If `ASSEL` = 0, the `SPI_SEL[n]` line can remain active between frames; however, the first bit is only driven when an active transition of `SPI_CLK` occurs.

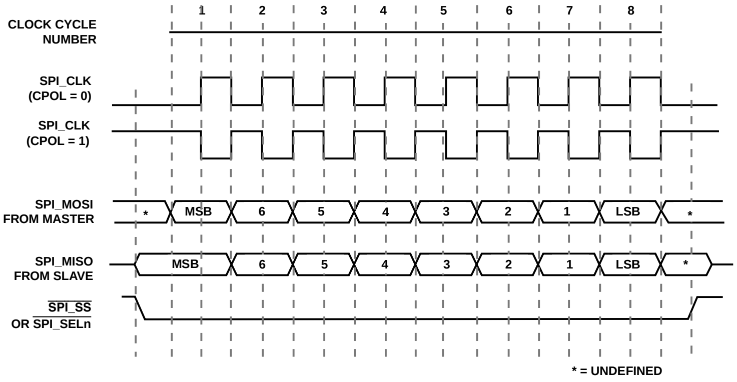

CLOCK CYCLE NUMBER
1
2
3
4
5
6
7
8
SPI_CLK (CPOL = 0)
SPI_CLK (CPOL = 1)
SPI_MOSI FROM MASTER
*
MSB
6
5
4
3
2
1
LSB
*
SPI_MISO FROM SLAVE
MSB
6
5
4
3
2
1
LSB
*
SPI_SS OR SPI_SELn
* = UNDEFINED

Figure 16-2: SPI Transfer Protocol for CPHA=0

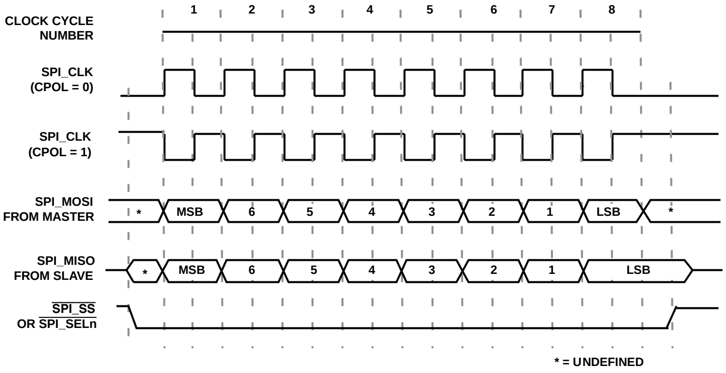

CLOCK CYCLE NUMBER
1
2
3
4
5
6
7
8
SPI_CLK (CPOL = 0)
SPI_CLK (CPOL = 1)
SPI_MOSI FROM MASTER
*
MSB
6
5
4
3
2
1
LSB
*
SPI_MISO FROM SLAVE
*
MSB
6
5
4
3
2
1
LSB
SPI_SS OR SPI_SELn
* = UNDEFINED

Figure 16-3: SPI Transfer Protocol for CPHA=1

Clock Considerations

The `SPI_CLK` signal is a gated clock that is only active during data transfers, for the time of the transferred word. In normal mode, the number of active edges is equal to the number of bits to be transmitted or received. In dual-I/O mode, it is half of the number of bits to be transmitted or received, and in quad-SPI mode it is one-fourth of the number. The clock rate can be as high as the SCLK1_0 rate, and both even and odd dividers from SCLK1_0 are supported. For master devices, the SPI uses the `SPI_CLK` register value to determine the clock rate, whereas this value is ignored for slave devices.

When the SPI controller is a master, `SPI_CLK` is an output signal. Conversely, when the SPI controller is a slave, `SPI_CLK` is an input signal. Slave devices ignore the SPI clock when the slave select input is driven inactive. The SPI uses the `SPI_CLK` signal to shift out and shift in the data driven onto the `SPI_MISO` and `SPI_MOSI` lines. The data is always shifted out on one edge of the clock (the active edge) and sampled on the opposite edge of the clock (the sampling edge). Clock polarity and clock phase relative to data are programmable through the `SPI_CTL` register and define the transfer format.

Controlling Delay Between Frames

The ***SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)*** figure illustrates SPI timing using the `SPI_DLY.LEADX` and `SPI_DLY.LAGX` programming. The SPI uses the `SPI_DLY.LAGX` bits to control the timing between the slave select (`SPI_SS`) signal assertion and the first `SPI_CLK` edge. The SPI uses the `SPI_DLY.LEADX` bits to control the timing between the last `SPI_CLK` edge and deassertion of the `SPI_SS` signal. The lead and lag timing can be extended by a 1 `SPI_CLK` duration to ease timing restrictions on the slave device.

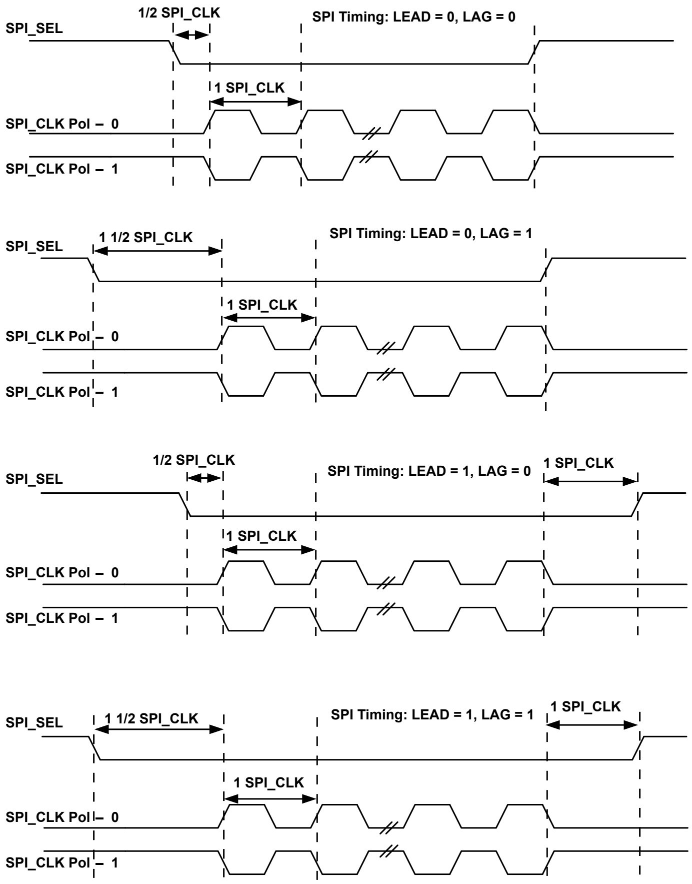

SPI_SEL
1/2 SPI_CLK
SPI Timing: LEAD = 0, LAG = 0
1 SPI_CLK
SPI_CLK Pol – 0
SPI_CLK Pol – 1
SPI_SEL
1 1/2 SPI_CLK
SPI Timing: LEAD = 0, LAG = 1
1 SPI_CLK
SPI_CLK Pol – 0
SPI_CLK Pol – 1
SPI_SEL
1/2 SPI_CLK
SPI Timing: LEAD = 1, LAG = 0
1 SPI_CLK
1 SPI_CLK
SPI_CLK Pol – 0
SPI_CLK Pol – 1
SPI_SEL
1 1/2 SPI_CLK
SPI Timing: LEAD = 1, LAG = 1
1 SPI_CLK
1 SPI_CLK
SPI_CLK Pol – 0
SPI_CLK Pol – 1

Figure 16-4: SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)

The *SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)* figure illustrates SPI timing with STOP programming. The SPI module uses this timing to insert multiples of `SPI_CLK` period delays between transfers. The `SPI_SS` line is deasserted for the duration specified in the `SPI_DLY.STOP` bit field, assuming the `SPI_CTL.SELST` bit is configured for deassertion between transfers.

If the `SPI_DLY.STOP` bit =0, the master operates in a *continuous mode*. This mode causes an immediate start of the second word after the last bit is transferred from the first word. During this mode of operation, the slave select line is continuously asserted.

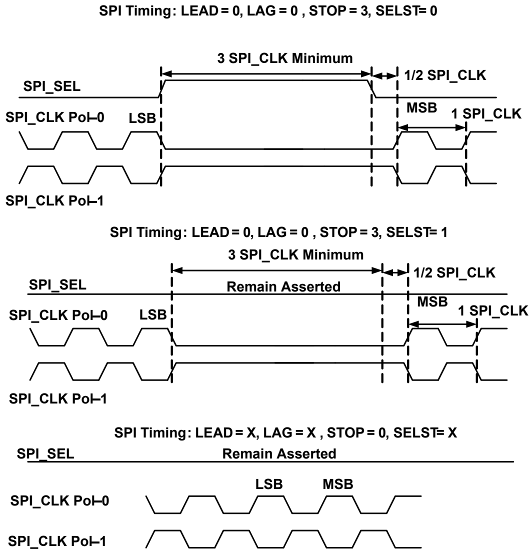

SPI Timing: LEAD = 0, LAG = 0 , STOP = 3, SELST= 0
3 SPI_CLK Minimum
1/2 SPI_CLK
SPI_SEL
SPI_CLK Pol=0
LSB
MSB
1 SPI_CLK
SPI_CLK Pol=1
SPI Timing: LEAD = 0, LAG = 0 , STOP = 3, SELST= 1
3 SPI_CLK Minimum
1/2 SPI_CLK
SPI_SEL
Remain Asserted
SPI_CLK Pol=0
LSB
MSB
1 SPI_CLK
SPI_CLK Pol=1
SPI Timing: LEAD = X, LAG = X , STOP = 0, SELST= X
SPI_SEL
Remain Asserted
LSB
MSB
SPI_CLK Pol=0
SPI_CLK Pol=1

Figure 16-5: SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)

When the `SPI_DLY.STOP` bit =0 and initial conditions for a transfer are not met, the interface pauses before the next transfer. During this pause, the SPI uses the `SPI_CTL.SELST` bit to determine the state of the slave select pin. The SPI uses the `SPI_DLY.LEADX` and `SPI_DLY.LAGX` bits to determine the timing between `SPI_CLK` edges and the slave select line.

Flow Control

In master mode, the slave device must drive the `SPI_RDY` pin. The pin acts as an input signal. The slave can deassert the `SPI_RDY` pin to stop the master from initiating any new transfer. If `SPI_RDY` is deasserted in the middle of a transfer, the current transfer continues, and the next transfer will not start unless the slave asserts the `SPI_RDY` signal. Whenever the slave deasserts `SPI_RDY` and stalls the master, the SPI controller goes into a waiting state, and the `SPI_STAT.FCS` bit is set. When the slave asserts `SPI_RDY`, the SPI controller resumes operation, and the `SPI_STAT.FCS` bit is cleared.

In slave mode, the `SPI_RDY` pin acts as an output signal. Flow control can be configured on either the TX channel or the RX channel. The SPI uses the `SPI_CTL.FCCH` bit to control this configuration. If flow control is configured on the TX channel, as the `SPI_TFIFO` status nears the empty condition, the `SPI_RDY` pin is deasserted. If flow control is configured on the RX channel, as the `SPI_RFIFO` status nears the full condition, the `SPI_RDY` pin is deasserted. The SPI uses the `SPI_CTL.FCWM` bits to control the FIFO status at which `SPI_RDY` deassertion takes place. Flow control in slave mode is purely based on the FIFO status and does not depend on the word counters.

The ***SPI Flow Control Timing in Master Mode*** figure illustrates this timing.

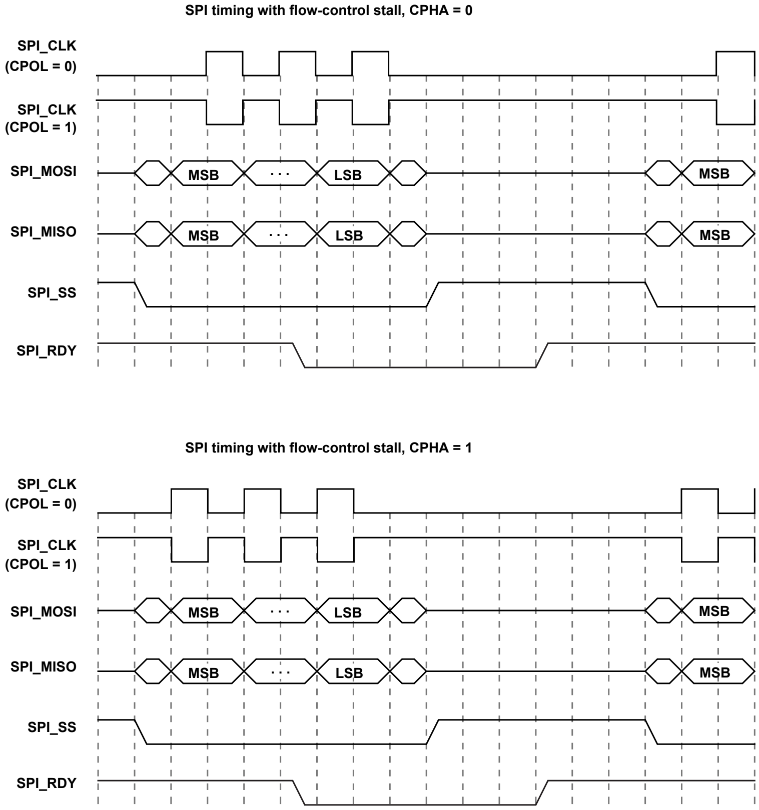

SPI timing with flow-control stall, CPHA = 0
SPI_CLK (CPOL = 0)
SPI_CLK (CPOL = 1)
SPI_MOSI
MSB
· · ·
LSB
MSB
SPI_MISO
MSB
· · ·
LSB
MSB
SPI_SS
SPI_RDY
SPI timing with flow-control stall, CPHA = 1
SPI_CLK (CPOL = 0)
SPI_CLK (CPOL = 1)
SPI_MOSI
MSB
· · ·
LSB
MSB
SPI_MISO
MSB
· · ·
LSB
MSB
SPI_SS
SPI_RDY

Figure 16-6: SPI Flow Control Timing in Master Mode.

Slave Select Operation

If the SPI is in slave mode, `SPI_SS` acts as the slave select input. When SPI is enabled as a master, `SPI_SS` can serve as an error detection input for the SPI in a multi-master environment. The `SPI_CTL.PSSE` bit enables this feature. When `SPI_CTL.PSSE`=1, the `SPI_SS` input is the master mode error input. Otherwise, `SPI_SS` is ignored.

The `SPI_SS` signal is an active-low signal. The master asserts the signal during the transfer. The signal can be deasserted or remain asserted between transfers. When `SPI_SS` is deasserted, `SPI_CLK` and inputs are ignored, and outputs are three-stated.

The slave select bits (`SPI_SLVSEL.SSEL1` – `SPI_SLVSEL.SSEL7`) are used in a multiple-slave SPI environment. For example, if there are eight SPI devices in the system including a processor master, the master processor can support the SPI mode transactions across the other seven devices. This configuration requires only one master processor in this multi-slave environment.

For example, assume that the SPI of the processor is the master. The `SPI_SLVSEL.SSEL1` – `SPI_SLVSEL.SSEL7` bits on the processor can be connected to the slave select pin of each slave device. In this configuration, the slave select bits can be used in three ways. In cases 1 and 2, the processor is the master and the seven microcontrollers or peripherals with SPI interfaces are slaves. The processor can do one of the following:

1. Transmit to all seven SPI devices at the same time in a broadcast mode. Here, all slave select bits are set.
2. Receive and transmit from one SPI device by enabling only one slave SPI device at a time.
3. If all the slaves are also processors, then the requester can receive data from only one processor at a time. (The functionality is enabled by clearing the `SPI_CTL.EMISO` bit in the six other slave processors.) The requestor can transmit broadcast data to all seven at the same time. This MISO enabling feature is available in some other microcontrollers. Therefore, it is possible to use the MISO enabling feature with any other SPI device that includes this functionality.

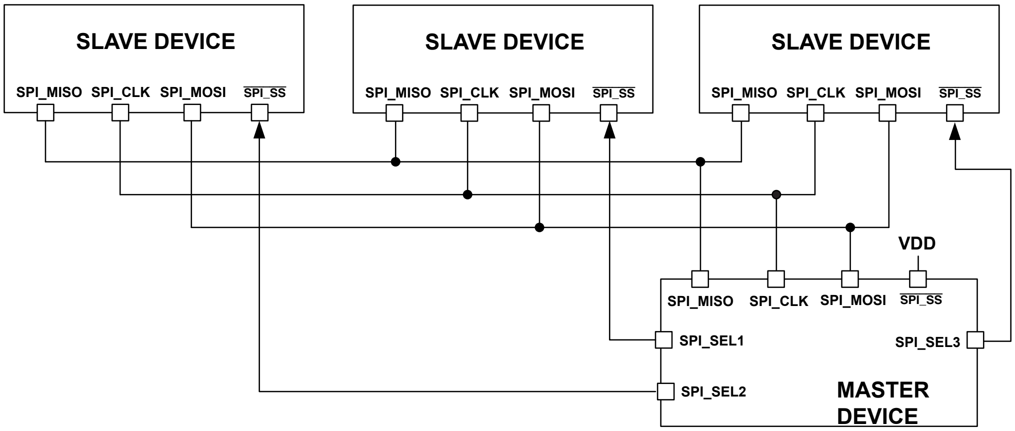


Figure 16-7: Single-Master, Multiple-Slave Configuration

Beginning and Ending a Non-DMA SPI Transfer

The start and finish of a non-DMA SPI transfer depend on the following settings.

1. Whether the device is configured as a master or a slave.
2. The state of the `SPI_CTL.ASSEL` bit, which selects between hardware and software control over `SPI_SLVSEL`.

When `SPI_CTL.CPHA`=0, the enabled slave select outputs are driven active. However, the `SPI_CLK` signal remains inactive for the first half of the first cycle of `SPI_CLK`. For a slave with `SPI_CTL.CPHA`=0, the transfer starts as soon as the `SPI_SS` input goes low.

When `SPI_CTL.CPHA`=1, a transfer starts with the first active edge of `SPI_CLK` for both slave and master devices. For a master device, a transfer is complete after it sends the last data and simultaneously receives the last data bit. A transfer for a slave device ends after the last sampling edge of `SPI_CLK`. If `SPI_CTL.ASSEL`=0, the hardware maintains responsibility for toggling `SPI_SS` between frames. If `SPI_CTL.ASSEL`=1, software controls the `SPI_SS` line and can keep it active between frames.

The `SPI_STAT.RFE` bit defines when the receive buffer can be read, indicating that `SPI_RFIFO` is not empty. The `SPI_STAT.TFF` bit defines when the transmit buffer can be written, indicating that the `SPI_TFIFO` is not full. The end of a single word transfer occurs when the `SPI_STAT.RFE` bit is cleared. The status indicates that a new word has been received and written into the receive FIFO. The `SPI_STAT.RFE` bit remains cleared as long as the receive FIFO has valid data.

To maintain software compatibility with other SPI devices, the `SPI_STAT.SPIF` bit is also available for polling. This bit can have a slightly different behavior from other commercially available devices.

In master mode with the `SPI_CTL.ASSEL` bit cleared, software manually asserts the required slave select signal before starting the transaction. After all data transfers, software typically releases the slave select line.

When the receive or transmit word counters are enabled in the `SPI_TXCTL` or `SPI_RXCTL` registers, the SPI generates a finish interrupt at the end of the transfer. It signals the end of all transfers related to that transaction.

Transmit Operation in Non-DMA Mode

The transmit operation in non-DMA mode is enabled through the `SPI_TXCTL.TEN` bit. It can be enabled independently from the receive operation, and the transmit channel can become the initiating channel based on the `SPI_TXCTL.TTI` bit setting.

Transmit underrun is not possible in this mode, as no new transfer initiates unless the transmit FIFO is empty (in the case that `SPI_TXCTL.TTI` =1). A receive overflow is detected when data from a new frame transfer replaces older data in a full receive FIFO. This event can occur if `SPI_TXCTL.TTI` =1 and the receive channel is enabled in a non-initiating capacity.

An SPI transmit interrupt request is signaled once the transmit channel has been enabled and the transmit FIFO is not full. The SPI uses the `SPI_TXCTL.TDR` bit setting to control the frequency of the interrupt request.

Receive Operation in Non-DMA Mode

The receive operation in non-DMA mode is enabled through the `SPI_RXCTL.REN` bit. It can be enabled independently from the transmit operation, and the receive channel can become the initiating channel based on the `SPI_RXCTL.RTI` bit setting.

Receive overflow is not possible in this mode, as no new transfer initiates when the receive FIFO is full (in the case of `SPI_RXCTL.RTI` =1). A transmit underrun can occur (`SPI_TXCTL.TDU` bit) when no valid data is in the `SPI_TFIFO` register when a transfer is initiated. This event can occur if `SPI_RXCTL.RTI` =1 and the transmit channel is enabled in a non-initiating capacity.

An SPI receive interrupt request is signaled once the receive channel has been enabled and there is data waiting to be read. The SPI uses the `SPI_RXCTL.RDR` bit setting to control the frequency of the interrupt request.

Dual I/O Mode

In Dual I/O mode, the `SPI_MISO` and `SPI_MOSI` pins are configured to operate in the same direction which doubles bandwidth. The SPI uses the `SPI_CTL.SOSI` bit to determine the order of bits on the pins. When set, the processor sends the first bit on the `SPI_MOSI` pin and the second bit on the `SPI_MISO` pin. If the

`SPI_CTL.SOSI` bit is cleared, the order is reversed. Since dual I/O mode uses both pins to transmit or receive data, only one channel can be enabled, either transmit or receive. Flow control through the `SPI_RDY` pin is supported. Interrupt request generation is unaffected by dual I/O mode. However, the interrupt service interval is reduced, since the individual transfer latency is halved.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

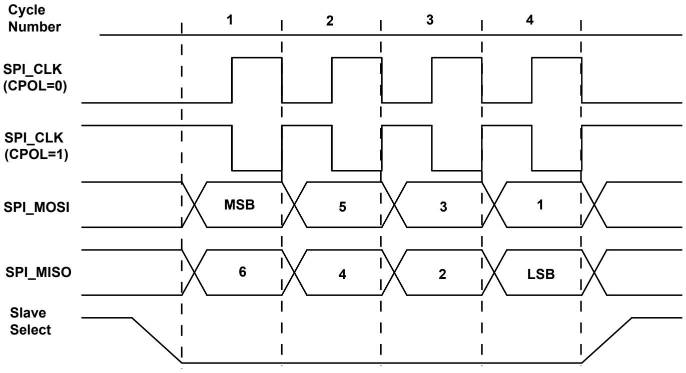

Cycle Number
1
2
3
4
SPI_CLK (CPOL=0)
SPI_CLK (CPOL=1)
SPI_MOSI
MSB
5
3
1
SPI_MISO
6
4
2
LSB
Slave Select

Figure 16-8: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

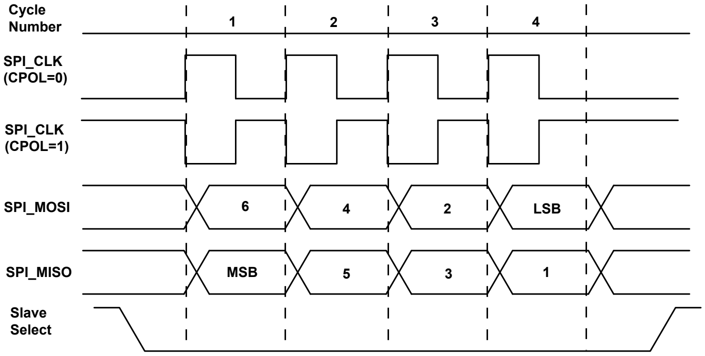

Cycle Number
1
2
3
4
SPI_CLK (CPOL=0)
SPI_CLK (CPOL=1)
SPI_MOSI
6
4
2
LSB
SPI_MISO
MSB
5
3
1
Slave Select

Figure 16-9: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

Quad I/O Mode (SPI2 only)

In quad SPI mode, the `SPI_MISO` and `SPI_MOSI` pins, in tandem with the `SPI_D2` and `SPI_D3` pins, are configured to operate in the same direction. The SPI uses the `SPI_CTL.SOSI` bit to determine the order of bits on the pins. When set, the processor sends:

- The first bit on the `SPI_MOSI` pin
- The second bit on the `SPI_MISO` pin
- The third bit on the `SPI_D2` pin
- The fourth bit on the `SPI_D3` pin

If the `SPI_CTL.SOSI` bit is cleared, the order is reversed. Since quad SPI mode uses all four pins to transmit or receive data, only one channel can be enabled as either transmit or receive. Flow control through the `SPI_RDY` pin is supported. Interrupt generation is unaffected by quad SPI mode.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

While using dual or quad I/O mode for communicating with flash devices, program the `SPI_CTL.CPHA` and the `SPI_CTL.CPOL` bits =1. This programming avoids bus contention during read operations, because the flash device starts driving out the bits immediately after dummy cycles in read header.

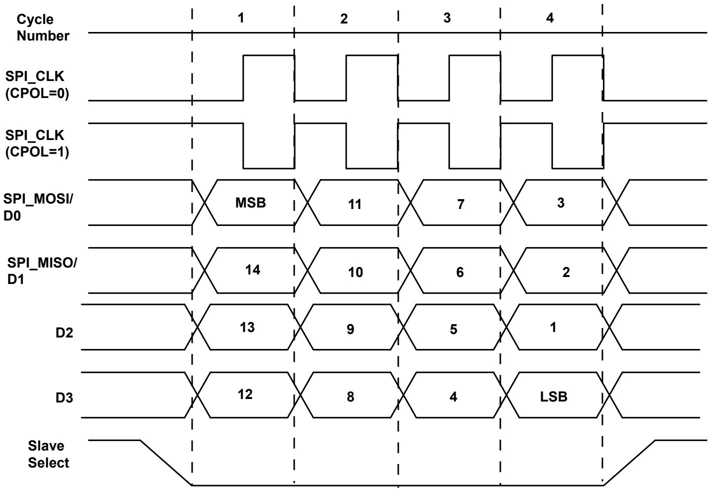

Cycle Number
1
2
3
4
SPI_CLK (CPOL=0)
SPI_CLK (CPOL=1)
SPI_MOSI/ D0
MSB
11
7
3
SPI_MISO/ D1
14
10
6
2
D2
13
9
5
1
D3
12
8
4
LSB
Slave Select

Figure 16-10: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

NOTE: The SPI does support quad SPI 8-bit transfer in slave continuous mode of operation with an SCLK:`SPI_CLK` ratio of less than 1:2. A minimum of 2 SCLK cycles is required between transfers in 8-bit quad SPI slave mode with an SCLK:`SPI_CLK` ratio of less than 1:2.

Fast Mode

Fast mode is similar to the normal mode of operation when transmitting. When receiving, data is sampled at the next transmit edge allowing a full cycle of timing in the receive direction. This mode is valid in master mode operation only. When the SPI operates in fast mode, the slave drives the data for one full cycle.

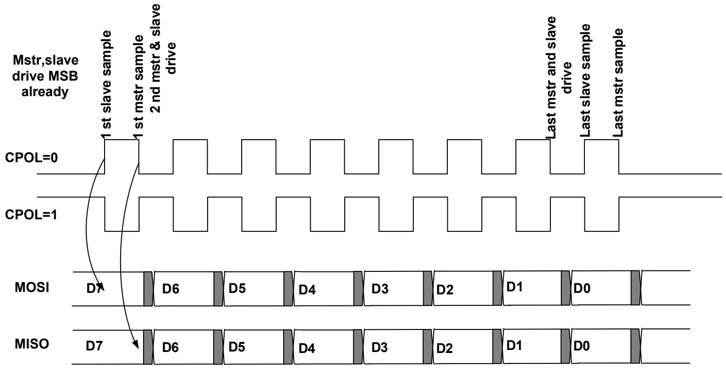
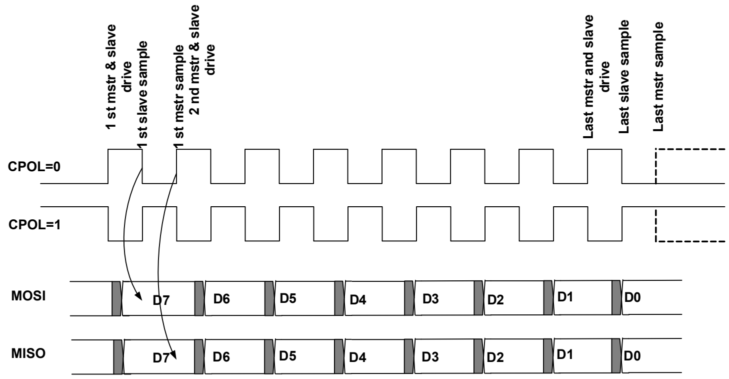

Mstr,slave drive MSB already
1 st slave sample
1 st mstr sample 2 nd mstr & slave drive
Last mstr and slave drive
Last slave sample
Last mstr sample
CPOL=0
CPOL=1
MOSI
D7
D6
D5
D4
D3
D2
D1
D0
MISO
D7
D6
D5
D4
D3
D2
D1
D0

Figure 16-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0



Note : Last Master sample edge is internally generated to latch incoming data

Figure 16-12: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

Memory-Mapped Mode (SPI2 only)

The SPI supports direct memory-mapped read accesses from a SPI memory device, enabled by setting the `SPI_CTL.MMSE` bit. This mode allows for direct execution of instructions from a SPI memory device without the need for a low-level software driver, as hardware handles all overhead tasks (for example, transmission of the read header, pin turnaround timing, and receive data sizing). The SPI features configurable options in the memory-mapped read header register (`SPI_MMRDH`) to provide compatibility with a wide range of SPI memory devices.

In non-memory-mapped mode, the software is responsible for providing the command and required dummy words for the read response, whereas this is all handled by hardware when the SPI is in memory-mapped mode. The memory of the SPI device is accessible directly through reads of the processor address space. The read accesses can be code or data accesses in core mode or when using memory DMA (MDMA). These accesses allow code to execute directly from SPI memory devices (true eXecute-In-Place operations), and the contents can be cached to improve performance. It is not necessary to access the SPI data buffer registers nor poll status bits; however, the hardware does not support peripheral DMA accesses nor write operations to the SPI memory space.

The *Types of Operations* table is a comparison of the permitted operations in the non-memory-mapped and memory-mapped modes supported by the SPI controller.

Table 16-6: Types of Operations

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
Core data write	Yes	No
Core data read	Yes	Yes
Code fetch: Execute-In-Place (XIP)	No	Yes
Read/Write accesses using SPI Peripheral DMA	Yes	No
Read/Write accesses by other peripheral DMA channels	No	No
MDMA read	No	Yes

Table 16-6: Types of Operations (Continued)

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
MDMA write	No	No

Memory-Mapped Description of Operation

Memory-mapped mode is enabled by setting the `SPI_CTL.MMSE` bit. When enabled, the SPI (if ready) accepts the read requests through a dedicated on-chip slave interface. The memory subsystem master drives this dedicated interface through the SCB fabric.

In a typical scenario, the memory subsystem master issues read requests to the fabric, and the fabric routes these requests to the slave port of the SPI peripheral. The master describes the read access using a number of parameters such as starting address, transfer size, and burst type. The SPI responds to this read access request when it is ready for a new transfer. It loads the opcode, a specified number of address bytes, and an optional mode byte into the transmit FIFO. The SPI memory state machine begins when both the transmit and receive channels of the SPI are enabled:

- the transmit transfer initiation bit is set (`SPI_TXCTL.TTI`=1), *and*
- the receive initiation bit is cleared (`SPI_RXCTL.RTI`=0)

The SPI memory read sequence starts with the assertion of `SPI_SEL1`. If the SPI memory state machine is in the reset state, it looks for a command. The SPI hardware then sends the specific 8-bit read command (which can be optionally skipped), followed by the SPI memory read address. After this, a dummy period is inserted, in which a mode byte is optionally sent and the pins are held or three-stated during the dummy clocking period.

NOTE: This read header is transmitted over the SPI standard protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, `SPI_SEL1`) or over the extended SPI protocol pins (`SPI_CLK`, `SPI_MOSI`, `SPI_MISO`, `SPI_D2`, `SPI_D3`, `SPI_SEL1`), based on the `SPI_MMRDH.CMDPINS`, `SPI_MMRDH.ADRPINS`, and `SPI_CTL.MIOM` bit settings. SPI memory devices usually support communication in MSB-first mode. In dual mode, the SPI typically uses `SPI_MISO` as IO1 and `SPI_MOSI` as IO0. In quad mode, the SPI typically uses `SPI_D3` pin as IO3, `SPI_D2` as IO2, `SPI_MISO` as IO1, and `SPI_MOSI` as IO0.

When all I/O data pins are three-stated, the SPI continues clocking the SPI memory device, which drives out the data bits at the addressed location, until all bytes are received. The SPI hardware reads the data as configured by the `SPI_CTL.MIOM` bit setting. Upon reception of the last byte, the SPI typically deasserts `SPI_SEL1` to prepare for the next requested read header.

Application code must ensure that the opcode sent is consistent with multiple I/O programming and that the parameters specified in the memory-mapped read header register are consistent with flash read access timing.

The ***SPI Memory-Mapped Register Operations Flow*** diagram shows how the fields of the `SPI_MMRDH` register determine the read header while initiating transfers in memory-mapped mode.

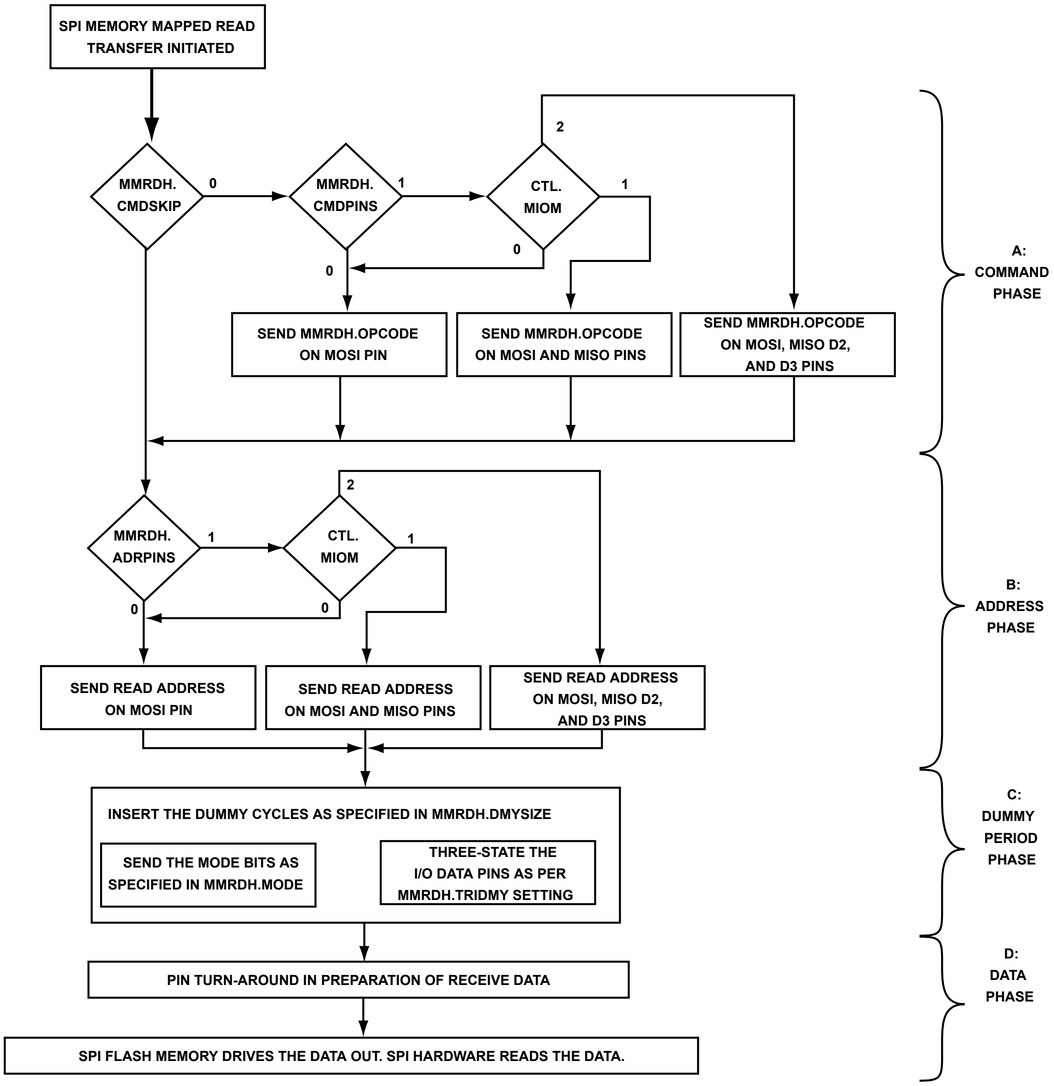

SPI MEMORY MAPPED READ TRANSFER INITIATED
MMRDH. CMDSKIP
0
MMRDH. CMDPINS
1
CTL. MIOM
2
1
0
0
SEND MMRDH.OPCODE ON MOSI PIN
SEND MMRDH.OPCODE ON MOSI AND MISO PINS
SEND MMRDH.OPCODE ON MOSI, MISO D2, AND D3 PINS
A: COMMAND PHASE
MMRDH. ADRPINS
1
CTL. MIOM
2
1
0
0
SEND READ ADDRESS ON MOSI PIN
SEND READ ADDRESS ON MOSI AND MISO PINS
SEND READ ADDRESS ON MOSI, MISO D2, AND D3 PINS
B: ADDRESS PHASE
INSERT THE DUMMY CYCLES AS SPECIFIED IN MMRDH.DMYSIZE
SEND THE MODE BITS AS SPECIFIED IN MMRDH.MODE
THREE-STATE THE I/O DATA PINS AS PER MMRDH.TRIDMY SETTING
C: DUMMY PERIOD PHASE
PIN TURN-AROUND IN PREPARATION OF RECEIVE DATA
SPI FLASH MEMORY DRIVES THE DATA OUT. SPI HARDWARE READS THE DATA.
D: DATA PHASE

Figure 16-13: SPI Memory-Mapped Register Operations Flow

Memory-Mapped Architectural Concepts

In memory-mapped mode, the SPI accepts read requests through a dedicated on-chip slave interface. The SPI (if ready) accepts these requests and begins the process of assembling the read header based on access attributes described in both the `SPI_MMRDH` register and the internal bus request. After the read header transmission is complete, a pin turnaround period is timed and the receiver is enabled. The SPI continues clocking the SPI memory device until all bytes are received.

The SPI memory-mapped hardware accommodates various memory devices with different read timing. The capabilities include extra mode bits, flexible dummy period timing, and three-state control, as configured in the `SPI_MMRDH` register.

The *Memory-Mapped Protocol* figure shows the protocol for the SPI controller in memory-mapped mode.

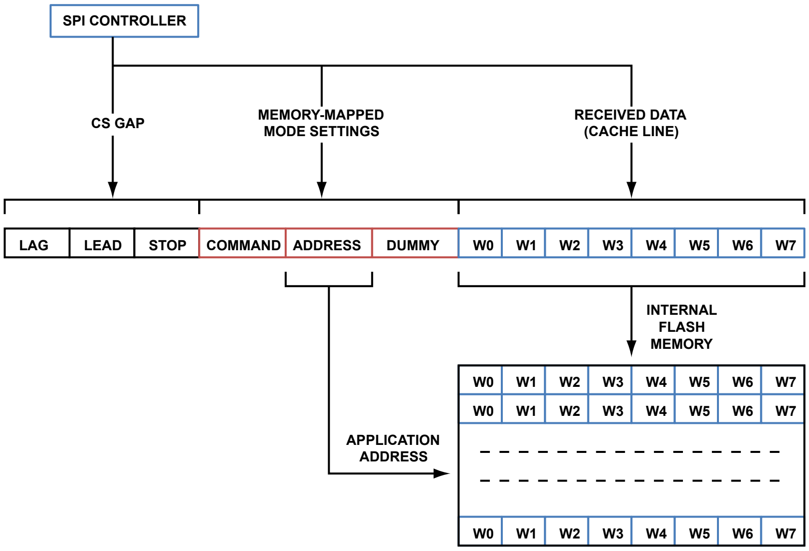

SPI CONTROLLER
CS GAP
MEMORY-MAPPED MODE SETTINGS
RECEIVED DATA (CACHE LINE)
LAG
LEAD
STOP
COMMAND
ADDRESS
DUMMY
W0
W1
W2
W3
W4
W5
W6
W7
INTERNAL FLASH MEMORY
APPLICATION ADDRESS

Figure 16-14: Memory-Mapped Protocol

As shown in the figure, the COMMAND field (`SPI_MMRDH.OPCODE`) is transmitted upon assertion of the `SPI_SEL[n]` signal. The SPI memory interprets this 8-bit value as a read command. Any 8-bit read opcode whose timing is compliant with the processor SPI and features provided by memory-mapped hardware is allowed, the most common being:

- Standard Read (0x03)
- Fast Read (0x0B)
- Fast Read Dual Output (0x3B)
- Fast Read Dual I/O (0x6B)
- Fast Read Quad Output (0xBB)
- Fast Read Quad I/O (0xEB)
- Word Read Quad I/O (0xE7)
- Octal Word Read Quad I/O (0xE3)

NOTE: The SPI hardware does not validate the content of the `SPI_MMRDH.OPCODE` field prior to transmitting.

DMYSIZE (Number of Dummy Bytes)

When operating at a high clock frequency in multi-IO modes, most flash devices require some dummy clocks after the address bits. These dummy clock cycles allow the internal circuits of the device extra time for setting up the initial address. These bits specify the number of bytes separating address transmission and read data return.

The number of dummy cycles required varies per manufacturer, the read command used, and the SPI access time. The SPI hardware allows dummy cycles to be programmed in bytes in the `SPI_MMRDH.DMYSIZE` field, the value of which is a function of the number of pins used to transmit the address (`SPI_MMRDH.ADRPINS`), as shown in the *Pins Used to Transmit the Address (ADRPINS)* table.

Table 16-7: Pins Used to Transmit the Address (ADRPINS)

	Dummy clock cycles		
SPI_MMRDH.DMYSIZE	(**SPI_MMRDH.ADRPINS**=0, **SPI_CTL.MIOM**=x) Dummy bytes elapse over 1-pin	(**SPI_MMRDH.ADRPINS**=1, **SPI_CTL.MIOM**=1) Dummy bytes elapse over 2-pins	(**SPI_MMRDH.ADRPINS**=1, MIOM=2) Dummy bytes elapse over 4-pins
000	0	0	0
001	8	4	2
010	16	8	4
011	24	12	6
100	32	16	8
101	40	20	10
110	48	24	12
111	56	28	14

This dummy clocking period allows the mode bits to be sent, the pins to be three-stated, and the pins to be turned around in preparation for the receive data.

Memory-Mapped Read Accesses

The SPI hardware supports the most commonly used read operations.

- Two standard SPI reads (read and read fast), which use the unidirectional `SPI_MOSI` and `SPI_MISO` pins in addition to `SPI_SEL[n]` and `SPI_CLK`
- Four extended SPI multiple I/O reads: dual output, quad output, dual I/O, and quad I/O reads

The *SPI Read Operations* table and *SPI Flash Fast Read Sequence* figures summarize the types of read operations. Program each read operation in a way that is compatible with the description given in the SPI flash data sheet.

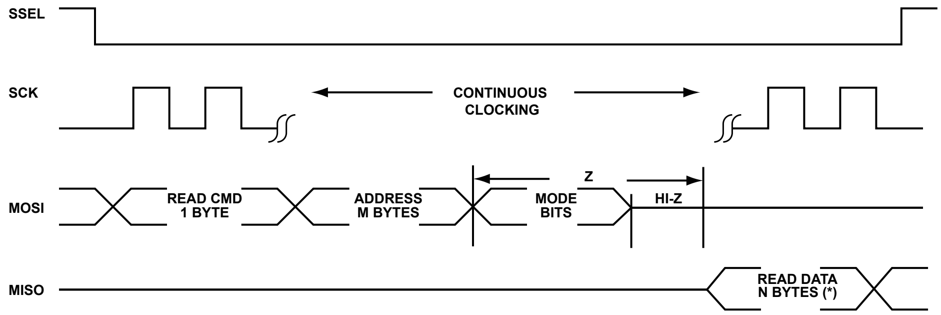
Table 16-8: SPI Read Operations

Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Read	0x03	1	1	Zero	No	No	1
Fast Read	0x0B	1	1	Non-Zero	Yes	No	1
Dual Output Read	0x3B	1	1	Non-Zero	Yes	Yes(IO0-1)	2
Quad Output Read	0x6B	1	1	Non-Zero	Yes	Yes(IO0-3)	4
Dual I/O Read	0xBB	1, 2	2	Non-Zero	Yes	Yes (IO0-1)	2

Table 16-8: SPI Read Operations (Continued)

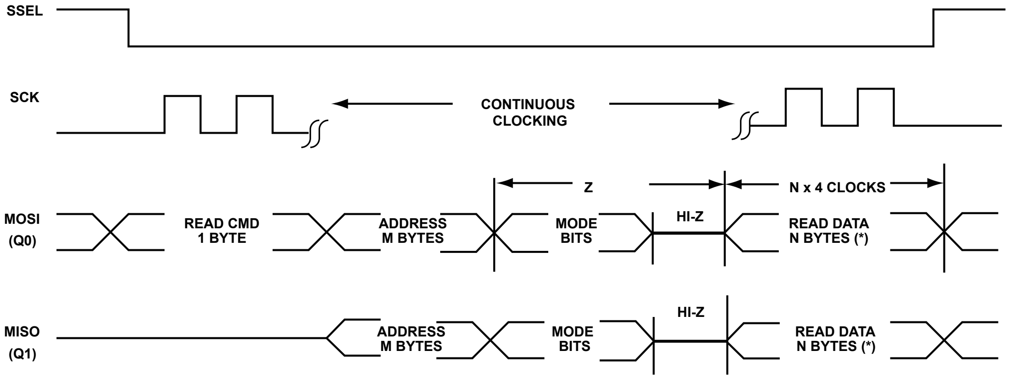
Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Quad I/O Read	0xEB	1, 4	4	Non-Zero	Yes	Yes (IO0-3)	4

Some memory devices also support word quad I/O read (0xE7) and octal quad I/O read (0xE3) operations. These operations require fewer dummy cycles than normal quad I/O read operations.


SSEL
SCK
CONTINUOUS CLOCKING
MOSI
READ CMD 1 BYTE
ADDRESS M BYTES
Z
MODE BITS
HI-Z
MISO
READ DATA N BYTES (*)

M = 1, 2, 3, 4 ADDRESS BYTES
N = 4, 8, 16, 32 READ DATA BYTES
Z = 0, 8, 16, 24, ..., 56 BUS TURN AROUND CYCLES
* = 1 DATA BIT iS SENT PER CLOCK PERIOD ON MISO

Figure 16-15: SPI Flash Fast Read Sequence


SSEL
SCK
CONTINUOUS CLOCKING
Z
N x 4 CLOCKS
MOSI (Q0)
READ CMD 1 BYTE
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
MISO (Q1)
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)

M = 1, 2, 3, 4 ADDRESS BYTES
N = 4, 8, 16, 32 READ DATA BYTES
Z = 0, 4, 12, ..., 28 BUS TURN AROUND CYCLES
* = 2 DATA BITS ARE SENT PER CLOCK PERIOD ON Q<1:0>

Figure 16-16: SPI Flash Fast Read (Dual Output) Sequence

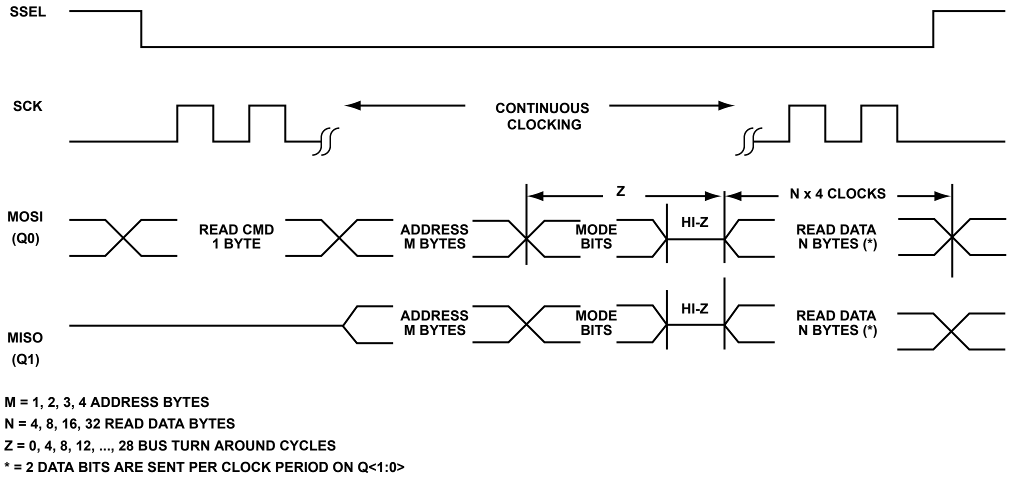

SSEL
SCK
CONTINUOUS CLOCKING
MOSI (Q0)
READ CMD 1 BYTE
ADDRESS M BYTES
Z
MODE BITS
HI-Z
N x 4 CLOCKS
READ DATA N BYTES (*)
MISO (Q1)
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
M = 1, 2, 3, 4 ADDRESS BYTES
N = 4, 8, 16, 32 READ DATA BYTES
Z = 0, 4, 8, 12, ..., 28 BUS TURN AROUND CYCLES
* = 2 DATA BITS ARE SENT PER CLOCK PERIOD ON Q<1:0>

Figure 16-17: SPI Flash Fast Read (Dual I/O) Sequence

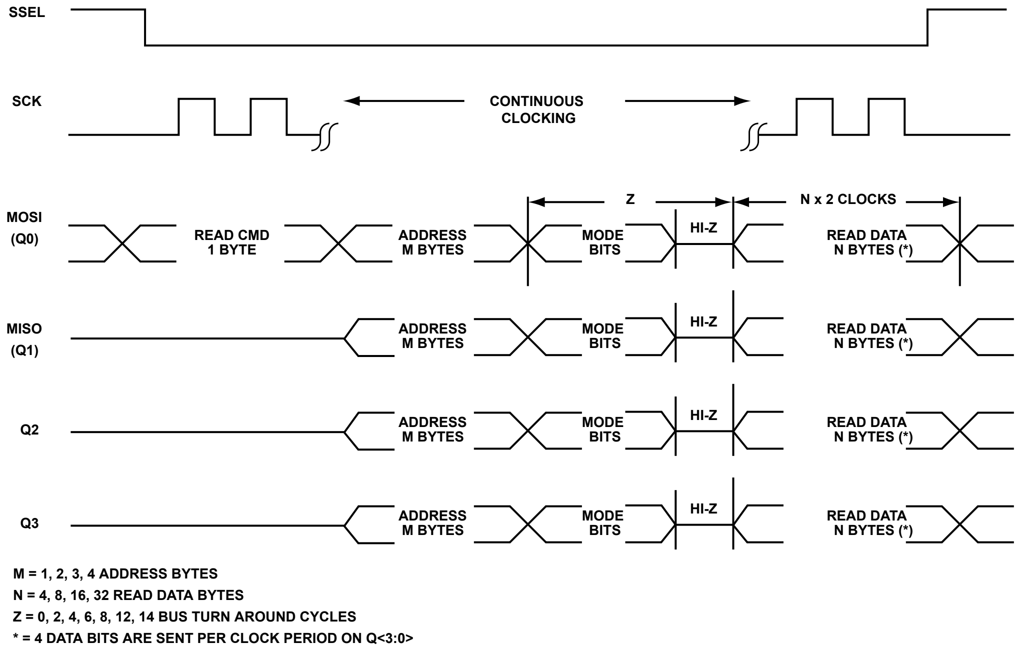
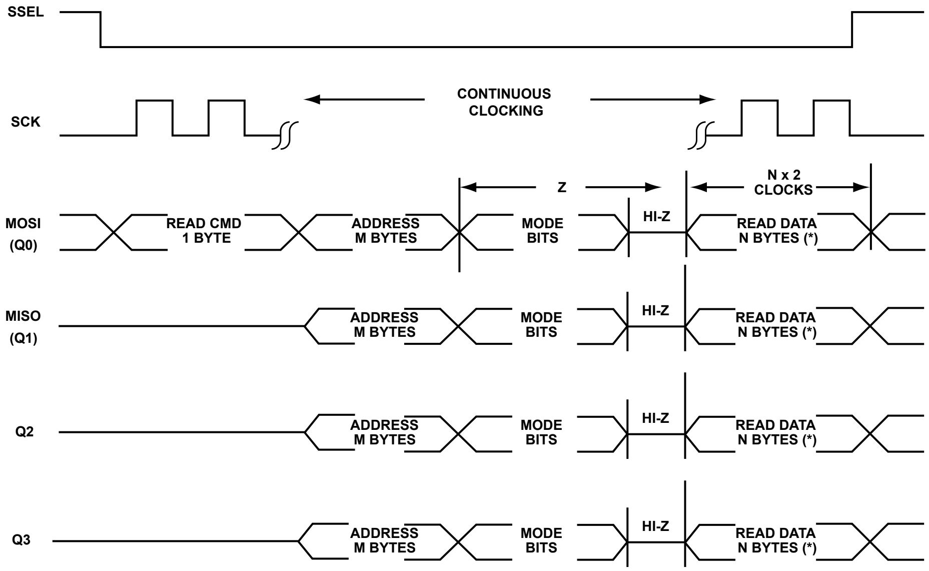

SSEL
SCK
CONTINUOUS CLOCKING
MOSI (Q0)
READ CMD 1 BYTE
ADDRESS M BYTES
Z
MODE BITS
HI-Z
N x 2 CLOCKS
READ DATA N BYTES (*)
MISO (Q1)
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
Q2
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
Q3
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
M = 1, 2, 3, 4 ADDRESS BYTES
N = 4, 8, 16, 32 READ DATA BYTES
Z = 0, 2, 4, 6, 8, 12, 14 BUS TURN AROUND CYCLES
* = 4 DATA BITS ARE SENT PER CLOCK PERIOD ON Q<3:0>

Figure 16-18: SPI Flash Quad Output Read Sequence


SSEL
SCK
CONTINUOUS CLOCKING
Z
N x 2 CLOCKS
MOSI (Q0)
READ CMD 1 BYTE
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
MISO (Q1)
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
Q2
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)
Q3
ADDRESS M BYTES
MODE BITS
HI-Z
READ DATA N BYTES (*)

M = 1, 2, 3, 4 ADDRESS BYTES
N = 4, 8, 16, 32 READ DATA BYTES
Z = 0, 2, 4, 6, 8, 12, 14 BUS TURN AROUND CYCLES
* = 4 DATA BITS ARE SENT PER CLOCK PERIOD ON Q<3:0>

Figure 16-19: SPI Flash Quad I/O Read Sequence

SPI memory-mapped reads can be made cacheable in the core's internal memory by properly configuring the region as cacheable memory without bypass (see the related core's cache configuration documentation for details). In the figures, the number of read data bytes (N) is based on the following:

- For an instruction fetch by core (when in XIP mode); the number of instruction bytes to fetch depends the cache line size of the cache.
- For a data fetch by the core (data read), the number of data bytes to fetch depends on the cache line size of the cache.

Although the minimum size of a memory-mapped data read transfer is 4 bytes, applications can fetch a single byte or a 2-byte data. (For example, it can fetch an unsigned char or short access in C code). In this case, only the required bytes are provided to the core and the other bytes are cached.

The on-chip memory subsystem master provides a starting address for the burst and the SPI hardware issues this address as part of the read header. The address provided is N-byte aligned. For example, to read the 30th byte from SPI memory, then the typical address to provide is:

- 28 (0x0000_001C) for a 32-bit cache line
- 24 (0x0000_0018) for a 64-bit cache line
- 16 (0x0000_0010) for a 128-bit cache line
- 0 (0x0000_0000) for a 256-bit cache line

The read data is returned to the memory subsystem in the order provided by the SPI memory. There can be considerable delay for the expected data provided to the master.

To minimize this delay, the wrap feature can be used where the memory subsystem provides the address of the critical word.

- For MDMA reads, the number of read data bytes (N) is always equal to 4 bytes. The MDMA read does not depend on the cache setting. For MDMA reads, limit the `DMA_CFG.MSIZE` field to 1, 2 or 4 bytes. The address provided by the memory subsystem master to the SPI hardware is always 4 byte-aligned.

Memory-Mapped High-Performance Features

In addition to automating the SPI memory read accesses, the memory-mapped hardware also provides some features to improve SPI memory fetches and increase the system performance. The following sections describe these features.

Merged Read Accesses

It is common for the memory subsystem to fetch two or more cache lines from consecutive addresses (the address sequencing is linear without any jumps). To take advantage of this situation, the SPI memory-mapped hardware provides a feature called merging. Enable merging by setting the `SPI_MMRDH.MERGE` bit.

When enabled, the hardware compares the address of an incoming read request to the address of a request the SPI memory is actively servicing. It can decide to merge two accesses when the address for the second access is incremental. For example, if the first address of a 32-byte cache line fetch is 0x0000_0000 and the second fetch is to address 0x0000_0020, then these two accesses can be merged. Merging increases efficiency and overall fetch bandwidth by eliminating the read header for those accesses which only require continuation of the SPI clock.

Wrap Around Accesses

Many SPI flash memory devices support wrapping which is used to enhance critical word fetching of cache lines. In this mode, the SPI device automatically wraps the read address to the base of a cache line once the end of the cache line is reached.

Wrap around accesses are enabled by setting the `SPI_MMRDH.WRAP` bit.

Some flash devices require programs to send a ***Set Wrap*** command to place the device in wrap mode. Other flash devices provide a configuration register which must be programmed to set the flash in wrap mode. Since the SPI memory-mapped hardware does not support any write operations to flash, perform this step in non-memory-mapped mode (`SPI_CTL.MMSE`=0) by accessing the SPI registers.

Table 16-9: Wrap Modes

Cache Line Size	Wrap Mode	Comments
4-byte	Not applicable	Usually flash does not support 4-byte wrapping.
8-byte	8 byte wrapping	Read data wraps within an aligned 8-byte boundary starting from the specified address.
16-byte	16 byte wrapping	Read data wraps within an aligned 16-byte boundary starting from the specified address.
32-byte	32 byte wrapping	Read data wraps within an aligned 32-byte boundary starting from the specified address.

Data access is limited to 8-byte, 16-byte, or 32-byte sections of flash page in wrap mode. The ARM core uses the Wrap 4 access (64-bit data) for L1 cache. The ARM core uses Wrap 4 and Wrap 8 accesses (64-bit data) for L2 cache. The cores use the Wrap 8 accesses for unaligned accesses. During the read request to the SPI memory-mapped hardware, the memory subsystem master of the processor provides the address of a critical word instead of the line base. The read-data starts at the address specified in the instruction. Once it reaches the end boundary of the 8, 16, or 32-byte section, the output automatically wraps around to the beginning boundary to the line base address. The data fetch continues. It is not necessary to deassert the SPI `SPI_SEL[n]` signal or resend the read header to wrap to the cache line base when servicing misaligned cache fill requests.

The ***Byte Sequence in Wrap Modes*** table shows byte sequences in various wrap modes.

Table 16-10: Byte Sequence in Wrap Modes

Starting Address	8-Byte Wrap (cache_line = 8 byte)
0	0-1-2- . . . -6-7
1	1-2- 3-. . . -7-0
7	7-0-1- . . . -5-6
15	15-8-9- . . . -13-14
31	31-24-25-. . . .-29-30

The burst with wrap feature allows applications to fetch a critical address quickly. Applications then fill the cache afterwards within a fixed length (8/16/32-byte) of data without issuing multiple read commands. Certain applications can benefit from this feature to improve cache fill efficiency and overall performance of system code execution.

NOTE: Do not use the merge and wrap feature together. Using wrap bursts can unintentionally disable merging (merging cannot occur for unaligned wrapping bursts). A wrap burst can start fetching data words in the middle of the cache line and cannot be merged with the next access.

Execute-In-Place (XIP, SPI2 only)

Execute-In-Place, most commonly known as XIP, allows software code to execute directly from an SPI flash device rather than downloading the code and executing it out of RAM. XIP, also known as Command Skip mode, is a general term and can be applied to fetching data as well.

There is a difference between XIP mode and standard mode. In XIP mode, after the SPI memory device is selected (CS# =LOW), the memory device does not decode the first input byte as command code. Instead, it expects the read header to directly start with address bytes. In standard mode, the memory decodes the first input byte it receives as a command code.

The XIP mode dramatically reduces random access time for applications that require fast code execution without shadowing the memory content on a RAM. The SPI memory-mapped hardware provides a control bit, `SPI_MMRDH.CMDSKIP` to skip the command from read header.

Some SPI memory devices require configuration of their control register to enable the XIP mode of operation, using the non-memory-mapped mode of the processor SPI. Typically, during the dummy cycle period, the mode bits are used to confirm the XIP operation and the `SPI_MMRDH.MODE` field must be set appropriately. A dummy memory-mapped access may be needed before setting the `SPI_MMRDH.CMDSKIP` bit in order to set the SPI memory device in Command Skip mode.

For more details about how to configure SPI memories into XIP mode, refer to the device data sheet.

> **NOTE:** When configuring the flash to XIP mode from the SHARC+ core, ensure that the routine that configures flash to XIP is not routed through the L2CC. This is accomplished by first configuring the flash to XIP mode, then enabling the L2CC from the core.

Memory-Mapped Mode Error Status Bits

The SPI memory-mapped hardware provides bits in the `SPI_STAT` register to report errors. It provides these bits for notification only and their state has no effect on SPI operations. The status register bits are sticky. A W1C (write-1-to-clear) operation clears the bits.

- Memory-Mapped Write Error (`SPI_STAT.MMWE`). This bit is set (=1) if an attempt is made to write to address space that is reserved for memory-mapped SPI memory. The SPI memory-mapped hardware does not support automated write access to SPI memory space.
- Memory-Mapped Read Error (`SPI_STAT.MMRE`). This bit is set (=1) if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (`SPI_CTL.MMSE` =0).
- Memory-Mapped Access Error (`SPI_STAT.MMAE`). This bit is set (=1) if an attempt is made to access either the TX or RX FIFO while memory-mapped access of SPI memory is enabled. In this case, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses. Legacy methods include any direct access made to the TX and RX FIFOs, whether by DMA or processor MMR.
- Memory-Mapped Write Error Mask (`SPI_CTL.MMWEM`) bit specifies whether an error response is returned to the fabric on write attempts to address space that is reserved for memory-mapped SPI memory reads. Regardless of whether a write error response is masked using this bit, the memory-mapped write error (`SPI_STAT.MMWE`) sticky notification bit is still set.

> **NOTE:** Unlike other bits in the `SPI_STAT` register, these memory-mapped mode error bits do not have associated bits in the SPI interrupt mask (`SPI_IMSK`) and SPI interrupt condition (`SPI_ILAT`) registers.

The memory-mapped top register (`SPI_MMTOP`) is used to specify the upper limit of the SPI memory address. The memory-mapped accesses to SPI memory addresses equal to or above this range are considered illegal. The accesses are blocked and a bus error response is generated.

This register is useful to block the invalid SPI memory address accesses. Some SPI memory vendors do not clearly specify (guarantee) that overrange address bits are ignored (address spaces can be wrapped).

Memory-Mapped Programming Guidelines

Seting the `SPI_CTL.MMSE` bit enables SPI memory-mapped mode. When enabled, the SPI interface is forced to be consistent with SPI memory requirements regardless the settings of certain control bits. The following tables specify typical settings for configuring the SPI in memory-mapped mode:

Table 16-11: SPI Control (SPI_CTL) Register

Bits	Typical values to set	Description	Comments
`SPI_CTL.MSTR`	1	Master mode enable	
`SPI_CTL.PSSE`	0	Protected slave select enable	
`SPI_CTL.ODM`	0	Open-drain mode enable	
`SPI_CTL.CPHASPI_CTL.CPOL`	0–0 or 1–1	SPI mode of communication	Flash dependent, usually SPI flash supports mode-0 (CPHA=CPOL=0) and mode-3 (CPHA=CPOL=1)
`SPI_CTL.ASSEL`	1	Hardware slave select pin control	
`SPI_CTL.SELST`	1	Slave select asserted between transfers	
`SPI_CTL.EMISO`	1	MISO pin enable	
`SPI_CTL.SIZE`	2	32-bit transfer size	
`SPI_CTL.LSBF`	0	MSB bit first mode	Flash dependent, usually SPI flash communicates in MSB bit first mode

`SPI_CTL.FCEN`
`SPI_CTL.FCCH`
`SPI_CTL.FCPL`
`SPI_CTL.FCWM`
	0	Hardware flow control related bits	

`SPI_CTL.FMODE`	1	Fast mode enable	Typically set to 1 for full cycle timing, 0 only works at low speed
`SPI_CTL.SOSI`	0	Treat `SPI_MOSI` pin as IO0 pin.	

Table 16-12: SPI Receive Control Register

Bits	Typical values to set	Description
`SPI_RXCTL.REN`	1	Receive channel enable
`SPI_RXCTL.RTI`	0	Receive transfer initiation disable
`SPI_RXCTL.RWCEN`	0	Receive word counter disable
`SPI_RXCTL.RDR`	0	Receive data request disable
`SPI_RXCTL.RDO`	0	Discard incoming data if RFIFO is full
`SPI_RXCTL.RRWM`	0	Receive FIFO regular watermark
`SPI_RXCTL.RUWM`	0	Receive FIFO urgent watermark disable

Table 16-13: SPI Transmit Control Register

Bits	Typical values to set	Description
`SPI_TXCTL.TEN`	1	Transmit channel enable
`SPI_TXCTL.TTI`	1	Transmit transfer initiation disable
`SPI_TXCTL.TWCEN`	0	Transmit word counter disable
`SPI_TXCTL.TDR`	0	Transmit data request disable
`SPI_TXCTL.TDU`	0	Send last word when TFIFO is empty
`SPI_TXCTL.TRWM`	0	Transmit FIFO regular watermark
`SPI_TXCTL.TUWM`	0	Transmit FIFO urgent watermark disable

Table 16-14: SPI DLY Control Register

Bits	Typical values to set	Description	Comments See Flash data sheet for CS (for example. SSEL) timing specs
`SPI_DLY.LAGX`	1	Extended lag timing	
`SPI_DLY.LEADX`	1	Extended lead timing	
`SPI_DLY.STOP`	3	Stop bit between the transfers	Can be set to 1 at lower SPI clock frequencies.

The multiple I/O mode (`SPI_CTL.MIOM`) bits are partially ignored:

- The command (opcode) is transmitted using either just one or the number of pins specified by the `SPI_CTL.MIOM` bits, depending on `SPI_MMRDH.CMDPINS` bit setting.
- The address is then transmitted using either just one or the number of pins specified by the `SPI_CTL.MIOM` bits, depending on `SPI_MMRDH.ADRPINS` bit setting.
- The data is always read with the number of pins specified by the `SPI_CTL.MIOM` bits.

NOTE: Set the SPI module enable bits `SPI_CTL.EN` last after configuring all registers.

Use the following programming guidelines for memory-mapped mode:

- The SPI memory-mapped hardware does not check the flash status before initiating the access. It assumes that SPI memory is always able to respond to a read access. Befor enabling memory-mapped mode (for example, setting the `SPI_CTL.MMSE` bit) ensure that SPI flash is ready for a read access. When using non-memory-mapped mode, a write-complete status can be examined prior to enabling the SPI in memory-mapped mode. (See the write in progress bit in the SPI flash memory status register.) Also, immediately after initial power-up, SPI memory devices can be inaccessible for a vendor-specified period.
- When SPI is enabled in memory-mapped mode, attempts to communicate with the SPI device using legacy methods are blocked. Legacy methods include any direct access made to the transmit or receive FIFOs, whether initiated by DMA or by a processor MMR access.

- To use some of the features offered by SPI memory devices, programs can first configure the SPI memory device by setting its control word or sending some commands. Since SPI memory-mapped hardware does not allow any type of SPI write operations, configure the SPI in non-memory-mapped mode prior to enabling memory-mapped mode.
- The memory-mapped hardware does not interpret the opcode. It does not check the validity of the timing that is specified in the `SPI_MMRDH` register for a particular opcode. Programs must set the fields of the `SPI_MMRDH` register to be consistent with the read-type selected.
- When the core requests the data or code fetch, the memory-mapped transfer depends on cache settings. The cache configuration register in the SPI memory device must be appropriately configured before enabling memory-mapped mode. Some of the high performance modes like merge, wrap, and transfer size depend on cache parameters.
- SPI memory-mapped MDMA reads do not support wrapping. For MDMA reads, limit the `DMA_CFG.MSIZE` field to 1 byte, 2 bytes or 4 bytes.
- There is not always tool support to change the SPI memory-mapped hardware setting or cache settings on-the-fly. Changing these settings can optimize the performance of code that accesses SPI memory in memory-mapped mode. It is expected that the SPI memory, SPI peripheral, and cache are programmed to one specific set of control settings for the whole application. Profiling or benchmarking of the actual application can be done to find the setting that works best.

SPI Interrupt Signals

The SPI controller supports three types of interrupt request signals that correspond to data, status, and error conditions.

Data Interrupts

The SPI peripheral supports two data interrupt channels – receive and transmit. These interrupt signals are multiplexed into the DMA request lines. Since the peripheral interfaces with separate read and write interfaces with DMA, the read and write data interrupts are independent. When the DMA channels are not used, the interrupts are routed directly to the system event controller. The interrupts occupy the same vector locations as the corresponding DMA channels.

Each of the data interrupt requests can be individually controlled. Program the `SPI_RXCTL.RDR` and `SPI_TXCTL.TDR` bit fields for receive and transmit, respectively. When receive is enabled, the RX interrupt request is issued whenever there is data available in the receive datapath for reading. (The event occurs according to the `SPI_RXCTL.RDR` bit setting.) When transmit is enabled, the TX interrupt request is issued whenever the transmit datapath can be written. (The event occurs according to the `SPI_TXCTL.TDR` setting.) DMA data interrupts are compatible with second-generation DMA to incorporate urgent data requests and transfer finish interrupt requests apart from the usual data request interrupts. Transmit interrupt requests operate independently from the word counter-value in the `SPI_TWC` register.

Status Interrupts

The SPI controller supports several status interrupt requests to indicate different conditions of the receiver and transmitter. All status interrupt requests can be masked. Status interrupt requests are signaled directly through a single SPI status IRQ line. The line cannot be combined with the SPI error IRQ line for some processors. The ***SPI Status Interrupts*** table describes the status interrupt requests that are available for the SPI controller.

Table 16-15: SPI Status Interrupts

SPI_STAT Bit	Description
`SPI_STAT.RUWM`	Receive FIFO urgent watermark interrupt request. Issued when the level of the RFIFO breaches the watermark set in the `SPI_RXCTL.RUWM` field. It is cleared when the level of the RFIFO reaches the watermark set in the `SPI_RXCTL.RRWM` field. If the RX channel is configured in DMA mode, `SPI_RXCTL.RUWM` is multiplexed with the data request.
`SPI_STAT.TUWM`	Receive FIFO urgent watermark interrupt request. Issued when the level of the TFIFO breaches the watermark set using the `SPI_TXCTL.TUWM` bit. It is cleared when the level of the TFIFO reaches the watermark set in the `SPI_TXCTL.TRWM` field. If the TX channel is configured in DMA mode, `SPI_STAT.TUWM` is multiplexed with the data request.
`SPI_STAT.TS`	Transmit start interrupt request. Issued when the start of a transmit burst is detected by loading of the `SPI_TWC` register with the contents of the `SPI_TWCR` register.
`SPI_STAT.RS`	Receive start interrupt request. Issued when the start of a receive burst is detected by the loading of `SPI_RWC` with the contents of `SPI_RWCR`.
`SPI_STAT.TF`	Transmit finish interrupt request. Issued when a transmit burst completes (`SPI_TWC` decrements to zero).
`SPI_STAT.RF`	Receive finish interrupt request. Issued when a receive burst completes (`SPI_RWC` decrements to zero).

Error Conditions

The SPI controller supports interrupt requests upon several different error conditions. All interrupt requests are maskable. The individual error indications combine into a single SPI error IRQ signal, which can be multiplexed on some processors with the aggregated SPI status IRQ signal. The ***SPI Error Interrupts*** table details the possible error indications.

Error conditions arise depending on which of the channels (transmit or receive) are enabled. If a channel is disabled, all errors related to it are ignored. When both channels are enabled, errors from both channels are enabled.

Table 16-16: SPI Error Interrupts

Bit	Description
`SPI_STAT.MF`	Mode fault. Signaled when another device also tries to be a master in a multi-master system and drives the `SPI_SS` input low. This error is signaled in master mode operation.
`SPI_STAT.TUR`	Transmission error. Signaled when an underflow condition occurs on the transmit channel. This event occurs when a new transfer starts but `SPI_TFIFO` is empty. This error does not occur in master transmit initiating mode since `SPI_TFIFO` *Not Empty* is one of the conditions for transfer initiation.

Table 16-16: SPI Error Interrupts (Continued)

Bit	Description
`SPI_STAT.ROR`	Reception error. Signaled when an overflow condition occurs on the receive channel. This event occurs when a new data word is received, but the `SPI_RFIFO` is full. This error condition does not occur in master receive initiating mode since `SPI_RFIFO` *Not Full* is one of the conditions for transfer initiation.
`SPI_STAT.TC`	Transmit collision error. Signaled when loading data to the transmit shift register happens near the first transmitting edge of `SPI_CLK`. In slave mode of operation, the SPI controller is unaware of when the next transfer starts. Loading of data to the transmit shift register can happen just after the transmitting edge. This event results in the setup time not being met for the first bit transmitted. The transmitted data is corrupt. In `SPI_CTL.CPHA` 1 mode, the first `SPI_CLK` edge is taken as the first transmitting edge. If `SPI_CTL.CPHA` =0, then the last `SPI_CLK` edge of the last transmission (`SPI_CTL.SELST` =1) or slave select deassertion (`SPI_CTL.SELST` =0) is taken as the first transmitting edge. This error is signaled only in the slave mode of operation. In master mode of operation, loading of data happens before the first transmitting edge of `SPI_CLK`.

SPI Programming Concepts

The following sections provide general programming guidelines and procedures.

Programming Guidelines

It is acceptable to program `SPI_RXCTL` and `SPI_TXCTL` registers after programming the `SPI_CTL` register. However, program the initiating mode register and its counter-register, if enabled, after the non-initiating mode register. For example, if transmit is the initiating mode and receive is the non-initiating mode, then program the `SPI_RXCTL` and `SPI_RWC` registers before the `SPI_TXCTL` and `SPI_TWC` registers. If enabling both transmit and receive in initiating mode, enable the `SPI_CTL` register after programming both the `SPI_RXCTL` and `SPI_TXCTL` registers.

These programming guidelines prevent SPI from starting a transfer when SPI registers are not fully programmed. Other ways of programming are also allowed as long as the initiating conditions prevent the start of communication until after programming of SPI registers is complete.

Take precautions to avoid data corruption when changing the SPI module configuration. Do not change the configuration during a data transfer. Additionally, change the clock polarity only when no slaves are selected. However, an exception to this rule exists. When an SPI communication link consists of a single master and slave, `SPI_CTL.ASSEL` = 0. The slave select input of the slave is permanently tied low. In this case, the slave is always selected. Avoid data corruption by enabling the slave only after both the master and slave devices are configured.

The module supports 8, 16-bit and 32-bit word sizes. To ensure correct operation, configure both the master and slave with the same word size.

Master Operation in Non-DMA Modes

This section describes the operation of the SPI as a master in non-DMA mode.

1. Write to the `SPI_SLVSEL` register, setting one or more of the SPI select enable bits. This operation ensures that the desired slaves are properly deselected while the master is configured.
2. The `SPI_RXCTL.RTI` and `SPI_TXCTL.TTI` bits determine the SPI initiating mode. The initiating mode defines the primary transfer channel, and also the initiating condition for the transfer.
3. Write to the `SPI_CLK`, `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers. This operation enables the device as a master and configures the SPI system. It specifies the transfer modes and channels, appropriate word length, transfer format, baud rate, and other control information.

   ADDITIONAL INFORMATION: If `SPI_RXCTL.RTI` is enabled and `SPI_TXCTL.TTI` is not, write to the `SPI_RXCTL` register after writing into `SPI_CTL`, `SPI_TXCTL`, and `SPI_TFIFO` registers to prevent a transmit underrun for the first transfer.
4. If `SPI_CTL.ASSEL`=0, activate the desired slaves by clearing one or more of the `SPI_SLVSEL` flag bits. Otherwise, the SPI hardware performs slave activation.
5. The SPI controller then generates the programmed clock pulses on `SPI_CLK` and simultaneously shifts data out of `SPI_MOSI` while shifting data in from `SPI_MISO`. Before a shift, the shift register is loaded with the contents of the `SPI_TFIFO` register. At the end of the transfer, the contents of the shift register are loaded into `SPI_RFIFO`.
6. Whenever the initiating conditions are satisfied, the SPI continues to send and receive words. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the `SPI_TXCTL.TDU` and `SPI_RXCTL.RDO` bits.
7. It is possible to program a secondary channel in addition to the initiating channel. This feature allows usage of available channel resources for receives or transmits simultaneously with the initiating channel.

Slave Operation in Non-DMA Modes

When a device is enabled as a slave in a non-DMA mode, a transition of the $\overline{\texttt{SPI\_SS}}$ select signal to the active state (low) triggers the the start of a transfer. Or, the first active edge of `SPI_CLK` triggers the start, depending on the state of `SPI_CTL.CPHA` bit. The interface operates in the following manner.

1. The core writes to the `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers. The operation defines the mode of the serial link to be the same as the mode setup in the SPI master.
2. To prepare for the data transfer, the core writes data to be transmitted into `SPI_TFIFO`.
3. Once the $\overline{\texttt{SPI\_SS}}$ falling edge is detected, the slave starts sending data on active `SPI_CLK` edges and sampling data on inactive `SPI_CLK` edges.
4. Reception or transmission continues until $\overline{\texttt{SPI\_SS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive or transmit with each new falling edge transition on $\overline{\texttt{SPI\_SS}}$ or active `SPI_CLK` edge. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the `SPI_TXCTL.TDU` and `SPI_RXCTL.RDO` bits.

Configuring DMA Master Mode

The SPI interface supports a write DMA channel and a read DMA channel. It can use these functions individually or in a lock-step manner in duplex mode (`SPI_TXCTL.TTI`= `SPI_RXCTL.RTI`=1).

1. Write to the appropriate DMA registers to enable the SPI DMA channel and to configure the necessary work units, access direction, word count, and so on.
2. Write to the `SPI_SLVSEL` register, setting one or more of the SPI flag select bits.
3. Write to the `SPI_CLK` and `SPI_CTL` registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and so forth.
4. Write to `SPI_RXCTL` to configure SPI master receive mode, or write to `SPI_TXCTL` to configure SPI master transmit mode.
5. Finally, write to the `SPI_RXCTL.REN` bit to enable the receive channel, or write to `SPI_TXCTL.TEN` to enable the transmit channel.
6. If the `SPI_RXCTL.RTI` bit is enabled, a receive transfer is initiated upon enabling `SPI_CTL.EN` bit. If the receive word counter is enabled (`SPI_RXCTL.RWCEN`), then the `SPI_RWC` register must be non-zero for a transfer to initiate.

   ADDITIONAL INFORMATION: If enabling both receive and transmit DMA channels, but not enabling `SPI_TXCTL.TTI`, write to the `SPI_RXCTL` register after writing the `SPI_CTL` and `SPI_TXCTL` registers. In this way, a transmit underrun can be prevented for the first transfer. Subsequent transfers are initiated as the SPI reads data from the receive shift register and writes to the SPI receive FIFO. The SPI then requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the SPI receive FIFO and writes to memory. New requests continue to be initiated as long as the receive FIFO does not fill up, when `SPI_RWC` does not become zero while `SPI_RXCTL.RWCEN`=1.

7. If `SPI_TXCTL.TTI` is enabled, the SPI controller requests DMA reads from memory as long as there is space for more data in the transmit pipe. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO. As long as transmit data is available in the FIFO, and the `SPI_TWC` register is non-zero when `SPI_TXCTL.TWCEN`=1, the SPI continues to initiate transfers until disabled.
8. If both the `SPI_TXCTL.TTI` and `SPI_RXCTL.RTI` bits are enabled, the SPI controller requests a DMA read from memory. However, there must be space for more data in the transmit pipe and the number of words written into the SPI must be less than `SPI_TWC` if `SPI_TXCTL.TWCEN`=1. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.

   ADDITIONAL INFORMATION: As the SPI writes data from the transmit FIFO into the transmit shift register, it initiates a transfer on the SPI link. Data received from the transfer is moved from the SPI receive shift register to the receive FIFO. The SPI controller requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory. Transfer continues to be initiated as long as both receives and transmits can accommodate new data.

9. If the receive pipe fills up due to unavailability of DMA grants, the transmit pipe stalls until the pipe is drained. If the transmit pipe fills up, the SPI stops requesting for DMA writes. If the value in `SPI_RWC` expires, further write-requests to DMA stop. However, data already written into the transmit FIFO is sent, and read requests to DMA continue until the receive data is read from the receive FIFO.
10. The SPI then generates the programmed clock pulses on `SPI_CLK` and simultaneously shifts data out of `SPI_MOSI` while shifting data in from `SPI_MISO`. For receive transfers, the value in the shift register is loaded into the `SPI_RFIFO` register at the end of the transfer. For transmit transfers, the value in the `SPI_TFIFO` register is loaded into the shift register at the start of the transfer.

Configuring DMA Slave Mode Operation

This mode occurs when the SPI is enabled as a slave and the DMA engine is configured to transmit or receive data. A transition of the `SPI_SS` signal to the active-low state triggers the start of a transfer. Or, the first active edge of `SPI_CLK` triggers the start of a transfer, depending on the state of the `SPI_CTL.CPHA` bit. The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave (in response to a master command). The SPI supports a receive DMA channel and a transmit DMA channel.

1. Write to the appropriate DMA registers to enable the SPI DMA channel and configure the necessary work units, access direction, word count, and so on.
2. Write to the `SPI_CTL`, `SPI_RXCTL`, and `SPI_TXCTL` registers to define the mode of the serial link to be the same as the mode configured in the SPI master.
3. If the receive channel is enabled (`SPI_RXCTL.REN` is asserted), the following actions occur:
    a. Once the slave select input is active, the slave starts receiving and transmitting data on active `SPI_CLK` edges.
    b. The value in the shift register is loaded into the `SPI_RFIFO` register at the end of the transfer.
    c. Once `SPI_RFIFO` has valid data, it requests a write from DMA to memory.
    d. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory.
    e. As long as there is data in the receive FIFO, the SPI slave continues to request a DMA write to memory. The DMA engine continues to read a word from the FIFO and writes to memory until the `SPI_RWC` counts to zero. The SPI slave continues receiving words on active `SPI_CLK` edges as long as the `SPI_SS` input is active.
    f. If the data collected in the receive pipe breaches the set level, and the DMA engine cannot keep up with the receive rate, the slave can deassert the `SPI_RDY` signal. This signaling throttles the master. The receive pipe level is set according to the `SPI_CTL.FCWM` field. The signal is deasserted as the DMA drains the receive FIFO. Alternatively, the SPI can use the `SPI_RXCTL.RDO` bit to decide when the incoming data is discarded or overwritten into the receive FIFO (when `SPI_CTL.FCEN` is inactive).
4. If the transmit channel is enabled (`SPI_TXCTL.TEN` is asserted), the following actions occur:
    a. The SPI requests a DMA read from memory.

   b. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.
   c. The SPI then reads DMA data from the transmit FIFO and writes to the transmit shift register, awaiting the start of the next transfer.
   d. Once the slave select input is active, the slave starts receiving and transmitting data on active `SPI_CLK` edges.
   e. As long as there is room in the transmit FIFO, the SPI slave continues to request a DMA read from memory. The DMA engine continues to read a word from memory and write to the transmit FIFO until the `SPI_TWC` register value counts down to 0. The SPI slave continues transmitting words on active `SPI_CLK` edges as long as the $\overline{\texttt{SPI\_SS}}$ input is active.
   f. If the number of outstanding data entries in the transmit pipe breaches the level set and the DMA cannot keep up with the transmit rate, the slave deasserts the `SPI_RDY` signal. This signaling throttles the master. The transmit pipe level is set according to the `SPI_CTL.FCWM` field. The signal is deasserted as the DMA fills the transmit FIFO. Alternately, the `SPI_TXCTL.TDU` bit decides the state of the transmit data (when `SPI_CTL.FCEN` is deasserted).

5. If both receive and transmit channels are enabled, the following actions occur after the actions for each channel. Transfers continue as long as both receive and transmit channels can accommodate new data.
   a. If the receive pipe fills up due to the unavailability of DMA grant, the SPI interface stalls the master by asserting the `SPI_RDY` pin. This signal is deasserted as the DMA drains the receive FIFO. Alternately, the SPI uses the `SPI_RXCTL.RDO` bit to decide when the incoming data is discarded or overwritten in the receive FIFO (when `SPI_CTL.FCEN` is deasserted).
   b. If the transmit pipe fills up, the SPI stops requesting DMA writes until the pipe clears.
   c. If there is an underflow problem in the transmit pipe, the slave stalls the master by deasserting `SPI_RDY` while the DMA fills the transmit FIFO. Alternately, the SPI uses the `SPI_TXCTL.TDU` bit to decide the state of the transmit data (when `SPI_CTL.FCEN` is deasserted).

ADSP-SC58x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 16-17: ADSP-SC58x SPI Register List

Name	Description
`SPI_CLK`	Clock Rate Register
`SPI_CTL`	Control Register
`SPI_DLY`	Delay Register
`SPI_ILAT`	Masked Interrupt Condition Register
`SPI_ILAT_CLR`	Masked Interrupt Clear Register

Table 16-17: ADSP-SC58x SPI Register List (Continued)

Name	Description
`SPI_IMSK`	Interrupt Mask Register
`SPI_IMSK_CLR`	Interrupt Mask Clear Register
`SPI_IMSK_SET`	Interrupt Mask Set Register
`SPI_MMRDH`	Memory Mapped Read Header (Only on SPI2)
`SPI_MMTOP`	SPI Memory Top Address (Only on SPI2)
`SPI_RFIFO`	Receive FIFO Data Register
`SPI_RWC`	Received Word Count Register
`SPI_RWCR`	Received Word Count Reload Register
`SPI_RXCTL`	Receive Control Register
`SPI_SLVSEL`	Slave Select Register
`SPI_STAT`	Status Register
`SPI_TFIFO`	Transmit FIFO Data Register
`SPI_TWC`	Transmitted Word Count Register
`SPI_TWCR`	Transmitted Word Count Reload Register
`SPI_TXCTL`	Transmit Control Register

Clock Rate Register

The `SPI_CLK` register selects the baud rate for SPI data transfers, relating this rate to the SPI serial clock (SPI clock) and the system clock (SCLK1_0).

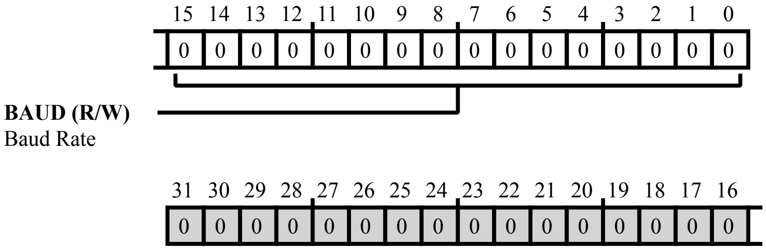

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BAUD (R/W)
Baud Rate
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-20: SPI_CLK Register Diagram

Table 16-18: SPI_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	BAUD	Baud Rate. The `SPI_CLK.BAUD` bits set the SPI baud rate according to the formula: BAUD = (SCLK1_0 / SPI Clock) - 1

Control Register

The SPI_CTL register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

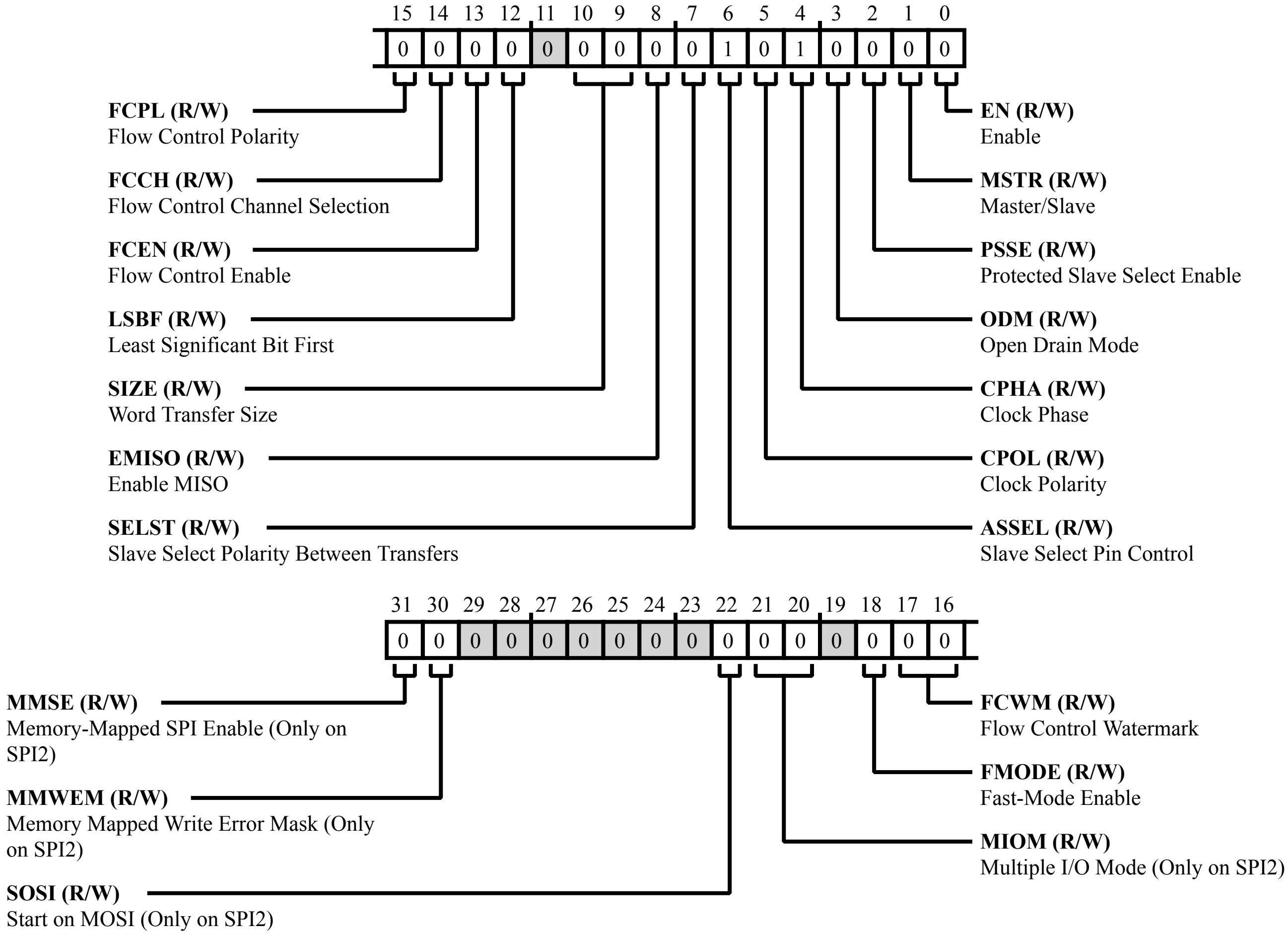

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
FCPL (R/W)
Flow Control Polarity
FCCH (R/W)
Flow Control Channel Selection
FCEN (R/W)
Flow Control Enable
LSBF (R/W)
Least Significant Bit First
SIZE (R/W)
Word Transfer Size
EMISO (R/W)
Enable MISO
SELST (R/W)
Slave Select Polarity Between Transfers
EN (R/W)
Enable
MSTR (R/W)
Master/Slave
PSSE (R/W)
Protected Slave Select Enable
ODM (R/W)
Open Drain Mode
CPHA (R/W)
Clock Phase
CPOL (R/W)
Clock Polarity
ASSEL (R/W)
Slave Select Pin Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MMSE (R/W)
Memory-Mapped SPI Enable (Only on SPI2)
MMWEM (R/W)
Memory Mapped Write Error Mask (Only on SPI2)
SOSI (R/W)
Start on MOSI (Only on SPI2)
FCWM (R/W)
Flow Control Watermark
FMODE (R/W)
Fast-Mode Enable
MIOM (R/W)
Multiple I/O Mode (Only on SPI2)

Figure 16-21: SPI_CTL Register Diagram

Table 16-19: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	MMSE	Memory-Mapped SPI Enable (Only on SPI2). When the `SPI_CTL.MMSE` bit is asserted, communication to an SPI memory device is automated such that the memory it contains is accessible directly through the read of processor address space assigned to it. (As far as the SPI peripheral is concerned, this includes all read accesses received by the SPI peripherals system crossbar slave port.) Note that when memory-mapped access of SPI memory is enabled, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses are generated. Legacy methods include any direct access made to the Tx and Rx FIFOs, whether initiated by DMA or processor MMR access.	
		0	Hardware automated access of memory-mapped SPI memory disabled.
		1	Hardware-automated access of memory-mapped SPI memory enabled.
30 (R/W)	MMWEM	Memory Mapped Write Error Mask (Only on SPI2). The `SPI_CTL.MMWEM` bit specifies whether an error response is returned to the fabric upon write attempts to address space reserved for memory-mapped reads of SPI memory.	
		0	Write error response returned upon write attempts to memory-mapped SPI memory
		1	Write error response masked (not returned) upon write attempts to memory-mapped SPI memory
22 (R/W)	SOSI	Start on MOSI (Only on SPI2). The `SPI_CTL.SOSI` bit is valid only when `SPI_CTL.MIOM` is enabled for either DIOM or QIOM, and this bit selects the starting pin and the bit placement on pins for these modes. In DIOM, by default, (`SPI_CTL.SOSI` =0) SPI sends the first bit on the `SPI_MISO` pin and the second bit on the `SPI_MOSI` pin. In QIOM, by default, the SPI sends the first bit on the `SPI_D3` pin, the second bit on the `SPI_D2` pin, the third bit on the `SPI_MISO` pin and the fourth bit on the `SPI_MOSI` pin. This order can be reversed by setting the `SPI_CTL.SOSI` bit. When this bit is set, the SPI sends the first bit on the `SPI_MOSI` pin. The first bit referred to here depends on the `SPI_CTL.LSBF` bit setting (MSB bit or LSB bit).	
		0	Start on MISO (DIOM) or start on SPI_D3 (Only on SPI2)
		1	Start on MOSI

Table 16-19: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	MIOM	Multiple I/O Mode (Only on SPI2). The `SPI_CTL.MIOM` bits enable SPI operation in dual I/O mode (DIOM) or quad I/O mode (QIOM). These bits can only be changed when the SPI is disabled (`SPI_CTL.EN` =0).	
		0	No MIOM (disabled)
		1	DIOM operation
		2	QIOM operation (Only on SPI2)
		3	Reserved
18 (R/W)	FMODE	Fast-Mode Enable. The `SPI_CTL.FMODE` bit enables fast mode operation for SPI receive transfers. SPI transmit operations in fast mode are the same as normal mode.	
		0	Disable
		1	Enable
17:16 (R/W)	FCWM	Flow Control Watermark. The `SPI_CTL.FCWM` bits select the watermark level of the transmit channel (`SPI_TFIFO` buffer) or receive channel (`SPI_RFIFO` buffer) that triggers flow control operation. These bits are applicable only when the SPI is a slave (`SPI_CTL.MSTR` = 0) and flow control is enabled (`SPI_CTL.FCEN` =1). When the watermark condition is met, the SPI slave deasserts the `SPI_RDY` pin.	
		0	TFIFO empty or RFIFO full
		1	TFIFO 75% or more empty, or RFIFO 75% or more full
		2	TFIFO 50% or more empty, or RFIFO 50% or more full
		3	Reserved
15 (R/W)	FCPL	Flow Control Polarity. The `SPI_CTL.FCPL` bit selects flow control polarity for the `SPI_RDY` pin when flow control is enabled. When the `SPI_RDY` pin is active, the SPI is indicating it is ready for data transfer.	
		0	Active-low RDY
		1	Active-high RDY

Table 16-19: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	FCCH	Flow Control Channel Selection. The `SPI_CTL.FCCH` bit selects whether the SPI applies flow control to the transmit channel (`SPI_TFIFO` buffer) or receive channel (`SPI_RFIFO` buffer). This bit is applicable only when the SPI is a slave and flow control is enabled.	
		0	Flow control on RX buffer
		1	Flow control on TX buffer
13 (R/W)	FCEN	Flow Control Enable. The `SPI_CTL.FCEN` bit enables SPI flow control operation, which permits slow slave devices to interface with fast master devices. This bit controls the operation of the `SPI_RDY` pin. Note that options for flow control operation are available using the `SPI_CTL.FCCH`, `SPI_CTL.FCPL`, and `SPI_CTL.FCWM` bits.	
		0	Disable
		1	Enable
12 (R/W)	LSBF	Least Significant Bit First. The `SPI_CTL.LSBF` bit selects whether the SPI transmits/receives data as LSB first (little endian) or MSB first (big endian). This bit can only be changed when the SPI is disabled.	
		0	MSB sent/received first (big endian)
		1	LSB sent/received first (little endian)
10:9 (R/W)	SIZE	Word Transfer Size. The `SPI_CTL.SIZE` bits select the SPI transfer word size as 8, 16 or 32 bits. To ensure correct operation, both the master and slave must be configured with the same word size. This bit can only be changed when the SPI is disabled (`SPI_CTL.EN` =0).	
		0	8-bit word
		1	16-bit word
		2	32-bit word
		3	Reserved
8 (R/W)	EMISO	Enable MISO. The `SPI_CTL.EMISO` bit enables master-in-slave-out (MISO) mode. This SPI mode is applicable only when the SPI is a slave.	
		0	Disable
		1	Enable

Table 16-19: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The `SPI_CTL.SELST` bit selects the state (polarity) for the $\overline{\texttt{SPI\_SEL[n]}}$ pin between SPI transfers when the SPI is a master and hardware slave select assertion is enabled (`SPI_CTL.ASSEL` =1). In slave mode, this bit affects the detection of both transmit collision (`SPI_STAT.TC` and underrun (`SPI_STAT.TUR`) errors.	
		0	Deassert slave select (high)
		1	Assert slave select (low)
6 (R/W)	ASSEL	Slave Select Pin Control. The `SPI_CTL.ASSEL` bit selects whether the SPI hardware sets the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value (ignoring the slave select `SPI_SLVSEL.SSEL1` - `SPI_SLVSEL.SSEL7` bits) or whether software control of the slave select bits set the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the $\overline{\texttt{SPI\_SEL[n]}}$ pin output is asserted during the transfers, and the pin polarity between transfers is selected by the `SPI_CTL.SELST` bit. When software control is enabled, the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value is set through software control of the slave select bits, and as such, the pin may either remain asserted (low) or be deasserted between transfers.	
		0	Software slave select control
		1	Hardware slave select control
5 (R/W)	CPOL	Clock Polarity. The `SPI_CTL.CPOL` bit selects whether the SPI uses an active-low or active-high signal for the SPI clock (`SPI_CLK`). This bit works with the `SPI_CTL.CPHA` bit to select combinations of clock phase and polarity for the `SPI_CLK` pin. This bit can only be changed when the SPI is disabled.	
		0	Active-high SPI CLK
		1	Active-low SPI CLK
4 (R/W)	CPHA	Clock Phase. The `SPI_CTL.CPHA` bit selects whether the SPI starts toggling the signal for the SPI clock (`SPI_CLK`) from the start of the first data bit or from the middle of the first data bit. The `SPI_CTL.CPHA` bit works with the `SPI_CTL.CPOL` bit to select combinations of clock phase and polarity for the `SPI_CLK` pin. This bit can only be changed when the SPI is disabled.	
		0	SPI CLK toggles from middle
		1	SPI CLK toggles from start

Table 16-19: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ODM	Open Drain Mode. The `SPI_CTL.ODM` bit configures the data output pins (`SPI_MOSI` and `SPI_MISO`) to behave as open drain outputs, which prevents contention and possible damage to pin drivers in multi-master or multi-slave SPI systems. When `SPI_CTL.ODM` is enabled and the SPI is a master, the SPI three-states the `SPI_MOSI` pin when the data driven out on MOSI is a logic-high. The SPI does not three-state the `SPI_MOSI` pin when the driven data is a logic-low. When `SPI_CTL.ODM` is enabled and the SPI is a slave, the SPI three-states the `SPI_MISO` pin when the data driven out on `SPI_MISO` is a logic-high. Note that an external pull-up resistor is required on both the `SPI_MOSI` and `SPI_MISO` pins when `SPI_CTL.ODM` is enabled.	
		0	Disable
		1	Enable
2 (R/W)	PSSE	Protected Slave Select Enable. The `SPI_CTL.PSSE` bit enables the $\overline{\texttt{SPI\_SS}}$ pin to provide error detection input in a multi-master environment when the SPI is in master mode. If some other device in the system asserts the $\overline{\texttt{SPI\_SS}}$ pin while SPI is enabled as master (and `SPI_CTL.PSSE` is enabled), this condition causes a mode fault error.	
		0	Disable
		1	Enable
1 (R/W)	MSTR	Master/Slave. The `SPI_CTL.MSTR` bit toggles the SPI between master mode and slave mode. This bit can only be changed when the SPI is disabled.	
		0	Slave
		1	Master
0 (R/W)	EN	Enable. The `SPI_CTL.EN` bit enables SPI operation.	
		0	Disable SPI module
		1	Enable

Delay Register

The `SPI_DLY` register selects a transfer delay and the lead/lag timing between slave select signals and SPI clock edge assertion/deassertion.

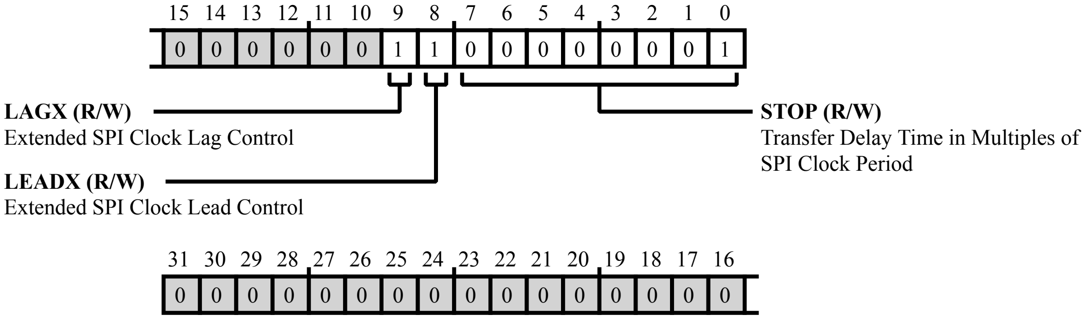

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1
LAGX (R/W)
Extended SPI Clock Lag Control
LEADX (R/W)
Extended SPI Clock Lead Control
STOP (R/W)
Transfer Delay Time in Multiples of SPI Clock Period
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-22: SPI_DLY Register Diagram

Table 16-20: SPI_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	LAGX	Extended SPI Clock Lag Control. The `SPI_DLY.LAGX` bit enables insertion of a 1-SPI_CLK cycle lag (extend lag) in the timing between the slave select (`SPI_SEL[n]`) assertion and first SPI clock edge.	
		0	Disable
		1	Enable
8 (R/W)	LEADX	Extended SPI Clock Lead Control. The `SPI_DLY.LEADX` bit enables insertion of a 1-SPI_CLK cycle lead (extend lead) in the timing between the slave select (`SPI_SEL[n]`) deassertion and last SPI clock edge.	
		0	Disable
		1	Enable
7:0 (R/W)	STOP	Transfer Delay Time in Multiples of SPI Clock Period. The `SPI_DLY.STOP` bits select a delay (number of stop bits in multiples of SPI clock duration) at the end of each SPI transfer. The default delay is the minimum value required to comply with the SPI protocol (1-bit duration). The `SPI_DLY.STOP` bits can be programmed with smaller delay values, resulting in continuous operation (for example, stop bits =0).	

Masked Interrupt Condition Register

The SPI_ILAT register latches interrupts, queuing the interrupt requests for service. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt request is unmasked in SPI_IMSK, the SPI latches the interrupt request bit in SPI_ILAT.

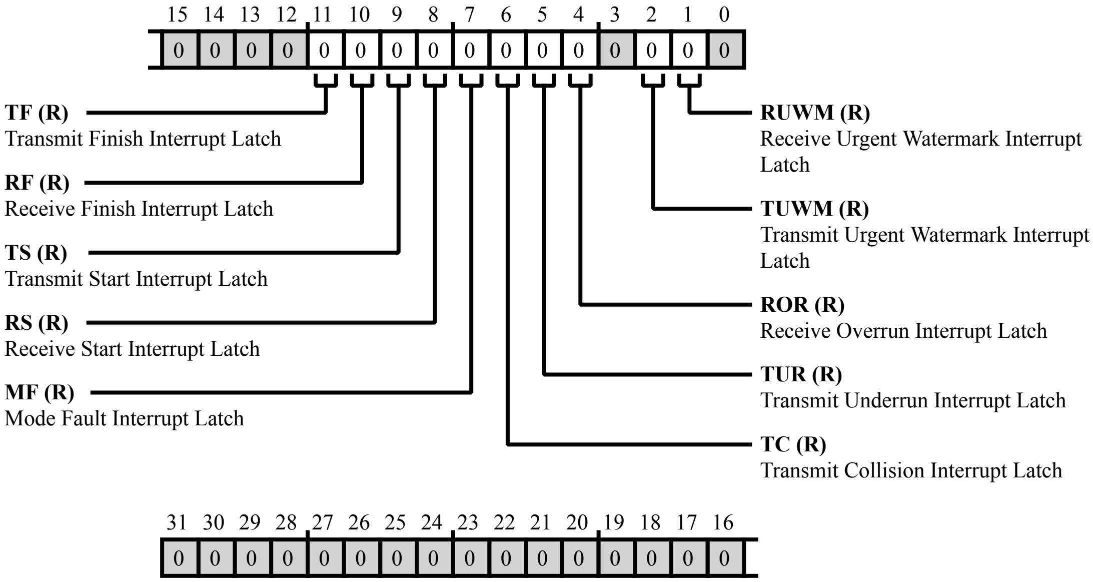

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TF (R)
Transmit Finish Interrupt Latch
RF (R)
Receive Finish Interrupt Latch
TS (R)
Transmit Start Interrupt Latch
RS (R)
Receive Start Interrupt Latch
MF (R)
Mode Fault Interrupt Latch
RUWM (R)
Receive Urgent Watermark Interrupt Latch
TUWM (R)
Transmit Urgent Watermark Interrupt Latch
ROR (R)
Receive Overrun Interrupt Latch
TUR (R)
Transmit Underrun Interrupt Latch
TC (R)
Transmit Collision Interrupt Latch
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-23: SPI_ILAT Register Diagram

Table 16-21: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
10 (R/NW)	RF	Receive Finish Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
9 (R/NW)	TS	Transmit Start Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
8 (R/NW)	RS	Receive Start Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request

Table 16-21: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	MF	Mode Fault Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
6 (R/NW)	TC	Transmit Collision Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
5 (R/NW)	TUR	Transmit Underrun Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request

Masked Interrupt Clear Register

The `SPI_ILAT_CLR` register permits clearing individual mask bits in the `SPI_ILAT` register without affecting other bits in the register. Use write-1-to-clear on a bit in the `SPI_ILAT_CLR` register to clear the corresponding bit in the `SPI_ILAT` register.

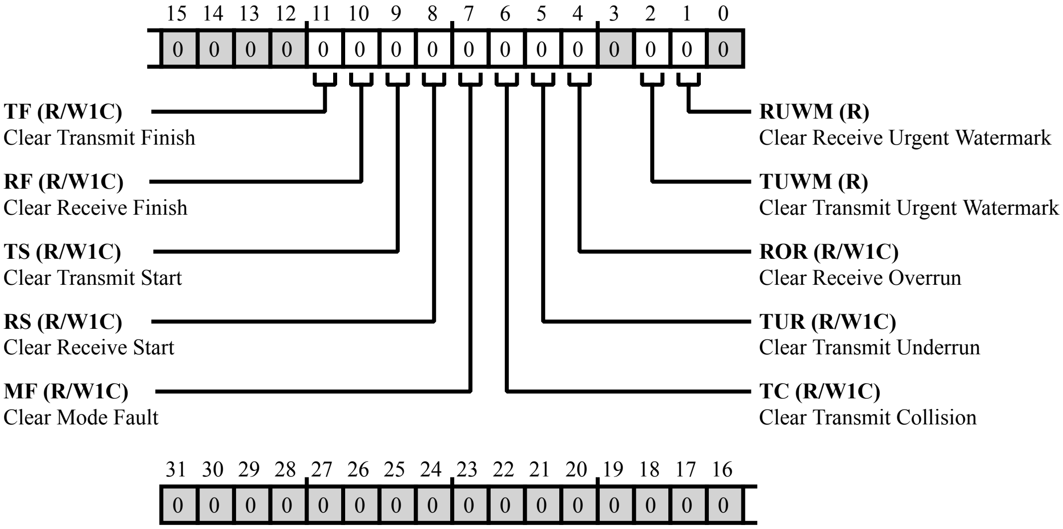

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TF (R/W1C)
Clear Transmit Finish
RF (R/W1C)
Clear Receive Finish
TS (R/W1C)
Clear Transmit Start
RS (R/W1C)
Clear Receive Start
MF (R/W1C)
Clear Mode Fault
RUWM (R)
Clear Receive Urgent Watermark
TUWM (R)
Clear Transmit Urgent Watermark
ROR (R/W1C)
Clear Receive Overrun
TUR (R/W1C)
Clear Transmit Underrun
TC (R/W1C)
Clear Transmit Collision
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-24: SPI_ILAT_CLR Register Diagram

Table 16-22: SPI_ILAT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The `SPI_ILAT_CLR.TF` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The `SPI_ILAT_CLR.RF` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The `SPI_ILAT_CLR.TS` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit

Table 16-22: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The SPI_ILAT_CLR.RS bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The SPI_ILAT_CLR.MF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The SPI_ILAT_CLR.TC bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The SPI_ILAT_CLR.TUR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The SPI_ILAT_CLR.ROR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
2 (R/NW)	TUWM	Clear Transmit Urgent Watermark. The SPI_ILAT_CLR.TUWM bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit

Table 16-22: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	RUWM	Clear Receive Urgent Watermark. The `SPI_ILAT_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Register

The `SPI_IMSK` register unmasks (enables) or masks (disables) SPI interrupt requests. When a condition is indicated by a bit in the `SPI_STAT` register and the corresponding interrupt request is unmasked in `SPI_IMSK`, the SPI latches the interrupt request bit in the `SPI_ILAT` register, queuing the interrupt request for service.

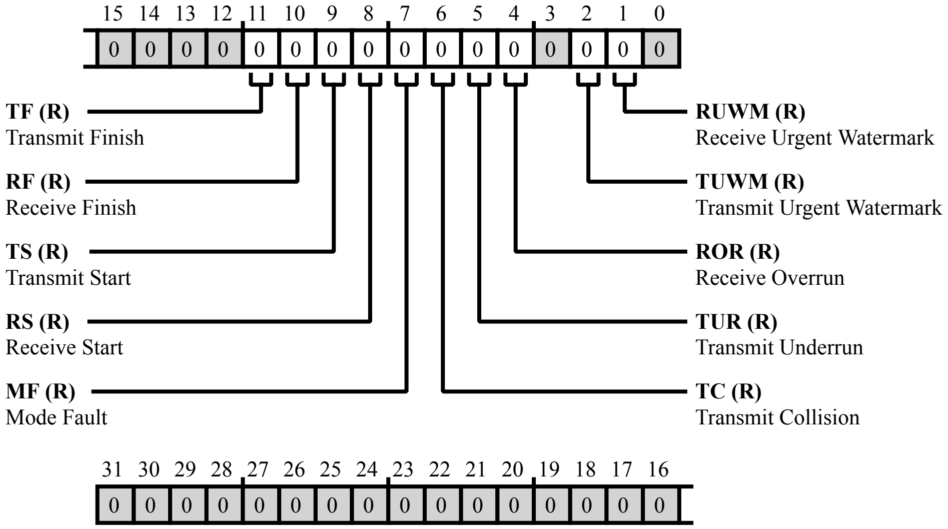

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TF (R)
Transmit Finish
RF (R)
Receive Finish
TS (R)
Transmit Start
RS (R)
Receive Start
MF (R)
Mode Fault
RUWM (R)
Receive Urgent Watermark
TUWM (R)
Transmit Urgent Watermark
ROR (R)
Receive Overrun
TUR (R)
Transmit Underrun
TC (R)
Transmit Collision
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-25: SPI_IMSK Register Diagram

Table 16-23: SPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish. The `SPI_IMSK.TF` bit unmasks (enables) or masks (disables) the TF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
10 (R/NW)	RF	Receive Finish. The `SPI_IMSK.RF` bit unmasks (enables) or masks (disables) the RF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
9 (R/NW)	TS	Transmit Start. The `SPI_IMSK.TS` bit unmasks (enables) or masks (disables) the TS interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request

Table 16-23: SPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RS	Receive Start. The `SPI_IMSK.RS` bit unmasks (enables) or masks (disables) the RS interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
7 (R/NW)	MF	Mode Fault. The `SPI_IMSK.MF` bit unmasks (enables) or masks (disables) the MF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
6 (R/NW)	TC	Transmit Collision. The `SPI_IMSK.TC` bit unmasks (enables) or masks (disables) the TC interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
5 (R/NW)	TUR	Transmit Underrun. The `SPI_IMSK.TUR` bit unmasks (enables) or masks (disables) the TUR interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
4 (R/NW)	ROR	Receive Overrun. The `SPI_IMSK.ROR` bit unmasks (enables) or masks (disables) the ROR interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark. The `SPI_IMSK.TUWM` bit unmasks (enables) or masks (disables) the TUWM interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark. The `SPI_IMSK.RUWM` bit unmasks (enables) or masks (disables) the RUWM interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request

Interrupt Mask Clear Register

The SPI_IMSK_CLR register permits clearing individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-clear on a bit in the SPI_IMSK_CLR register to clear the corresponding bit in the SPI_IMSK register.

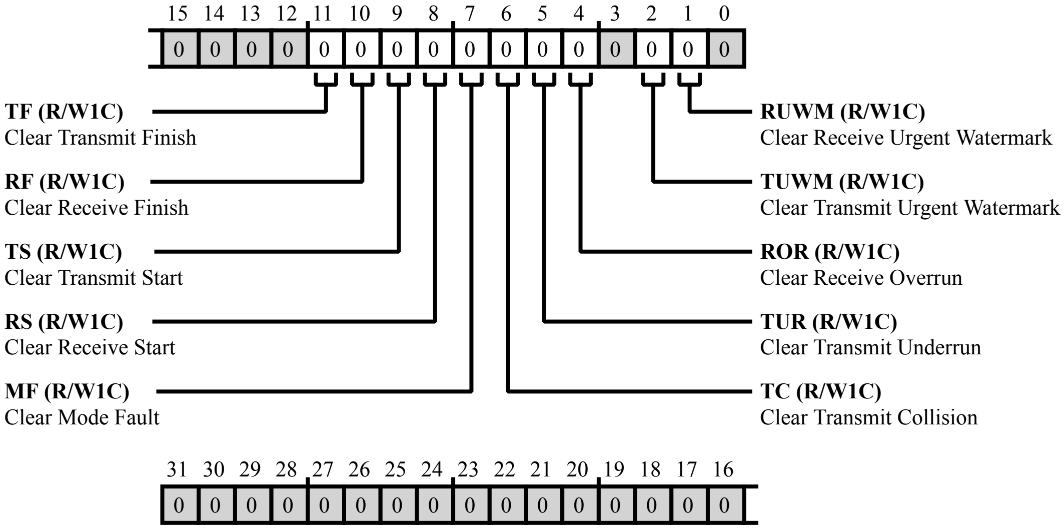

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TF (R/W1C)
Clear Transmit Finish
RF (R/W1C)
Clear Receive Finish
TS (R/W1C)
Clear Transmit Start
RS (R/W1C)
Clear Receive Start
MF (R/W1C)
Clear Mode Fault
RUWM (R/W1C)
Clear Receive Urgent Watermark
TUWM (R/W1C)
Clear Transmit Urgent Watermark
ROR (R/W1C)
Clear Receive Overrun
TUR (R/W1C)
Clear Transmit Underrun
TC (R/W1C)
Clear Transmit Collision
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-26: SPI_IMSK_CLR Register Diagram

Table 16-24: SPI_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The SPI_IMSK_CLR.TF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The SPI_IMSK_CLR.RF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The SPI_IMSK_CLR.TS bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit

Table 16-24: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The `SPI_IMSK_CLR.RS` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The `SPI_IMSK_CLR.MF` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The `SPI_IMSK_CLR.TC` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The `SPI_IMSK_CLR.TUR` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The `SPI_IMSK_CLR.ROR` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit
2 (R/W1C)	TUWM	Clear Transmit Urgent Watermark. The `SPI_IMSK_CLR.TUWM` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit

Table 16-24: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RUWM	Clear Receive Urgent Watermark. The `SPI_IMSK_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Set Register

The SPI_IMSK_SET register permits setting individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-set on a bit in the SPI_IMSK_SET register to set the corresponding bit in the SPI_IMSK register.

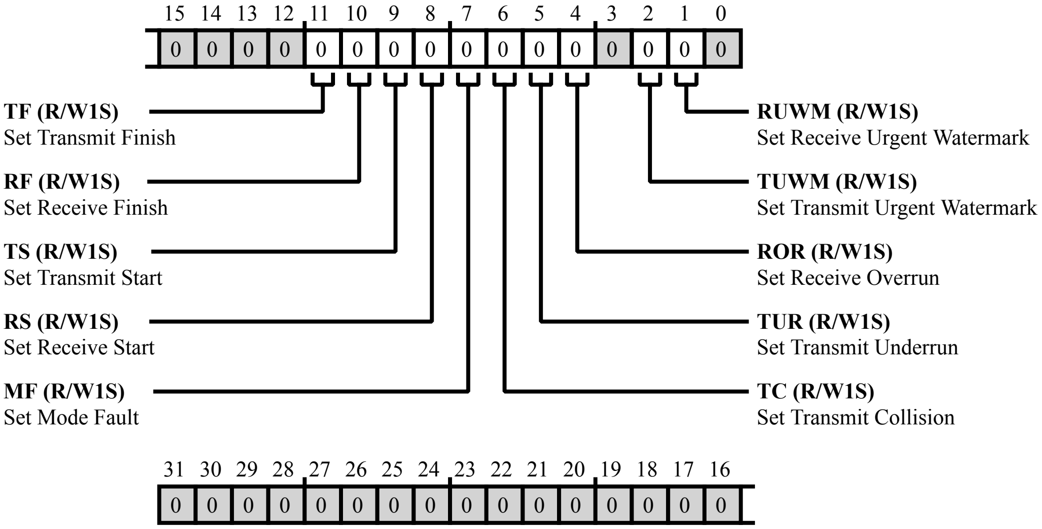

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TF (R/W1S)
Set Transmit Finish
RF (R/W1S)
Set Receive Finish
TS (R/W1S)
Set Transmit Start
RS (R/W1S)
Set Receive Start
MF (R/W1S)
Set Mode Fault
RUWM (R/W1S)
Set Receive Urgent Watermark
TUWM (R/W1S)
Set Transmit Urgent Watermark
ROR (R/W1S)
Set Receive Overrun
TUR (R/W1S)
Set Transmit Underrun
TC (R/W1S)
Set Transmit Collision
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-27: SPI_IMSK_SET Register Diagram

Table 16-25: SPI_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1S)	TF	Set Transmit Finish. The SPI_IMSK_SET.TF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
10 (R/W1S)	RF	Set Receive Finish. The SPI_IMSK_SET.RF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
9 (R/W1S)	TS	Set Transmit Start. The SPI_IMSK_SET.TS bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit

Table 16-25: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1S)	RS	Set Receive Start. The `SPI_IMSK_SET.RS` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit
7 (R/W1S)	MF	Set Mode Fault. The `SPI_IMSK_SET.MF` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit
6 (R/W1S)	TC	Set Transmit Collision. The `SPI_IMSK_SET.TC` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit
5 (R/W1S)	TUR	Set Transmit Underrun. The `SPI_IMSK_SET.TUR` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit
4 (R/W1S)	ROR	Set Receive Overrun. The `SPI_IMSK_SET.ROR` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit
2 (R/W1S)	TUWM	Set Transmit Urgent Watermark. The `SPI_IMSK_SET.TUWM` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit

Table 16-25: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	RUWM	Set Receive Urgent Watermark. The `SPI_IMSK_SET.RUWM` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit

Memory Mapped Read Header (Only on SPI2)

The `SPI_MMRDH` register enables the use of memory-mapped mode. This mode allows direct memory-mapped read accesses of an SPI memory device and is primarily used to directly execute instructions from an SPI FLASH memory without using a low-level software driver. All overhead tasks such as transmission of the read header, pin turnaround timing and receive data sizing are handled in hardware.

The memory-mapped access mode is enabled by setting the `SPI_CTL.MMSE` bit. The features within the `SPI_MMRDH` register include a command skip mode, variable length byte addressing, and independent multi-pin support for command transmission, address transmission and data reception. In addition, the command opcode and mode bytes are fully programmable.

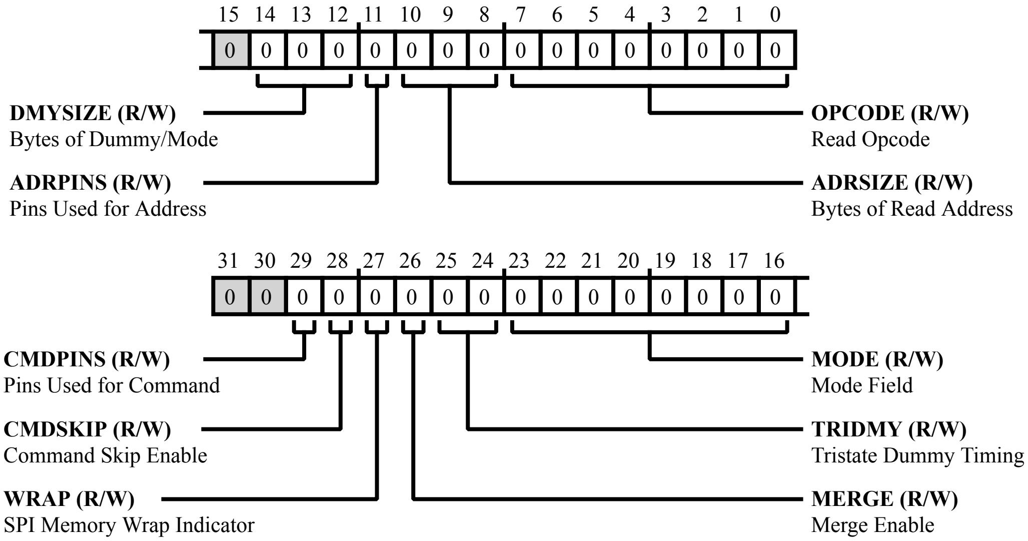

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMYSIZE (R/W)
Bytes of Dummy/Mode
ADRPINS (R/W)
Pins Used for Address
OPCODE (R/W)
Read Opcode
ADRSIZE (R/W)
Bytes of Read Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CMDPINS (R/W)
Pins Used for Command
CMDSKIP (R/W)
Command Skip Enable
WRAP (R/W)
SPI Memory Wrap Indicator
MODE (R/W)
Mode Field
TRIDMY (R/W)
Tristate Dummy Timing
MERGE (R/W)
Merge Enable

Figure 16-28: SPI_MMRDH Register Diagram

Table 16-26: SPI_MMRDH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	CMDPINS	Pins Used for Command. The `SPI_MMRDH.CMDPINS` bit specifies the number of pins to be used for command transmission. This bit must be set consistent with the expectations established by the read opcode. Hardware does not interpret `SPI_MMRDH.OPCODE`, but rather relies on this bit to specify behavior. When cleared, it overrides the `SPI_CTL.MIOM` bits. When set, it uses bits specified by the `SPI_CTL.MIOM` bit setting.	
		0	Use only one pin: MOSI (overrides `SPI_CTL.MIOM` bits)
		1	Use pins specified by `SPI_CTL.MIOM` bits

Table 16-26: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	CMDSKIP	Command Skip Enable. The `SPI_MMRDH.CMDSKIP` bit enables command skip mode where the address is sent first and the OPCODE field is not sent (`SPI_MMRDH.CMDSKIP` bit =1). This mode is useful for supporting XIP (Execute-In-Place) operation where only the address is sent and the same read command is assumed. The SPI flash device must be primed with an initial read command, before the `SPI_MMRDH.CMDSKIP` bit is set.	
		0	OPCODE field is sent first followed by address
		1	OPCODE field is not sent; address is sent first
27 (R/W)	WRAP	SPI Memory Wrap Indicator. The `SPI_MMRDH.WRAP` bit must be set by software if software places a connected SPI memory device into a 8-byte, 16-byte or 32-byte wrap mode based on the ILINE and DLINE field setting of the cache configuration register address wrap mode. Software achieves this by transmitting a vendor specified command to the SPI memory device while the `SPI_CTL.MMSE` bit =0. If the `SPI_MMRDH.WRAP` bit =1, the SPI does not need to deassert the SPI slave select signal and resend the read header in order to wrap to the cache line base when servicing misaligned cache fill requests. Although this improves cache fill efficiency, it requires that the SPI deassert the SPI slave select pin and resend the read header whenever a DMA burst requests crosses 32 byte alignments. Setting this bit improves cache throughput but decreases DMA throughput.	
		0	SPI Memory auto increments address purely sequentially
		1	SPI Memory auto increments address but wraps within 32 Byte lines
26 (R/W)	MERGE	Merge Enable. When the `SPI_MMRDH.MERGE` bit is set, SPI hardware combines the two successive transfers. This increases the throughput rate when accessing a large number of sequential memory locations. For more information refer to the Merged Read Accesses section.	

Table 16-26: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25:24 (R/W)	TRIDMY	Tristate Dummy Timing. The `SPI_MMRDH.TRIDMY` bits specify whether and when output pins are three-stated during the interval of time specified by the `SPI_MMRDH.DMYSIZE` bits. Output pins potentially three-stated include all pins which were used to transmit the address	
		0	Tristate outputs immediately
		1	Tristate outputs after 4 bits of dummy/mode are transmitted
		2	Tristate outputs after 8 bits of dummy/mode are transmitted
		3	Never tristate outputs (previously specified output state is held)
23:16 (R/W)	MODE	Mode Field. These bits specify up to a leading byte to be transmitted during the interval of time specified by the `SPI_MMRDH.DMYSIZE` bit field. This first byte, or a portion of it, is interpreted as mode bits when certain opcodes are used in conjunction with certain SPI memory devices. Mode bits are sent using the same number of pins which were used to transmit the address. Once sent, output pins will be held in their final resultant state until the conclusion of all dummy byte periods, unless three-stating the outputs is specified first by the `SPI_MMRDH.TRIDMY` bits.	
14:12 (R/W)	DMYSIZE	Bytes of Dummy/Mode. The `SPI_MMRDH.DMYSIZE` bit field specifies the number of bytes separating address transmission and read data return. Dummy bytes elapse assuming dummy bits are transmitted using the same number of pins which were used to transmit address.	
		0	0 Bytes
		1	1 Bytes
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes
		5	5 Bytes
		6	6 Bytes
		7	7 Bytes

Table 16-26: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ADRPINS	Pins Used for Address. The `SPI_MMRDH.ADRPINS` bit specifies the number of pins to be used for address transmission. This bit must be set consistent with expectations established by read opcode. Hardware does not interpret the `SPI_MMRDH.OPCODE`, but rather relies on this bit to specify behavior.	
		0	Use only one pin: MOSI (overrides `SPI_CTL.MIOM` bits)
		1	Use pins specified by `SPI_CTL.MIOM` bits
10:8 (R/W)	ADRSIZE	Bytes of Read Address. The `SPI_MMRDH.ADRSIZE` bit field defines the number of bytes used to specify the read address. The read address is sent immediately following the transmission of opcode. Unlike opcode bits, address bits may be sent using either one or multiple pins. The number of pins is selected using the `SPI_MMRDH.ADRPINS` bit. The address sent to a connected SPI memory device is an echo of the read address received by the SPI peripheral slave port. The least significant bytes of address are sent when the entire address is not sent.	
		0	1 Byte
		1	1 Byte
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes
7:0 (R/W)	OPCODE	Read Opcode. The `SPI_MMRDH.OPCODE` bit field specifies the initial bits transmitted in response to a read request of SPI memory. Although any opcode may be sent, values 0x03, 0x0B, 0x3B, 0x6B, 0xBB, and 0xEB are likely to be the most commonly used. `SPI_MMRDH.OPCODE` is sent by the SPI without interpretation; the states of these bits have no effect beyond specifying what is initially shifted across the SPI interface.	

SPI Memory Top Address (Only on SPI2)

The `SPI_MMTOP` register specifies the top populated address of a connected SPI memory device.

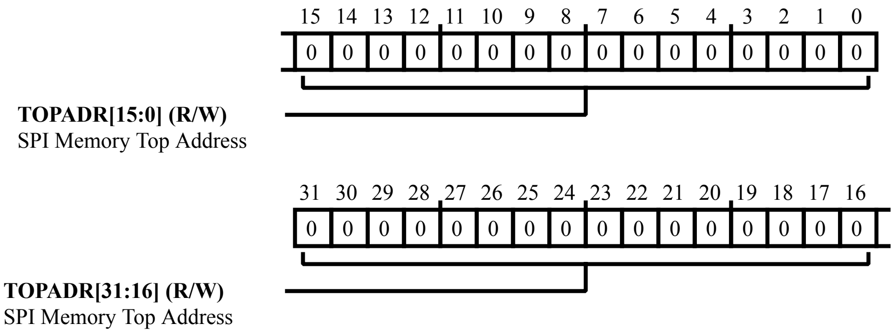

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TOPADR[15:0] (R/W)
SPI Memory Top Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TOPADR[31:16] (R/W)
SPI Memory Top Address

Figure 16-29: SPI_MMTOP Register Diagram

Table 16-27: SPI_MMTOP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TOPADR	SPI Memory Top Address. The `SPI_MMTOP.TOPADR` bit field specifies the top populated address of a connected SPI memory device. Attempts to access SPI memory are not blocked if this address is exceeded and an error is generated as part of the read response.

Receive FIFO Data Register

The SPI_RFIFO register has an interface to the receive shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit SPI_RFIFO register, but the size (number of word locations) of the receive FIFO is actually flexible with transfer word size. The size of the receive FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall receive transfers based on FIFO status. When the receive FIFO is full, the SPI master stops initiating new transfers on the SPI if `SPI_RXCTL.RTI` is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data reception continues after SPI_RFIFO is full, the data in the receive FIFO is invalid. The SPI indicates this condition with receive overrun (`SPI_STAT.ROR`) error. This condition is possible when `SPI_RXCTL.RTI` =0 and `SPI_RXCTL.REN` =1 for a master, or for a slave that does not exercise flow control.

Note that the receive FIFO is reset (cleared) when the SPI is disabled after being enabled.

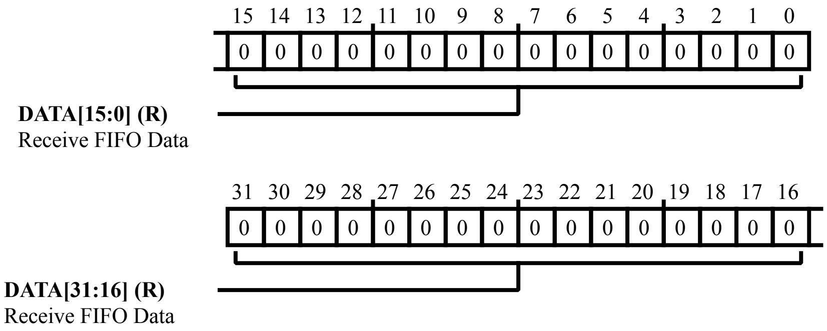


Figure 16-30: SPI_RFIFO Register Diagram

Table 16-28: SPI_RFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Receive FIFO Data. The `SPI_RFIFO.DATA` bit field contains the FIFO receive data.

Received Word Count Register

The `SPI_RWC` register holds a count of the number of words remaining to be received by the SPI. To start the decrement of the word count in `SPI_RWC`, enable the receive word counter (`SPI_RXCTL.RWCEN` =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the receive finish interrupt (`SPI_ILAT.RF`). In DMA mode, the SPI uses the `SPI_RWC` register to ensure that the number of frames received during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the `SPI_RWC` registers should match the word count in the DMA configuration. The `SPI_RWC` register maintains the number of frames to be received in a transfer. The `SPI_RWC` should only be changed when the counter is disabled.

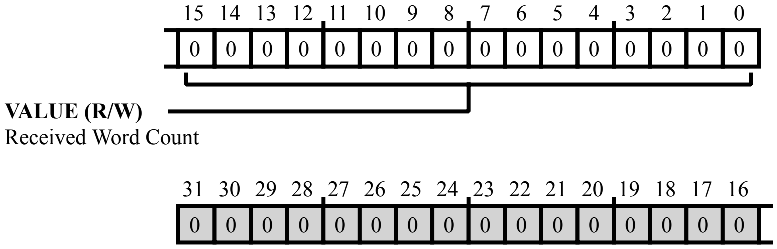

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Received Word Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-31: SPI_RWC Register Diagram

Table 16-29: SPI_RWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count. The `SPI_RWC.VALUE` bits hold the receive transfer word count.

Received Word Count Reload Register

The `SPI_RWCR` register holds the receive word count value that the SPI loads into the `SPI_RWC` register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The `SPI_RWCR` register should only be changed when the counter is disabled.

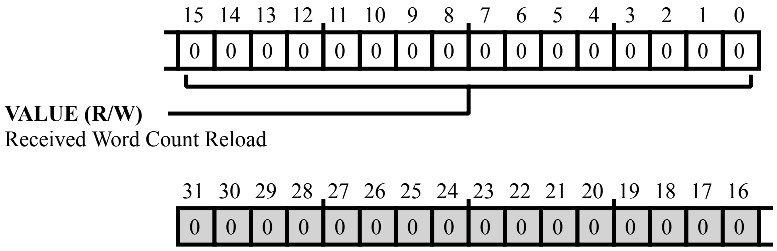

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Received Word Count Reload
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

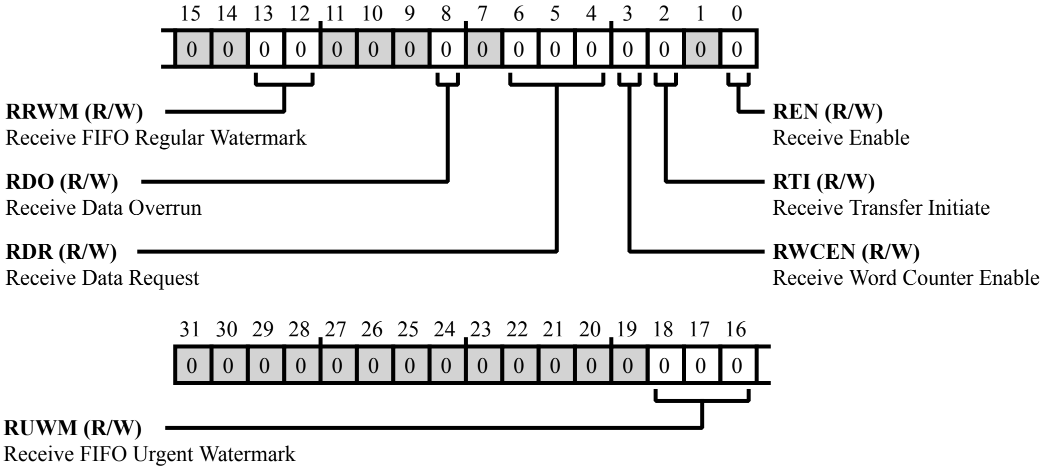
Figure 16-32: SPI_RWCR Register Diagram

Table 16-30: SPI_RWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count Reload. The `SPI_RWCR.VALUE` bits hold the receive transfer word count reload value.

Receive Control Register

The SPI_RXCTL register enables the SPI receive channel, initiates receive transfers, and configures SPI_RFIFO buffer watermark settings.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RRWM (R/W)
Receive FIFO Regular Watermark
RDO (R/W)
Receive Data Overrun
RDR (R/W)
Receive Data Request
REN (R/W)
Receive Enable
RTI (R/W)
Receive Transfer Initiate
RWCEN (R/W)
Receive Word Counter Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RUWM (R/W)
Receive FIFO Urgent Watermark

Figure 16-33: SPI_RXCTL Register Diagram

Table 16-31: SPI_RXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RUWM	Receive FIFO Urgent Watermark. The SPI_RXCTL.RUWM bits select the receive FIFO (SPI_RFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.RUWM interrupt. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the deassertion condition for any SPI_ILAT.RUWM interrupts that are latched.	
		0	Disabled
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
		5	Reserved
		6	Reserved
		7	Reserved

Table 16-31: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	RRWM	Receive FIFO Regular Watermark. The `SPI_RXCTL.RRWM` bits select the receive FIFO (`SPI_RFIFO`) watermark level for regular data bus requests. When an urgent `SPI_RFIFO` watermark is enabled with `SPI_RXCTL.RUWM`, the `SPI_RXCTL.RRWM` selection is used as the deassertion condition for any `SPI_ILAT.RUWM` interrupts that are latched.	
		0	Empty RFIFO
		1	RFIFO less than 25% full
		2	RFIFO less than 50% full
		3	RFIFO less than 75% full
8 (R/W)	RDO	Receive Data Overrun. The `SPI_RXCTL.RDO` bit selects handling for receive data requests when the receive buffer (`SPI_RFIFO`) is full. If enabled and `SPI_RFIFO` is full, the SPI overwrites old data in the buffer with incoming data. If disabled and `SPI_RFIFO` is full, the SPI keeps old data in the buffer and discards incoming data.	
		0	Discard incoming data if SPI_RFIFO is full
		1	Overwrite old data if SPI_RFIFO is full
6:4 (R/W)	RDR	Receive Data Request. The `SPI_RXCTL.RDR` bits select receive FIFO (`SPI_RFIFO`) watermark conditions that direct the SPI to generate a receive data request.	
		0	Disabled
		1	Not empty RFIFO
		2	25% full RFIFO
		3	50% full RFIFO
		4	75% full RFIFO
		5	Full RFIFO
		6	Reserved
		7	Reserved
3 (R/W)	RWCEN	Receive Word Counter Enable. The `SPI_RXCTL.RWCEN` bit enables the decrement of the `SPI_RWC` register when the count is not zero and `SPI_RXCTL.RTI` is enabled. Enabling `SPI_RXCTL.RWCEN` prevents receive overrun errors from occurring. The `SPI_RXCTL.RWCEN` bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 16-31: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	RTI	Receive Transfer Initiate. The `SPI_RXCTL.RTI` bit enables initiation of receive transfers if the receive FIFO (`SPI_RFIFO`) is not full. The bit also enables this initiation if `SPI_RWC` is not zero when `SPI_RXCTL.RWCEN` is enabled. Enabling `SPI_RXCTL.RTI` prevents receive overrun errors from occurring. The `SPI_RXCTL.RTI` bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	REN	Receive Enable. The `SPI_RXCTL.REN` bit enables SPI receive channel operation.	
		0	Disable
		1	Enable

Slave Select Register

The SPI_SLVSEL register enables the SPI_SEL[n] pins for output and indicates the state (high or low) of these pins when enabled.

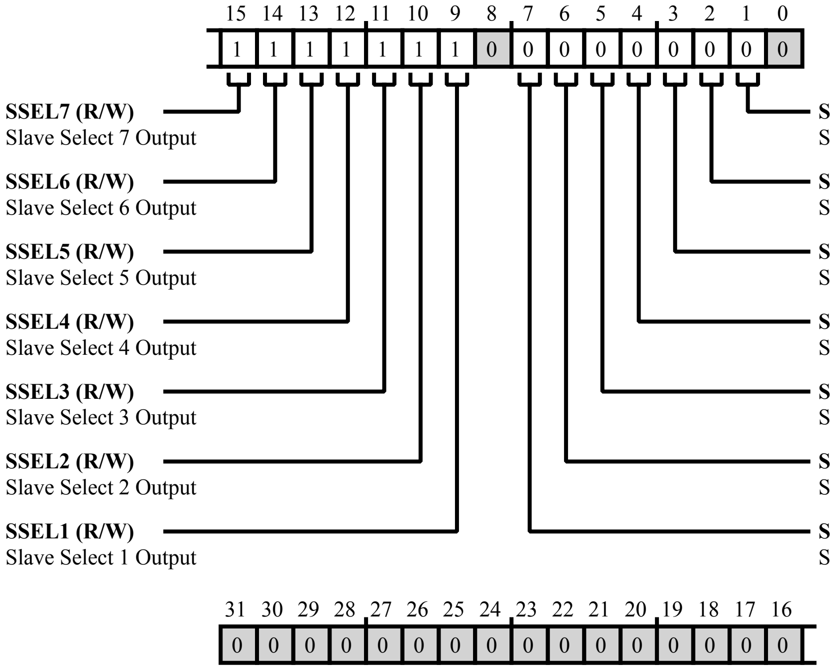

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
SSEL7 (R/W)
Slave Select 7 Output
SSEL6 (R/W)
Slave Select 6 Output
SSEL5 (R/W)
Slave Select 5 Output
SSEL4 (R/W)
Slave Select 4 Output
SSEL3 (R/W)
Slave Select 3 Output
SSEL2 (R/W)
Slave Select 2 Output
SSEL1 (R/W)
Slave Select 1 Output
SSE1 (R/W)
Slave Select 1 Enable
SSE2 (R/W)
Slave Select 2 Enable
SSE3 (R/W)
Slave Select 3 Enable
SSE4 (R/W)
Slave Select 4 Enable
SSE5 (R/W)
Slave Select 5 Enable
SSE6 (R/W)
Slave Select 6 Enable
SSE7 (R/W)
Slave Select 7 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-34: SPI_SLVSEL Register Diagram

Table 16-32: SPI_SLVSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SSEL7	Slave Select 7 Output. The SPI_SLVSEL.SSEL7 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
14 (R/W)	SSEL6	Slave Select 6 Output. The SPI_SLVSEL.SSEL6 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High

Table 16-32: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	SSEL5	Slave Select 5 Output. The `SPI_SLVSEL.SSEL5` bit state indicates the value driven on the related `SPI_SEL[n]` pin.	
		0	Low
		1	High
12 (R/W)	SSEL4	Slave Select 4 Output. The `SPI_SLVSEL.SSEL4` bit state indicates the value driven on the related `SPI_SEL[n]` pin.	
		0	Low
		1	High
11 (R/W)	SSEL3	Slave Select 3 Output. The `SPI_SLVSEL.SSEL3` bit state indicates the value driven on the related `SPI_SEL[n]` pin.	
		0	Low
		1	High
10 (R/W)	SSEL2	Slave Select 2 Output. The `SPI_SLVSEL.SSEL2` bit state indicates the value driven on the related `SPI_SEL[n]` pin.	
		0	Low
		1	High
9 (R/W)	SSEL1	Slave Select 1 Output. The `SPI_SLVSEL.SSEL1` bit state indicates the value driven on the related `SPI_SEL[n]` pin.	
		0	Low
		1	High
7 (R/W)	SSE7	Slave Select 7 Enable. The `SPI_SLVSEL.SSE7` bit enables the related `SPI_SEL[n]` pin for output. If disabled, the SPI three-states the related `SPI_SEL[n]` pin. When the SPI is a slave, the master (not the SPI) asserts the input during the transfer. The input may be deasserted or remain asserted between transfers. While the input is deasserted, the SPI ignores the SPI clock, ignores inputs, and three-states outputs.	
		0	Disable
		1	Enable

Table 16-32: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SSE6	Slave Select 6 Enable. The `SPI_SLVSEL.SSE6` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable
5 (R/W)	SSE5	Slave Select 5 Enable. The `SPI_SLVSEL.SSE5` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable
4 (R/W)	SSE4	Slave Select 4 Enable. The `SPI_SLVSEL.SSE4` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable
3 (R/W)	SSE3	Slave Select 3 Enable. The `SPI_SLVSEL.SSE3` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable
2 (R/W)	SSE2	Slave Select 2 Enable. The `SPI_SLVSEL.SSE2` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable
1 (R/W)	SSE1	Slave Select 1 Enable. The `SPI_SLVSEL.SSE1` bit enables the related $\overline{\texttt{SPI\_SEL[n]}}$ pin for output. See the `SPI_SLVSEL.SSE7` bit description for more information.	
		0	Disable
		1	Enable

Status Register

The `SPI_STAT` register indicates SPI status including FIFO status, error conditions, and interrupt conditions. When an interrupt condition from this register is unmasked (enabled) by the corresponding bit in the `SPI_IMSK` register, the interrupt request is latched into the corresponding bit in the `SPI_ILAT` register.

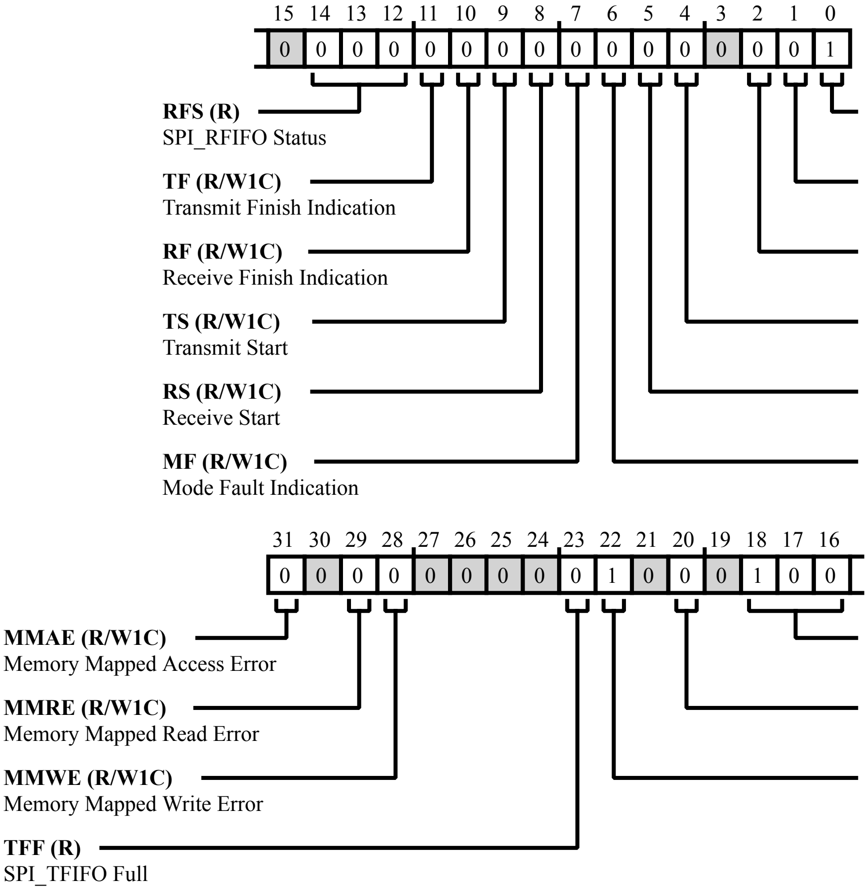

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
RFS (R)
SPI_RFIFO Status
TF (R/W1C)
Transmit Finish Indication
RF (R/W1C)
Receive Finish Indication
TS (R/W1C)
Transmit Start
RS (R/W1C)
Receive Start
MF (R/W1C)
Mode Fault Indication
SPIF (R)
SPI Finished
RUWM (R)
Receive Urgent Watermark Breached
TUWM (R)
Transmit Urgent Watermark Breached
ROR (R/W1C)
Receive Overrun Indication
TUR (R/W1C)
Transmit Underrun Indication
TC (R/W1C)
Transmit Collision Indication
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
MMAE (R/W1C)
Memory Mapped Access Error
MMRE (R/W1C)
Memory Mapped Read Error
MMWE (R/W1C)
Memory Mapped Write Error
TFF (R)
SPI_TFIFO Full
TFS (R)
SPI_TFIFO Status
FCS (R)
Flow Control Stall Indication
RFE (R)
SPI_RFIFO Empty

Figure 16-35: SPI_STAT Register Diagram

Table 16-33: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	MMAE	Memory Mapped Access Error. The `SPI_STAT.MMAE` bit =1 if an attempt is made to access either the Tx or Rx FIFO while memory-mapped access of SPI memory is enabled (see the `SPI_CTL.MMSE` bit). The `SPI_STAT.MMAE` bit =0 when a 1 is written to it. The `SPI_STAT.MMAE` bit is provided for software notification only. Its state has no further effect.

Table 16-33: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W1C)	MMRE	Memory Mapped Read Error. The `SPI_STAT.MMRE` bit =1 if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (see the `SPI_CTL.MMSE` bit). The `SPI_STAT.MMRE` bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
28 (R/W1C)	MMWE	Memory Mapped Write Error. The `SPI_STAT.MMWE` bit =1 if an attempt is made to write address space reserved for memory-mapped SPI memory. The `SPI_STAT.MMWE` bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
23 (R/NW)	TFF	SPI_TFIFO Full. The `SPI_STAT.TFF` bit indicates whether the `SPI_TFIFO` is full or not full.	
		0	Not full Tx FIFO
		1	Full Tx FIFO
22 (R/NW)	RFE	SPI_RFIFO Empty. The `SPI_STAT.RFE` bit indicates whether the `SPI_RFIFO` is empty or not empty.	
		0	Rx FIFO not empty
		1	Rx FIFO empty
20 (R/NW)	FCS	Flow Control Stall Indication. The `SPI_STAT.FCS` bit indicates whether a slave has deasserted the `SPI_RDY` pin to stall the SPI master while the slave is unable to service the SPI masters request. This bit is valid only when the SPI is a master (`SPI_CTL.MSTR` =1) and flow control is enabled (`SPI_CTL.FCEN` =1).	
		0	No Stall (RDY pin asserted)
		1	Stall (RDY pin deasserted)
18:16 (R/NW)	TFS	SPI_TFIFO Status. The `SPI_STAT.TFS` bits indicate the status of the `SPI_TFIFO`. The SPI uses this status when evaluating transmit watermark conditions.	
		0	Full TFIFO
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 16-33: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/NW)	RFS	SPI_RFIFO Status. The `SPI_STAT.RFS` bits indicate the status of the `SPI_RFIFO`. The SPI uses this status when evaluating receive watermark conditions.	
		0	Empty RFIFO
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
11 (R/W1C)	TF	Transmit Finish Indication. The `SPI_STAT.TF` bit indicates that the SPI has detected the finish of a transmit burst transfer (the `SPI_TWC` count decrements to zero). This condition can only occur when `SPI_TXCTL.TTI` and `SPI_TXCTL.TWCEN` are enabled.	
		0	No status
		1	Transmit finish detected
10 (R/W1C)	RF	Receive Finish Indication. The `SPI_STAT.RF` bit indicates that the SPI has detected the finish of a receive burst transfer (the `SPI_RWC` count decrements to zero). This condition can only occur when `SPI_RXCTL.RTI` and `SPI_RXCTL.RWCEN` are enabled.	
		0	No status
		1	Receive finish detected
9 (R/W1C)	TS	Transmit Start. The `SPI_STAT.TS` bit indicates that the SPI has detected the start of a transmit burst transfer. A transmit bursts starts with the load of `SPI_TWC` from the `SPI_TWCR`. This condition can only occur when `SPI_TXCTL.TTI` and `SPI_TXCTL.TWCEN` are enabled.	
		0	No status
		1	Transmit start detected
8 (R/W1C)	RS	Receive Start. The `SPI_STAT.RS` bit indicates that the SPI has detected the start of a receive burst transfer. A receive bursts starts with the load of `SPI_RWC` from the `SPI_RWCR`. This condition can only occur when `SPI_RXCTL.RTI` and `SPI_RXCTL.RWCEN` are enabled.	
		0	No status
		1	Receive start detected

Table 16-33: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	MF	Mode Fault Indication. The `SPI_STAT.MF` bit, when SPI is a master and `SPI_CTL.PSSE` is enabled, indicates that multiple masters have asserted slave select inputs.	
		0	No status
		1	Mode fault occurred
6 (R/W1C)	TC	Transmit Collision Indication. The `SPI_STAT.TC` bit, when SPI is a slave, indicates that the load of data into the shift register has occurred too close to the first transmitting edge of the SPI clock.	
		0	No status
		1	Transmit collision occurred
5 (R/W1C)	TUR	Transmit Underrun Indication. The `SPI_STAT.TUR` bit, when the transmit FIFO (`SPI_TFIFO`) is empty, indicates that the last word in the transmit FIFO has been re-sent as transmit data. Alternately, it indicates that zero has been sent as transmit data.	
		0	No status
		1	Transmit underrun occurred
4 (R/W1C)	ROR	Receive Overrun Indication. The `SPI_STAT.ROR` bit, when the receive FIFO (`SPI_RFIFO`) is full, indicates that a word in the receive FIFO has been overwritten with incoming receive data. Alternately, it indicates that incoming receive data has been discarded.	
		0	No status
		1	Receive overrun occurred
2 (R/NW)	TUWM	Transmit Urgent Watermark Breached. The `SPI_STAT.TUWM` bit indicates that the transmit urgent watermark (`SPI_TXCTL.TUWM`) has been reached. This condition is cleared when the transmit FIFO fills enough to reach the transmit regular watermark (`SPI_TXCTL.TRWM`).	
		0	Tx regular watermark reached
		1	Tx urgent watermark breached
1 (R/NW)	RUWM	Receive Urgent Watermark Breached. The `SPI_STAT.RUWM` bit indicates that the receive urgent watermark (`SPI_RXCTL.RUWM`) has been reached. This condition is cleared when the receive FIFO empties enough to reach the receive regular watermark (`SPI_RXCTL.RRWM`).	
		0	Rx regular watermark reached
		1	Rx urgent watermark breached

Table 16-33: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SPIF	SPI Finished. The `SPI_STAT.SPIF` bit indicates that a single word transfer is complete.	
		0	No status
		1	Completed single word transfer

Transmit FIFO Data Register

The `SPI_TFIFO` register has an interface to the transmit shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit `SPI_TFIFO` register, but the size (number of word locations) of the transmit FIFO is actually flexible with transfer word size. The size of the transmit FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall transmit transfers based on FIFO status. When the transmit FIFO is empty, the SPI master stops initiating new transfers on the SPI if `SPI_TXCTL.TTI` is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data transmit requests continue after `SPI_TFIFO` is empty, the data sent from the transmit FIFO is invalid, and the SPI indicates this condition with transmit underrun (`SPI_STAT.TUR`). This condition is possible when `SPI_TXCTL.TTI` =0 and `SPI_TXCTL.TEN` =1 for a master, or for a slave that does not exercise flow control.

Note that the transmit FIFO is reset (cleared) when the SPI is disabled after being enabled.

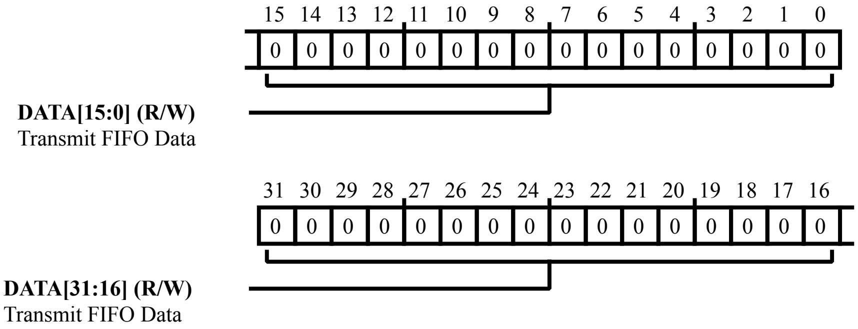

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
Transmit FIFO Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
Transmit FIFO Data

Figure 16-36: SPI_TFIFO Register Diagram

Table 16-34: SPI_TFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit FIFO Data. The `SPI_TFIFO.DATA` bit field contains the FIFO transmit data.

Transmitted Word Count Register

The `SPI_TWC` register holds a count of the number of words remaining to be transmitted by the SPI. To start the decrement of the word count in `SPI_TWC`, enable the transmit word counter (`SPI_TXCTL.TWCEN` =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the transmit finish interrupt request. In DMA mode, the SPI uses the `SPI_TWC` to ensure that the number of frames transmitted during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the `SPI_TWC` registers should match the word count in the DMA configuration. The `SPI_TWC` maintains the number of frames to be transmitted in a transfer. The `SPI_TWC` should only be changed when the counter is disabled.

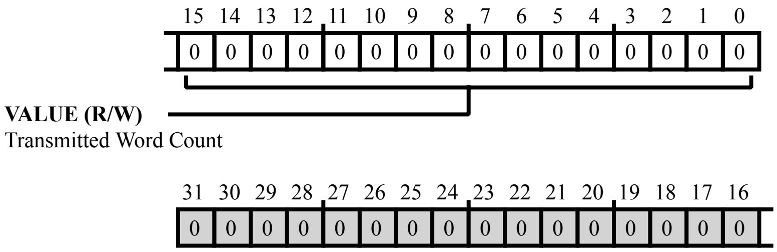

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Transmitted Word Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-37: SPI_TWC Register Diagram

Table 16-35: SPI_TWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count. The `SPI_TWC.VALUE` bits hold the transmit transfer word count.

Transmitted Word Count Reload Register

The `SPI_TWCR` register holds the transmit word count value that the SPI loads into the `SPI_TWC` register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The `SPI_TWCR` should only be changed when the counter is disabled.

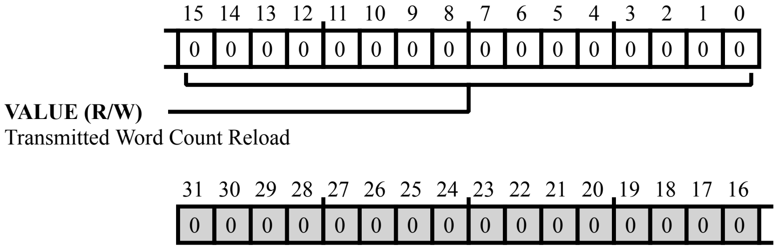

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Transmitted Word Count Reload
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-38: SPI_TWCR Register Diagram

Table 16-36: SPI_TWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count Reload. The `SPI_TWCR.VALUE` bits hold the transmit transfer word count reload value.

Transmit Control Register

The `SPI_TXCTL` register enables the SPI transmit channel, initiates transmit transfers, and configures `SPI_TFIFO` buffer watermark settings.

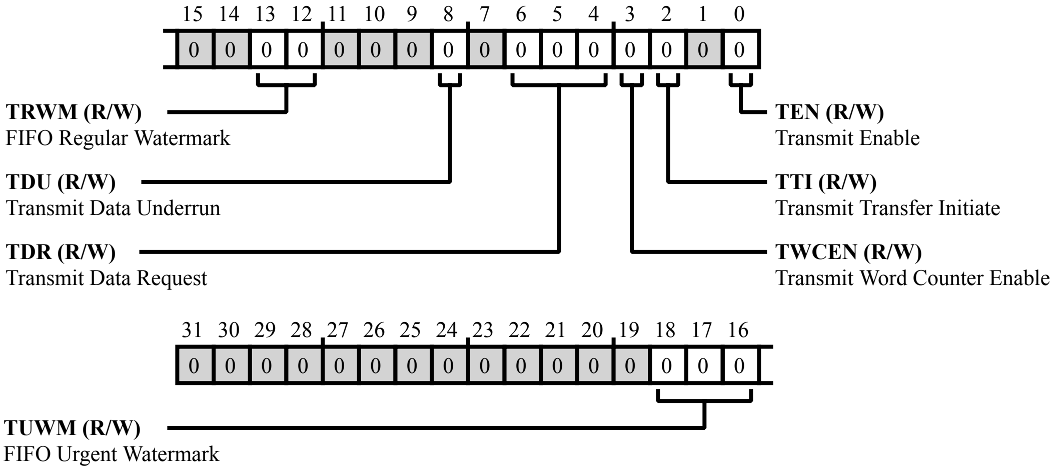

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRWM (R/W)
FIFO Regular Watermark
TDU (R/W)
Transmit Data Underrun
TDR (R/W)
Transmit Data Request
TEN (R/W)
Transmit Enable
TTI (R/W)
Transmit Transfer Initiate
TWCEN (R/W)
Transmit Word Counter Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TUWM (R/W)
FIFO Urgent Watermark

Figure 16-39: SPI_TXCTL Register Diagram

Table 16-37: SPI_TXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	TUWM	FIFO Urgent Watermark. The `SPI_TXCTL.TUWM` bits select the transmit FIFO (`SPI_TFIFO`) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the `SPI_ILAT.TUWM` interrupt request. When an urgent `SPI_TFIFO` watermark is enabled with `SPI_TXCTL.TUWM`, the `SPI_TXCTL.TRWM` selection is used as the deassertion condition for any `SPI_ILAT.TUWM` interrupt requests that are latched.	
		0	Disabled
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 16-37: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	TRWM	FIFO Regular Watermark. The `SPI_TXCTL.TRWM` bits select the transmit FIFO (`SPI_TFIFO`) watermark level for regular data bus requests. When an urgent `SPI_TFIFO` watermark is enabled with `SPI_TXCTL.TUWM`, the `SPI_TXCTL.TRWM` selection is used as the deassertion condition for any `SPI_ILAT.TUWM` interrupt requests that are latched.	
		0	Full TFIFO
		1	TFIFO less than 25% empty
		2	TFIFO less than 50% empty
		3	TFIFO less than 75% empty
8 (R/W)	TDU	Transmit Data Underrun. The `SPI_TXCTL.TDU` bit selects handling for transmit data requests when the transmit buffer (`SPI_TFIFO`) is empty. If enabled and `SPI_TFIFO` is empty, the SPI transmits zero as data. If disabled and `SPI_TFIFO` is empty, the SPI transmits the last word in the buffer as data.	
		0	Send last word when SPI_TFIFO is empty
		1	Send zeros when SPI_TFIFO is empty
6:4 (R/W)	TDR	Transmit Data Request. The `SPI_TXCTL.TDR` bits select transmit FIFO (`SPI_TFIFO`) watermark conditions that direct the SPI to generate a transmit status interrupt request.	
		0	Disabled
		1	Not full TFIFO
		2	25% empty TFIFO
		3	50% empty TFIFO
		4	75% empty TFIFO
		5	Empty TFIFO
3 (R/W)	TWCEN	Transmit Word Counter Enable. The `SPI_TXCTL.TWCEN` bit enables the decrement of the transmit word count (`SPI_TWC`) register when the count is not zero and `SPI_TXCTL.TTI` is enabled. Enabling `SPI_TXCTL.TWCEN` prevents transmit underrun errors from occurring. The `SPI_TXCTL.TWCEN` bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 16-37: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	TTI	Transmit Transfer Initiate. The `SPI_TXCTL.TTI` bit enables initiation of transmit transfers if the transmit FIFO (`SPI_TFIFO`) is not empty. The bit also enables this initiation if `SPI_TWC` is not zero when `SPI_TXCTL.TWCEN` is enabled. Enabling `SPI_TXCTL.TTI` prevents transmit underrun errors from occurring. The `SPI_TXCTL.TTI` bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	TEN	Transmit Enable. The `SPI_TXCTL.TEN` bit enables SPI transmit channel operation.	
		0	Disable
		1	Enable

17 Universal Asynchronous Receiver/Transmitter (UART)

The UART module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

The UART module supports partial modem status and control functionality to allow for hardware flow control.

The UART is a DMA-capable peripheral with separate transmit and receive DMA master channels. The use of DMA requires minimal software intervention as the DMA engine moves the data. The UART can also use a programmed core mode of operation. The core mode requires software management of the data flow using either interrupts or polling.

The UART can use one of the peripheral timers for a hardware-assisted auto-baud detection mechanism. The timers are external to the UART.

UART Features

Each UART includes the following features.

- 5–8 data bits
- Programmable extra stop bit and programmable extra half-stop bit
- Even, odd, and sticky parity bit options
- Extra 8-stage receive FIFO with programmable threshold interrupt request
- Flexible transmit and receive interrupt request timing
- Three interrupt outputs for receive, transmit, and status

- Independent DMA operation for receive and transmit
- Programmable automatic request to send (RTS)/clear to send (CTS) hardware flow control
- False start bit detection
- SIR IrDA operation mode
- MDB/ICP v2.0 operation mode
- Internal loopback
- Improved bit rate granularity
- LIN break command/Inter-frame gap transmission support

Table 17-1: UART Specifications

Feature	Availability
Protocol	
Master-Capable	Yes
Slave-Capable	Yes
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	Yes
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 (per UART Port)
DMA Descriptor	Yes
Boot Capable	Yes (Slave Mode)
Local Memory	No
Clock Operation	SCLK0_0/16

UART Functional Description

The following sections provide details on the UARTs functionality.

ADSP-SC58x UART Register List

The Universal Asynchronous Receiver/Transmitter module (UART) is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word length, stop bit, parity, and interrupt generation options. A set of registers governs UART operations. For more information on UART functionality, see the UART register descriptions.

Table 17-2: ADSP-SC58x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

ADSP-SC58x UART Interrupt List

Table 17-3: ADSP-SC58x UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
114	UART0_TXDMA	UART0 Transmit DMA	Level	20
115	UART0_RXDMA	UART0 Receive DMA	Level	21
116	UART0_STAT	UART0 Status	Level	
117	UART1_TXDMA	UART1 Transmit DMA	Level	34
118	UART1_RXDMA	UART1 Receive DMA	Level	35
119	UART1_STAT	UART1 Status	Level	
120	UART2_TXDMA	UART2 Transmit DMA	Level	37
121	UART2_RXDMA	UART2 Receive DMA	Level	38
122	UART2_STAT	UART2 Status	Level	
202	UART0_TXDMA_ERR	UART0 Transmit DMA Error	Level	

Table 17-3: ADSP-SC58x UART Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
203	UART0_RXDMA_ERR	UART0 Receive DMA Error	Level	
204	UART1_TXDMA_ERR	UART1 Transmit DMA Error	Level	
205	UART1_RXDMA_ERR	UART1 Receive DMA Error	Level	
206	UART2_TXDMA_ERR	UART2 Transmit DMA Error	Level	
207	UART2_RXDMA_ERR	UART2 Receive DMA Error	Level	

ADSP-SC58x UART Trigger List

Table 17-4: ADSP-SC58x UART Trigger List Masters

Trigger ID	Name	Description	Sensitivity
66	`UART0_TXDMA`	UART0 Transmit DMA	Edge
67	`UART0_RXDMA`	UART0 Receive DMA	Edge
68	`UART1_TXDMA`	UART1 Transmit DMA	Edge
69	`UART1_RXDMA`	UART1 Receive DMA	Edge
70	`UART2_TXDMA`	UART2 Transmit DMA	Edge
71	`UART2_RXDMA`	UART2 Receive DMA	Edge

Table 17-5: ADSP-SC58x UART Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
96	`UART0_TXDMA`	UART0 Transmit DMA	Pulse
97	`UART0_RXDMA`	UART0 Receive DMA	Pulse
98	`UART1_TXDMA`	UART1 Transmit DMA	Pulse
99	`UART1_RXDMA`	UART1 Receive DMA	Pulse
100	`UART2_TXDMA`	UART2 Transmit DMA	Pulse
101	`UART2_RXDMA`	UART2 Receive DMA	Pulse

ADSP-SC58x UART DMA Channel List

Table 17-6: ADSP-SC58x UART DMA Channel List

DMA ID	DMA Channel Name	Description
DMA20	UART0_TXDMA	UART0 Transmit DMA
DMA21	UART0_RXDMA	UART0 Receive DMA

Table 17-6: ADSP-SC58x UART DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA34	UART1_TXDMA	UART1 Transmit DMA
DMA35	UART1_RXDMA	UART1 Receive DMA
DMA37	UART2_TXDMA	UART2 Transmit DMA
DMA38	UART2_RXDMA	UART2 Receive DMA

UART Block Diagram

The *UART Block Diagram* figure shows a simplified block diagram of one UART module and how it interconnects to the processor system.

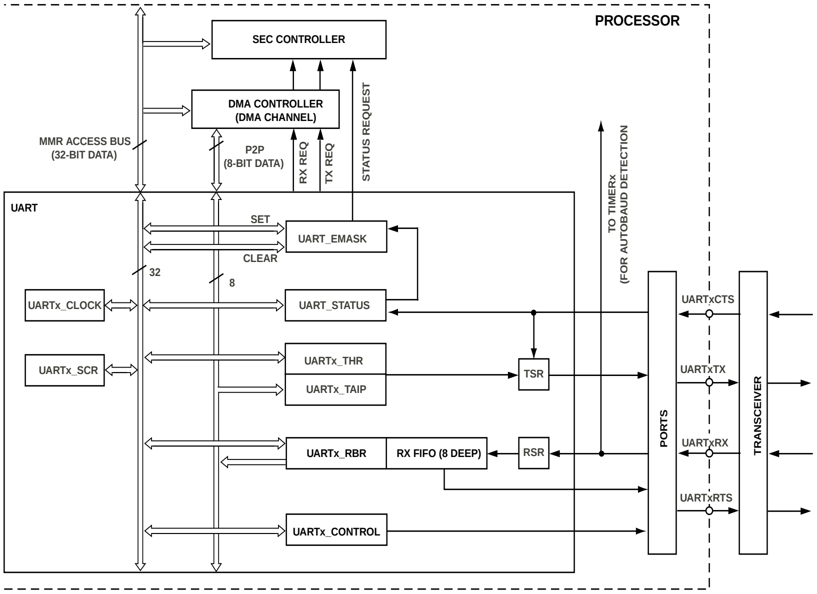

PROCESSOR
SEC CONTROLLER
DMA CONTROLLER
(DMA CHANNEL)
MMR ACCESS BUS
(32-BIT DATA)
P2P
(8-BIT DATA)
RX REQ
TX REQ
STATUS REQUEST
TO TIMERx
(FOR AUTOBAUD DETECTION)
UART
SET
CLEAR
UART_EMASK
32
8
UARTx_CLOCK
UART_STATUS
UARTx_SCR
UARTx_THR
UARTx_TAIP
TSR
UARTx_RBR
RX FIFO (8 DEEP)
RSR
UARTx_CONTROL
PORTS
TRANSCEIVER
UARTxCTS
UARTxTX
UARTxRX
UARTxRTS

Figure 17-1: UART Block Diagram

UART Architectural Concepts

The following sections provide information about the UART architecture.

Internal Interface

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It operates in either DMA or programmed core modes. The core mode requires software management of the data flow using either interrupts or polling. The DMA method requires minimal software intervention, as the DMA engine itself moves the data. The `UART_RBR` and `UART_THR` registers also connect to one of the peripheral DMA buses.

All UART registers are 32 bits wide and the registers connect to the peripheral MMR bus. Not all MMRs can be used and unused bits are zero-filled. The UART has three interrupt outputs described as follows.

- The transmit and receive request outputs can function as DMA requests and connect to the DMA controller. Therefore, if the DMA is not enabled, the DMA controller simply forwards the request to the system event controller (SEC).
- The status interrupt output connects directly to the SEC. On many processors, the alternative capture input (`TIMER_ACI[n]`) of one of the GP timers also senses the `UART_RX` pin. When configured in capture mode, the processor can then use the GP timer to detect the bit rate of the received signal.

External Interface

Each UART features a `UART_RX` (receive) pin and a `UART_TX` (transmit) pin available through the general-purpose ports. These two pins usually connect to an external transceiver device that meets the electrical requirements of full-duplex or half-duplex standards. For example, EIA-232, EIA-422, 4-wire EIA-485 for full-duplex or 2-wire EIA-485, LIN for half-duplex. Additionally, the UART features a pair of clear-to-send, input pins (`UART_CTS`), and request-to-send, output pins (`UART_RTS`) for hardware flow control. UART signals are multiplexed with other functions at the pin level.

Hardware Flow Control

To prevent the UART transmitter from sending data while the receiving counterpart is not ready, the UART features a `UART_RTS`/`UART_CTS` hardware flow control mechanism. The `UART_RTS` signal is an output that connects to the `UART_CTS` input of the communication partner. If data transfer is bidirectional, the figure shows the ***UART Hardware Flow Control*** handshake.

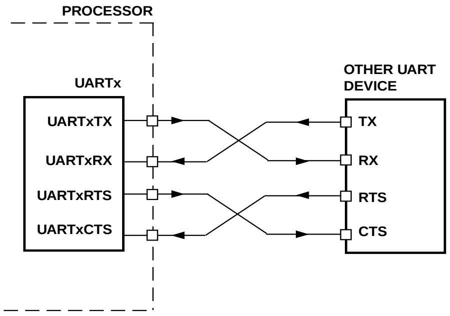


Figure 17-2: UART Hardware Flow Control

In both DMA and core mode, the receiver can deassert the `UART_RTS` signal to indicate that its receive buffer is almost full. Continued data transfers can cause an overrun error. The transmitter pauses when the `UART_CTS` input is in a deasserted state. In this state, the transmitter completes transmission of the data currently held in the transmit shift register (`UART_TSR`) but it does not continue with the data in the transmit hold register (`UART_THR`). If the `UART_CTS` pin is asserted again, the transmitter resumes and loads the content of `UART_THR` register into the `UART_TSR` register.

Bit Rate Generation

The peripheral clock (SCLK0_0) and the 16-bit divisor in the `UART_CLK` register characterize the sample clock. The UART uses the `UART_CTL.EN` bit to enable the clock. By default, every serial bit is oversampled 16 times.

The bit clock is 1/16th of the sample clock. If not in IrDA mode, the bit clock can equal the sample clock if the `UART_CLK.EDBO` bit is set, so that the following equation applies:

$\text{Bit Rate} = \text{SCLK0\_0}/16^{(1-\text{EDBO})} \times \text{Divisor}$

ADSP-SC5xx Processor Example

The ***UART Bit Rate Examples with 100 MHz*** SCLK0_0 table provides example divide factors required to support standard baud rates at an SCLK0_0 of 100 MHz.

Table 17-7: UART Bit Rate Examples with 100 MHz SCLK0_0

Bit Rate (bits/ sec)	D Factor = 16			D Factor = 1		
	DL	Actual	% Error	DL	Actual	% Error
2400	2604	2400.15	0.006	41667	2399.98	0.001
4800	1302	4800.31	0.006	20833	4800.08	0.002
9600	651	9600.61	0.006	10417	9599.69	0.003
19200	326	19171.78	0.147	5208	19201.23	0.006
38400	163	38343.56	0.147	2604	38402.46	0.006
57600	109	57339.45	0.452	1736	57603.69	0.006
115200	54	115740.74	0.469	868	115207.37	0.006
921600	7	892857.14	3.119	109	917431.19	0.452
1500000	4	1562500	4.167	67	1492537.31	0.498
3000000	2	3125000	4.167	33	3030303.03	1.01
6250000	1	6250000	0	16	6250000	0

NOTE: Careful selection of SCLK0_0 frequencies—that is, even multiples of desired bit rates— can result in lower error percentages.

Setting the bit clock equal to the sample clock (`UART_CLK.EDBO` =1) improves bit rate granularity and enables the bit clock to more closely match the bit rate of the communication partner. The disadvantage to this configuration is that the power dissipation is higher and the sample points are not always as accurate. Therefore, use `UART_CLK.EDBO` =1 mode only when bit rate accuracy is not acceptable in `UART_CLK.EDBO` =0 mode.

The `UART_CLK.EDBO` =1 mode is not intended to increase operation speed beyond the electrical limitations of the UART transfer protocol.

Autobaud Detection

At the chip level, the `UART_RX` pin is typically routed to an alternate capture input (`TIMER_ACI[n]`) of a general-purpose timer. When working in width capture mode, the processor uses this general-purpose timer to detect the bit rate applied to the $\overline{\texttt{UART\_RX}}$ pin automatically by an external device. It often uses the capture capabilities of the timer to supervise the bit rate at run time. If the UART communicates with any device supplied by a weak clock oscillator that drifts over time, the processor can then readjust its UART bit rate dynamically, as required.

Often, the processor uses autobaud detection for initial bit rate negotiations where it is most likely a slave device waiting for the host to send a predefined autobaud character. This situation is common for UART booting. Do not enable the `UART_CTL.EN` bit while autobaud detection is in-process, to prevent the UART from starting a receive operation with incorrect bit rate matching. Alternatively, set the `UART_CTL.LOOP_EN` bit to disconnect the UART from its $\overline{\texttt{UART\_RX}}$ pin.

A software routine can detect the pulse widths of serial stream bit cells. The sample base of the timer is synchronous with the UART operation (all derived from the same SCLK0_0). The UART uses pulse widths to calculate the bit rate divider as follows:

Divisor = `TIMER_TMR[n]_WID`/$16^{(1-\mathrm{EDBO})}$ × Number of captured UART bits

To increase the number of timer counts and the resolution of the captured signal, do not measure just the pulse width of a single bit. Instead, enlarge the pulse of interest over more bits. Traditionally, a NULL character (ASCII 0x00) is used in autobaud detection, as shown in the ***Autobaud Detection*** figure.

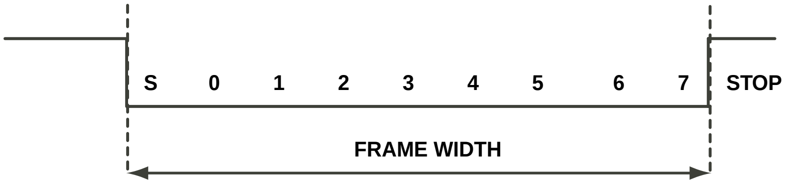


Figure 17-3: Autobaud Detection

Because the example frame encloses 8 data bits and 1 start bit, apply the following formula:

Divisor = `TIMER_TMR[n]_WID`/$16^{(1-\mathrm{EDBO})}$ × 9

> **NOTE:** For processor-specific mapping of timer alternate capture inputs to the UARTs of the processor, see "Width Capture (WIDCAP) Mode" in the "*General-Purpose Timer (TIMER)*" chapter.

Real receive signals often have asymmetrical falling and rising edges, and the sampling logic level is not exactly in the middle of the signal voltage range. At higher bit rates, such pulse-width-based autobaud detection does not always return adequate results without extra conditioning of the analog signal. Measure signal periods to work around this issue.

For example, predefine the ASCII character “@” (0x40) as the autobaud detection character and measure the period between two subsequent falling edges. As shown in the ***Autobaud Detection Character 0x40*** figure, measure the period between the falling edge of the start bit and the falling edge after bit 6. Since this period encloses 8 bits, apply the following formula:

Divisor = `TIMER_TMR[n]_PER`/$16^{(1-\mathrm{EDBO})}$ × 8

or:

- Divisor = `TIMER_TMR[n]_PER`>> 7, if `UART_CLK.EDBO`=0

- Divisor = TIMER_TMR[n]_PER>> 3, if UART_CLK.EDBO=1

The ***Autobaud Detection Character 0x40*** figure shows the ASCII "@" (0x40) detection character.

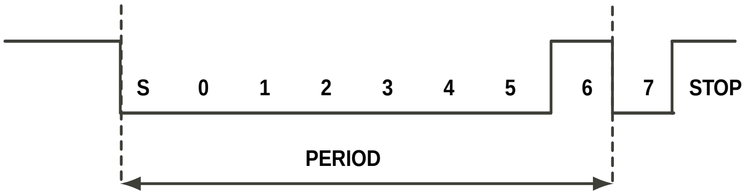


Figure 17-4: Autobaud Detection Character 0x40

UART Debug Features

The UART can automatically calculate and transmit a parity bit. The ***UART Parity*** table summarizes parity behavior assuming 8-bit data words (UART_CTL.WLS=b#11).

Table 17-8: UART Parity

PEN	STP	EPS	Data (hex)	Data (binary, LSB first)	Parity
0	x	x	x	x	None
1	0	0	0x60	0000 0110	1
1	0	0	0x57	1110 1010	0
1	0	1	0x60	0000 0110	0
1	0	1	0x57	1110 1010	1
1	1	0	x	x	1
1	1	1	x	x	0

The two force error bits, UART_CTL.FPE and UART_CTL.FFE, are intended for test purposes. They are useful for debugging software, especially in loopback mode.

The UART can be set to internal loopback mode (UART_CTL.LOOP_EN=1). Loopback mode disconnects the input of the receiver from the receive pin and internally redirects the transmit output to the receiver. The transmit pin remains active and continues to transmit data externally as well. Loopback mode also forces the UART_RTS pin to deassert, disconnects the UART_STAT.CTS bit from the UART_CTS input pin, and connects the internal version of UART_RTS to the UART_STAT.CTS bit.

Additionally, the UART_TX pin can be forced to zero asynchronously using the UART_CTL.SB bit.

UART Operating Modes

The following sections describe the main operating modes of the UART.

- UART Mode
- IrDA SIR Mode

- Multi-Drop Bus Mode

UART Mode

The UART mode follows an asynchronous serial communication protocol with these options:

- 1 start bit
- 5–8 data bits
- Address bit (available in MDB mode only)
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits valid only in 5-bit word length)

The `UART_CTL` register controls the format of received and transmitted character frames. Data is always transmitted and received with the least significant bit (LSB) first.

The ***Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)*** figure shows a typical physical bit stream measured on a `UART_TX` pin.

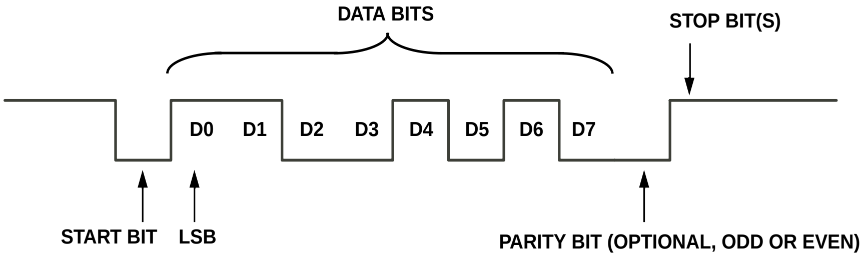


Figure 17-5: Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)

IrDA SIR Mode

The UART also supports serial data communication by way of infrared signals, according to the recommendations of the Infrared Data Association (IrDA). The physical layer known as IrDA SIR (9.6/115.2 Kbps rate) is based on return-to-zero-inverted (RZI) modulation. The UART does not support pulse position modulation.

Using the 16x data rate clock, RZI modulation is achieved by inverting and modulating the non-return-to-zero (NRZ) code normally transmitted by the UART. On the receive side, the UART uses a 16x clock to determine an IrDA pulse sample window, from which it recovers the RZI modulated NRZ code.

NOTE: The `UART_CLK.EDBO` bit is not valid in IrDA mode. Clear (=0) this bit in this mode.

Multi-Drop Bus Mode

The UART uses a protocol for point-to-point connections as well as in networks where the EIA-485 standard is representative of UART-based bus systems. The EIA-232 standard defines point-to-point connections. In such networks, node addressing is important.

In a multidrop bus (MDB) network, for example, an address bit enhances the UART frame. The address bit is inserted between the data bits and the optional parity bit. To configure the UART for MDB mode, set the mode of operation bits (`UART_CTL.MOD` [5:4]) to 01.

By convention, the address bit is transmitted low for regular data bytes. When set, it marks special address bytes that require the attention of all nodes on the network.

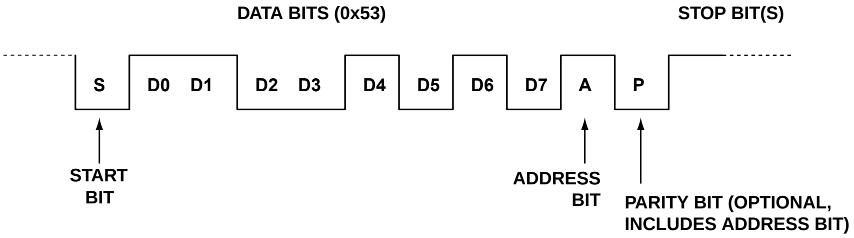


Figure 17-6: UART Frame with Address Bit

All transmit operations are processed through the transmit buffer register (`UART_THR`), so all DMA data transmissions clear the address bit. If data is written to the transmit address or insert pulse register (`UART_TAIP`) instead, the same transmit operation is initiated with the only exception that the address bit is sent high.

The UART uses the `UART_STAT.ADDR` bit of the receiver to signal whether the previously received frame had the address bit set or not. Hardware updates it every time a new frame is received. When the enable address word interrupt bit (`UART_IMSK.EAWI`) is set, the reception of an address byte triggers a special status interrupt request.

The address sticky bit (`UART_STAT.ASTKY`) is the sticky version of the `UART_STAT.ADDR` bit. Hardware sets it whenever the `UART_STAT.ADDR` bit is set. Software can clear the `UART_STAT.ASTKY` bit with a W1C operation.

In MDB mode, only address bytes progress to the receive FIFO by default. Data bytes are gated unless the `UART_STAT.ASTKY` bit is set. The receiver ignores all traffic on the UART bus. This way, the processor can go into low-power mode and interrupt activity does not load the processor every time a frame is transmitted on the UART bus. If, however, an address frame is transmitted, the receiver immediately samples all further traffic. A software routine can analyze the received data, decide whether it was of relevance for the local network node, and W1C the `UART_STAT.ASTKY` bit if it was not.

Software can overrule of the hardware address frame detection by setting the `UART_STAT.ADDR` bit and (indirectly) the `UART_STAT.ASTKY` bit with a W1S operation.

The MDB mode follows an asynchronous serial communication protocol with the following options.

- 1 start bit
- 5–8 data bits
- Address bit
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits are valid only in 5-bit word length)

NOTE: If the address bit and parity bit are both enabled, the parity check and generation includes the address bit in its parity calculation.

UART Data Transfer Modes

The UART can transfer data using both the core and DMA. Receive and transmit paths operate independently except that the bit rate and the frame format are identical for both transfer directions. Transmit and receive channels are both buffered. The `UART_THR` register buffers the transmit shift register (`UART_TSR`) and the `UART_RBR` register buffers the receive shift register (`UART_RSR`).

UART Mode Transmit Operation (Core)

In core mode, the processor core moves data to and from the UART. A write to the `UART_THR` register initiates the transmit operation. If no former operation is pending, the `UART_THR` register passes the data immediately to the `UART_TSR` register. There, it is shifted out at the bit rate characterized by the `UART_CTL` register, with start, stop, and parity bits appended as defined by the `UART_CTL` register.

The `UART_THR` register and the `UART_TSR` register can be modeled as a two-stage transmit buffer. The least significant bit (LSB) always transmits first. This bit is bit 0 of the value written to the `UART_THR` register.

UART Mode LIN Break Command

Some UART-based protocols demand synchronization methods that are not native to standard UART implementations. For example, the Local Interconnect Network (LIN) protocol requires a low-pulse of well-defined transmit length as a prologue to every multi-byte message. Its length must be at least 13 bit-times.

With previous UARTs, there were two options to implement this protocol:

- A null byte is transmitted with a temporarily lowered bit rate, or
- A software counter generates the period and the asynchronous set break (SB) mechanisms pull the transmit pin low

Since both methods have their disadvantages, the newer UART introduces a new inter-frame gap technique.

The feature is not available in MDB or IrDA operating modes. However, in standard UART mode (bits `UART_CTL.MOD` [5:4] =00), a write to the `UART_TAIP` register initiates the transmission of an inter-frame pulse. If the transmit buffer is not empty, the UART first transmits all bytes in the queue. It only initiates with pulse generation after the last stop bit of the last byte has been shifted out.

The value written into the `UART_TAIP` register defines the nature and the duration of the transmitted pulse. Bits [6:0] control the duration in bit-times and bit [7] controls the value (duration = `UART_TAIP`[6:0] / `UART_CLK`[15:0]). If `UART_TAIP`[7] is set, and an active high pulse is issued, the number of stop bits is extended. If `UART_TAIP`[7] is cleared, a low pulse is generated. Invert the polarity using the `UART_CTL.FCPOL` bit. Writing a value of 13 into the `UART_TAIP` register generates the break command as required by the LIN protocol.

NOTE: If the `UART_CTL.TPOLC` bit is enabled, an inverted most-significant bit can be transmitted.

NOTE: If another transmission is pending (in the UART_TSR register), the UART_TAIP initiated pulse is queued until after all pending operations have finished and all stop bits are transmitted.

The transmission of break command/inter-frame gap precedes transmission of the number of stop bits as set in the UART_CTL.STB and UART_CTL.STBH bit fields.

The UART receiver can detect break commands through the break indicator (UART_STAT.BI) flag. This flag reports that an entire UART frame has been received in low state. It does not report whether the duration of the received low pulse was exact or at least 13 bit-times as LIN masters transmit. Typically, the break indicator meets LIN requirements. The processor can use GP timers to determine the pulse width more precisely, if necessary.

Each UART_RX pin is also routed to a GP timer through its alternate capture input (TACI). This functionality is not only useful for bit rate detection (*autobaud*) but also helps to measure the pulse widths precisely on the UART_RX input. Additionally, the GP timers can issue an interrupt request or a fault condition when the received pulse width is shorter than a bit time or longer than the worst-case break condition. The windowed watchdog width mode of the GP timers controls this functionality.

UART Mode Receive Operation (Core)

The receive operation uses the same data format as the transmit configuration except that one valid stop bit is always sufficient. The UART_CTL.STB and UART_CTL.STBH bits have no impact on the receiver.

The UART receiver senses the falling edges of the receive input. When it detects an edge, the receiver starts sampling the input according to settings in the UART_CLK register. The receiver samples the start bit (majority sampling) close to its midpoint. If sampled low, it assumes a valid start condition. Otherwise, it discards the detected falling edge.

After detection of the start bit, the received word is shifted into the UART_RSR register.

After the corresponding stop bit is received, the content of the UART_RSR register is transferred to the 8-deep receive FIFO and is accessible by reading the UART_RBR register.

The receive FIFOs and the UART_RBR register act as a 9-stage receive buffer. If the stop bit of the ninth word is received before software reads the UART_RBR register, an overrun error is reported. Overruns protect data in the UART_RBR register and the receive FIFO from being overwritten by further data until the software clears the UART_STAT.OE bit. However, the data in the UART_RSR register is immediately destroyed as soon as the overrun occurs.

The sampling clock is 16 times faster than the bit clock. The receiver oversamples every bit 16 times and makes a majority-decision based on the middle three samples. This functionality improves immunity against noise and hazards on the line. The receiver disregards spurious pulses of less than two times the sampling clock period.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set to 1, the receiver samples bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy as required at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

Reception starts when the UART receiver detects a falling edge on the $\overline{\texttt{UART\_RX}}$ input pin. The receiver attempts to see a start bit. The data is shifted into the UART_RSR register. After the ninth sample of the first, the receiver processes the stop bit and copies the received data to the 8-stage receive FIFO. The UART_RSR recovers for further data reception.

The receiver samples data bits close to their midpoint. Because the receiver clock is typically asynchronous to the data rate of the transmitter, the sampling point can drift relative to the center of the data bits. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. The polarity of received data is selectable, using the UART_CTL.RPOLC bit.

NOTE: The receiver checks for only a single stop bit. After the third sample of the first stop bit has been received (at time 9/16th of the stop bit duration), the receiver immediately acts (status update). It then prepares for new falling edge detection (start detection).

IrDA Transmit Operation

To generate the IrDA pulse transmitted by the UART, the normal NRZ output of the transmitter is first inverted if the UART_CTL.TPOLC bit is configured for active-low operation. In this configuration, a zero is transmitted as a high pulse of 16 UART clock periods and a one is transmitted as a low pulse for 16 UART clock periods. Then, six UART clock periods delay the leading edge of the pulse. Similarly, eight UART clock periods truncate the trailing edge of the pulse. For a 16-cycle UART clock period, this operation results in the final representation of the original zero as a high pulse of only 3/16 clock periods. The ***IrDA Transmit Pulse*** figure shows how the pulse is centered around the middle of the bit time. The final IrDA pulse is fed to the off-chip infrared driver.

This modulation approach ensures a pulse width output from the UART of three cycles high out of every 16 UART clock cycles. As shown in the ***IrDA Transmit Pulse*** figure, the error terms associated with the bit rate generator are small and well within the tolerance of most infrared transceiver specifications.

NOTE: In IrDA mode, writes to the UART_TAIP register are equivalent to writes to the UART_THR register.

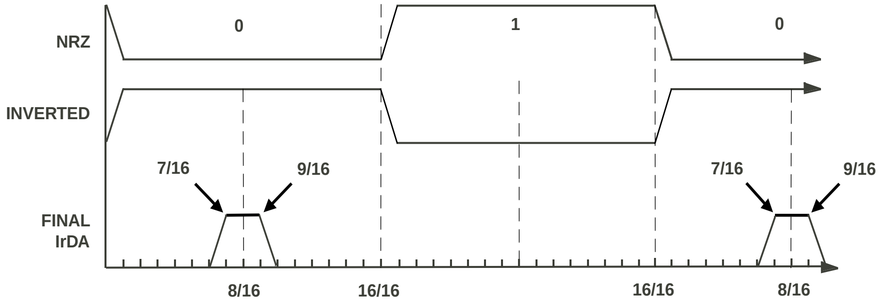


Figure 17-7: IrDA Transmit Pulse

IrDA Receive Operation

The IrDA receiver function is more complex than the transmit function. The receiver must discriminate the IrDA pulse and reject noise. The receiver looks for the IrDA pulse in a narrow window centered around the middle of the expected pulse.

Glitch filtering is accomplished by counting 16 system clocks from the time the receiver detects an initial pulse. If the pulse is absent when the counter expires, the receiver interprets it as a glitch. Otherwise, the receiver interprets it as a zero. This assessment is acceptable because glitches originating from on-chip capacitive cross-coupling typically do not last for more than a fraction of the system clock (SCLK0_0) period. Appropriate shielding avoids sources outside of the chip and not part of the transmitter. The only other source of a glitch is the transmitter itself. The processor relies on the transmitter to perform within specification. If the transmitter violates the specification, unpredictable results can occur. The 4-bit counter adds an extra level of protection at a minimal cost.

NOTE: Because SCLK0_0 can change across systems, the longest glitch tolerated is inversely proportional to the SCLK0_0 frequency.

A counter that is clocked at the 16x bit-time sample clock determines the receive sampling window. The sampling window is resynchronized with each start bit by centering the sampling window around the start bit.

The polarity of receive data is selectable, using the `UART_CTL.RPOLC` bit. The ***IrDA Receiver Pulse Detection*** figure provides examples of each polarity type.

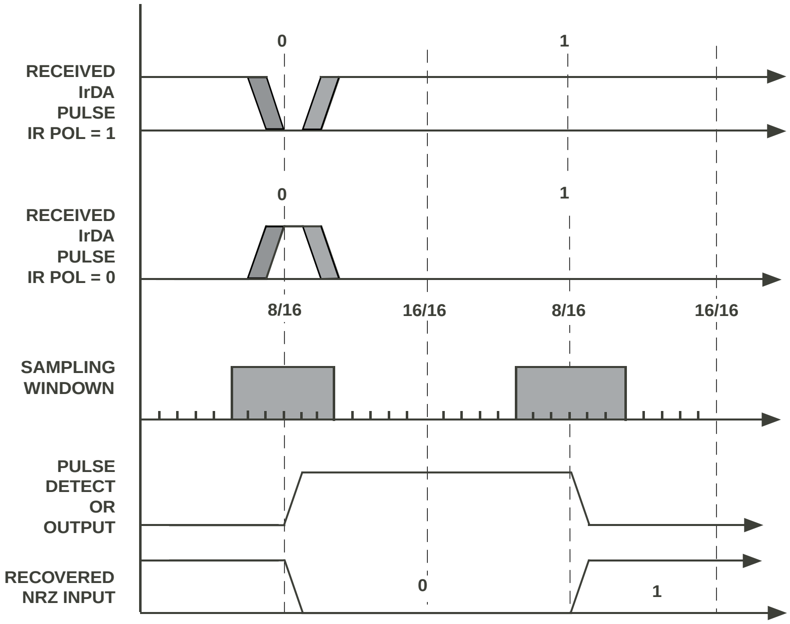


Figure 17-8: IrDA Receiver Pulse Detection

MDB Transmit Operation

In MDB mode, receive and transmit paths operate independently from each other, except for sharing bit rate and frame formats for both transfer directions.

Transmit operation is initiated by writing the `UART_THR` or `UART_TAIP` registers. A write to the `UART_THR` register transmits the written word with the appending address bit set low. A write to the `UART_TAIP` register transmits the written word with the appended address bit set high. The data is moved into the `UART_TSR` register, where it is shifted out at the bit rate programmed by the `UART_CLK` register, with start, stop, address, and parity bits appended, as required.

If DMA is enabled, the DMA engine always writes the data into the UART_THR register, and the written word is transmitted with the appending address bit set low.

The polarity of transmit data is selectable, using the UART_CTL.TPOLC bit.

MDB Receive Operation

Receive operations use the same data format as the transmit configuration, except that the number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the UART_RSR register at the programmed bit.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set, the receiver samples the bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy needed at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

After the appropriate number of bits (including address, parity, and stop bits) is received, the UART_RSR register is transferred to the receive FIFO and accessible through the UART_RBR register.

The polarity of receive data is selectable, using the UART_CTL.RPOLC bit.

DMA Mode

In DMA mode, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data; it just has to set up the appropriate transfers either through the descriptor mechanism or through autobuffer mode.

DMA channels provide a 4-deep FIFO, resulting in total buffer capabilities of 6 words at the transmit side and 9 words at the receive side. In DMA mode, the bus activity and arbitration mechanism determine the latency. The processor loading and interrupt priorities do not determine the latency.

To enable UART DMA, first set up the system DMA control registers. Then, enable the UART_IMSK.ERBFI or UART_IMSK.ETBEI interrupts. This sequence is necessary because these interrupt request lines double as DMA request lines. With DMA enabled, once these requests are received, the DMA control unit generates a direct memory access. If DMA is not enabled, the UART interrupt is passed on to the system interrupt handling unit. The status interrupt for the UART goes directly to the system event controller (SEC), bypassing the DMA unit completely.

For transmit DMA, programs must set the DMA_CFG.SYNC bit. With this bit set, interrupt generation is delayed until the entire DMA FIFO is drained to the UART module. The UART transmit DMA interrupt service routine can disable the DMA or to clear the UART_IMSK.ETBEI control bit only when the DMA_CFG.SYNC bit is set. Otherwise, up to four data bytes can be lost.

When the UART_IMSK.ETBEI bit is set, an initial transmit DMA request is issued immediately. The program then clears the UART_IMSK.ETBEI bit through the DMA service routine.

In DMA transmit mode, the UART_IMSK.ETBEI bit enables the peripheral request to the DMA FIFO. The DMA_CFG.EN bit enables the strobe on the memory side. If the DMA count is less than the DMA FIFO depth, which is 4, then the DMA interrupt can be requested before the UART_IMSK.ETBEI bit is set. If this behavior is unwanted, set the DMA_CFG.SYNC bit.

Regardless of the `DMA_CFG.SYNC` setting, the DMA stream has not left the UART transmitter completely at the time the interrupt request is generated. Transmission can abort in the middle of the stream, causing data loss, when the UART clock was disabled without extra synchronization with the `UART_STAT.TEMT` bit.

The UART provides functionality to avoid resource-consuming polling of the `UART_STAT.TEMT` bit. The `UART_IMSK_SET.EDTPTI` bit enables the `UART_STAT.TEMT` bit to trigger a DMA interrupt. To delay the DMA completion interrupt until the last data word of a STOP DMA has left the UART, keep the `DMA_CFG.INT` bit cleared and set the `UART_IMSK_SET.EDTPTI` bit instead. Then, the normal DMA completion interrupt is suppressed. Later, the `UART_STAT.TEMT` event triggers a DMA interrupt after the last word of the DMA has left the UART transmit buffers. If `DMA_CFG.INT` and `UART_IMSK.EDTPTI` are set, when finishing STOP mode, the DMA requests two interrupts.

The DMA of the UART module supports 8-bit and 16-bit operation, but not 32-bit operation. It does not support sign-extension.

Mixing DMA and Core Modes

Switching from DMA mode to core operation dynamically requires some consideration, especially for transmit operations. By default, the interrupt timing of the DMA is synchronized with the memory side of the DMA FIFOs. Normally, the transmit DMA completion interrupt is generated after the last byte is copied from the memory into the DMA FIFO. The transmit DMA interrupt service routine is not yet permitted to disable the `DMA_CFG.EN` bit. The interrupt is requested when the `DMA_STAT.IRQDONE` bit is set. The `DMA_STAT.RUN` bit, however, remains set until the data has completely left the transmit DMA FIFO.

When planning to switch from a DMA to core mode, set the `DMA_CFG.SYNC` bit in the word of the last descriptor or work unit before handing over control. Then, after the interrupt request occurs, software can write new data into the `UART_THR` register as soon as the `UART_STAT.THRE` bit permits. If the `DMA_CFG.SYNC` bit cannot be set, software can poll the `DMA_STAT.RUN` bit instead. Alternatively, using the `UART_IMSK.EDTPTI` bit can avoid expensive status bit polling.

When switching from core to DMA operation, ensure that the first DMA request is issued properly. If the DMA is enabled while the UART is still transmitting, no precaution is required. If, however, the DMA is enabled after the `UART_STAT.TEMT` bit is high, pulse the `UART_IMSK.ETBEI` bit to initiate DMA transmission.

Setting Up Hardware Flow Control

Use the following steps to set up UART hardware flow control.

1. Configure automatic or manual hardware flow control for the receiver through the `UART_CTL.ARTS` bit, or the transmitter through the `UART_CTL.ACTS` bit.
2. Configure $\overline{\texttt{UART\_CTS}}$ and $\overline{\texttt{UART\_RTS}}$ polarity through the `UART_CTL.FCPOL` bit.

On reset, when the UART is not yet enabled and the port multiplexing has not been programmed, the $\overline{\texttt{UART\_RTS}}$ pin is not driven. Some applications require a resistor pull the `UART_RTS` signal to either state during reset.

UART Event Control

Status flags in the `UART_STAT` register are available to signal data reception, parity, and error conditions, if necessary.

DMA and Interrupt Multiplexing

See the *Direct Memory Access (DMA)* chapter on for information on DMA multiplexing. Several interrupts and DMA channels in thr UART can be multiplexed.

> **NOTE:** To operate in interrupt mode without using DMA channels, set the `UART_IMSK.ELSI` bit. This configuration redirects receive and transmit requests to the status interrupt output. The status interrupt goes directly to the SEC without going through the DMA controller.

Interrupt Masks

Each UART features a set of interrupt mask registers: `UART_IMSK`, `UART_IMSK_SET`, and `UART_IMSK_CLR`. The `UART_IMSK` register supports read/write operations. Writing ones to the `UART_IMSK_SET` register enables interrupts, writing ones to the `UART_IMSK_CLR` register disables them. Reads from either register return the enabled bits. This way, different interrupt service routines can control transmit, receive, and status interrupt requests independently and easily.

The UART module uses the `UART_IMSK` registers to enable requests for system handling of empty or full states of data registers. Unless polling is used as a means of action, the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` bits in this register are normally set.

Each UART module has three interrupt outputs. It uses one for transmission, one for reception, and one for reporting status events. The UART module routes transmit and receive requests through the DMA controller. The status request goes directly to the system event controller (SEC).

If the associated DMA channel is enabled, the request functions as a DMA request. If the DMA channel is disabled, it simply forwards the request to the SEC. A DMA channel must be associated with the UART module to enable transmit and receive interrupts. Otherwise, transmit and receive requests cannot be forwarded.

> **NOTE:** To operate in interrupt mode without using DMA channels, set the `UART_IMSK.ELSI` bit. This configuration redirects receive and transmit requests to the status interrupt request output. The status interrupt goes directly to the SEC without going through the DMA controller.

Interrupt Servicing

Interrupt service routines (ISRs) perform UART writes and reads. Separate interrupt lines are provided for transmit, receive, and status. The `UART_IMSK` register group enables the independent interrupts individually. To enable UART transmit interrupts, set the `UART_CTL.EN` bit.

The ISRs evaluate the status bits in the `UART_STAT` register to determine the signaling interrupt source. The system event controller for the processor assigns and unmasks interrupts. The ISRs must clear the interrupt latches explicitly. To reduce interrupt frequency on the receive side in core mode, use the `UART_IMSK.ERFCI` status interrupt as an alternative to the regular UART_IMSK.ERBFI receive interrupt. Hardware must ensure that at least two (if UART_CTL.RFIT=0) or four (if UART_CTL.RFIT=1) words are available in the receive buffer by the time the interrupt is requested.

Transmit Interrupts

The UART module uses the UART_IMSK_SET.ETBEI bit to enable transmit interrupt requests.

The UART_THR and UART_TAIP registers are the same physical register, and both affect the signaling of the UART_STAT.TEMT, UART_STAT.TFI, and UART_STAT.THRE bits similarly.

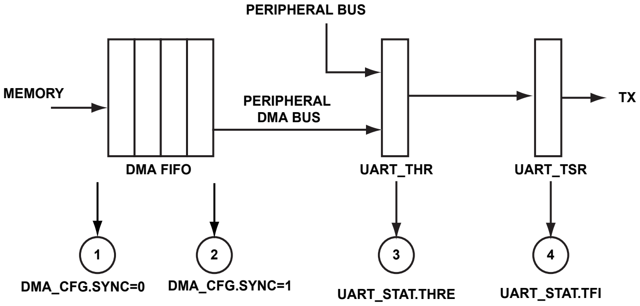


Figure 17-9: Transmit Interrupts

The UART module asserts the transmit request along with the UART_STAT.THRE bit, indicating that the transmit buffer is ready for new data. The UART_STAT.THRE bit resets to 1. When the UART_IMSK_SET.ETBEI bit is set, the UART module immediately issues an interrupt or DMA request. This way, no special handling of the first character is required when transmission of a string is initiated. Set the UART_IMSK_SET.ETBEI bit and let the interrupt service routine load the first character from memory and write it to the UART_THR register in the normal manner. ISRs can clear the UART_IMSK.ETBEI bit through the UART_IMSK_CLR register when the string transmission has completed.

Hardware clears the UART_STAT.THRE bit when new data is written to the UART_THR register. These write operations also clear the transmit interrupt request. However, they also initiate further transmission. If continued transmission is undesirable, the UART module can alternatively clear the transmit request through the UART_IMSK_CLR.ETBEI bit register. Transfers of data from the UART_THR register to the UART_TSR register reset this status flag in the UART_STAT register.

ISRs can interrogate the UART_STAT.TEMT bit to discover any ongoing transmission. The sticky counterpart of the UART_STAT.TEMT bit, UART_STAT.TFI, indicates when the transmit buffer has drained and can trigger a status interrupt. When data is pending in either one of these registers, the UART_STAT.TEMT flag is low. As soon as all data has left the UART_TSR register, the UART_STAT.TEMT bit goes high again and indicates that all pending transmit operations (including stop bits) have finished. Then, it is safe to disable the UART_CTL.EN bit or to three-state off-chip line drivers. Then, the UART module can generate an interrupt either through the status interrupt channel when the UART_IMSK.ETFI bit is set, or through the DMA controller when enabled by the UART_IMSK.EDTPTI bit.

When enabled by the `UART_IMSK.ETBEI` bit, the `UART_STAT.THRE` flag requests data along the peripheral command lines to the DMA controller (referred to as TXREQ). This signal is routed through the DMA controller. If the associated DMA channel is enabled, the TXREQ signal functions as a DMA request, otherwise the DMA controller simply forwards it to the SEC. Alternatively the `UART_IMSK.ETXS` bit can redirect the transmit interrupts to the UART status interrupt.

With interrupts disabled, the UART module can poll the status flags to determine when data is ready to move. Because polling is processor intensive, it is not typically used in real-time signal processing environments. Since read operations from `UART_STAT` registers have no side effects, different software threads can interrogate these registers without mutual impacts.

Polling the `SEC_SSTAT[n]` register without enabling the interrupts by the `SEC_CCTL[n]` register is an alternate method of operation to consider. Software can write up to two words into the `UART_THR` register before enabling the UART clock. As soon as the `UART_CTL.EN` bit is set, the UART module sends those two words.

Receive Interrupts

The UART module uses the `UART_IMSK_SET.ERBFI` bit to enable receive interrupt requests. If set, the `UART_STAT.DR` flag requests an interrupt on the dedicated RXREQ output, indicating that new data is available in the `UART_RBR` register. This signal is routed through the DMA controller. If the associated DMA channel is enabled, the RXREQ signal functions as a DMA request; otherwise the DMA controller simply forwards it to the SEC. Alternatively, if no DMA channel is assigned to the UART, the `UART_IMSK.ERXS` bit can redirect the receive interrupts to the UART status interrupt. When software reads the `UART_RBR` register, hardware clears the `UART_STAT.DR` bit again, which, in turn, clears the receive interrupt request.

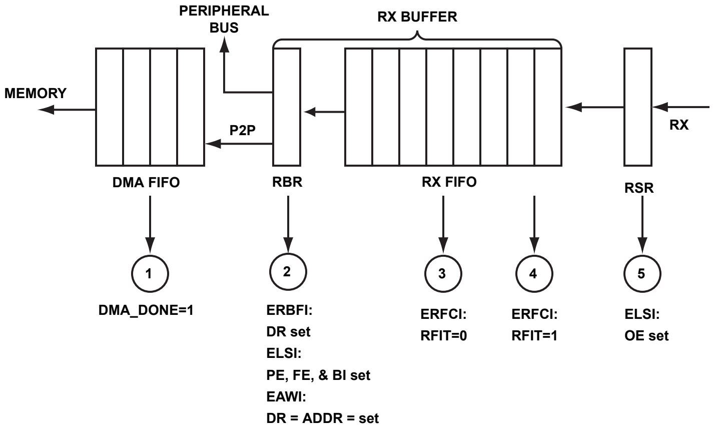


Figure 17-10: Receive Interrupts

Hardware updates the following:

- `UART_STAT.DR` bits
- `UART_STAT.ADDR` bits
- `UART_STAT.ASTKY` bits
- `UART_STAT.PE` bits

- `UART_STAT.FE` bits
- `UART_STAT.BI` bits
- `UART_RBR` register

The `UART_STAT.OE` bit is updated as soon as an overflow condition occurs (for example when a stop bit for a frame is received and the receive FIFO is full). When software does not read the `UART_RBR` register in time, the received data is protected from being overwritten by new data until software clears the `UART_STAT.OE` bit. Only the content of the `UART_RSR` register can be overwritten in the overrun case.

The UART module uses the `UART_STAT.RFCS` bit to monitor the state of the 8-deep receive FIFO. It uses the `UART_CTL.RFIT` bit to control the behavior of the buffer. If `UART_CTL.RFIT` is zero, the `UART_STAT.RFCS` bit is set when the receive buffer holds four or more words. If `UART_CTL.RFIT` is set, the `UART_STAT.RFCS` bit is set when the receive buffer holds seven or more words. Hardware clears the `UART_STAT.RFCS` bit when a core or DMA reads the `UART_RBR` register and when the buffer is flushed below the level of four (`UART_CTL.RFIT`=0) or seven (`UART_CTL.RFIT`=1). If the associated interrupt bit `UART_IMSK.ERFCI` is enabled, a status interrupt request is reported when the `UART_STAT.RFCS` bit is set.

If errors are detected during reception, an interrupt can be requested from the status interrupt output. This status interrupt request goes directly to the SEC. The bit enables status interrupt requests.

The controller detects the following error conditions, shown with their associated bits in the `UART_STAT` register.

- Overrun error (`UART_STAT.OE` bit)
- Parity error (`UART_STAT.PE` bit)
- Framing error or invalid stop bit (`UART_STAT.FE` bit)
- Break indicator (`UART_STAT.BI` bit)

Status Interrupts

The UART module uses status interrupt channels for the following purposes:

- Line status interrupt requests
- Flow control interrupt requests
- Receive FIFO threshold interrupt requests
- Transmission finished interrupt request

The UART module uses the `UART_IMSK.ELSI` bit to enable the line status interrupts. If set, the status interrupt request is asserted with one of the `UART_STAT.BI`, `UART_STAT.FE`, `UART_STAT.PE`, or `UART_STAT.OE` receive errors bits. A W1C operation to the `UART_STAT` register clears the error bits. Once all error conditions are cleared, the interrupt request deasserts.

The UART module uses the `UART_IMSK_SET.ERFCI` bit to enable the receive FIFO count interrupt. If set, a status interrupt request is generated when the `UART_STAT.RFCS` is active. The `UART_STAT.RFCS` bit indicates a receive buffer threshold level. If the `UART_CTL.RFIT` bit is cleared, software can safely read two words out of the `UART_RBR` register by the time the `UART_STAT.RFCS` interrupt occurs.

If the `UART_CTL.RFIT` bit is set, software can safely read four words. The interrupt request and the `UART_STAT.RFCS` bit are cleared when the `UART_RBR` is read enough of times, so that the receive buffer drains below the threshold of two (`UART_CTL.RFIT`=0) or four (`UART_CTL.RFIT`=1). Because in DMA mode a status service routine may not be permitted to read `UART_RBR`, this interrupt is only recommended in core mode. In DMA mode, use this functionality for error recovery only.

The UART module uses the `UART_IMSK_SET.EDSSI` bit to enable the flow control interrupts. If active, a status interrupt is generated when the sticky `UART_STAT.SCTS` bit register is set, indicating that the `UART_CTS` input of the transmitter been reasserted. A W1C operation to the `UART_STAT.SCTS` bit clears the interrupt request.

The UART module uses the `UART_IMSK_SET.ETFI` bit to enable the transmission finished interrupt. If active, a status interrupt request is asserted when the `UART_STAT.TFI` bit is set. The `UART_STAT.TFI` is the sticky version of the `UART_STAT.TEMT` bit, indicating that a byte that started transmission has finished. A W1C operation to the `UART_STAT.TFI` bit clears the interrupt request.

Multi-Drop Bus Events

Several status bits and interrupt features in the `UART_STAT` and `UART_IMSK` registers facilitate efficient data handling in multi-drop bus mode. These features include the address (`UART_STAT.ADDR`) bit, the address sticky (`UART_STAT.ASTKY`) bit and the enable address word interrupt (`UART_IMSK.EAWI`). One of the key features of the multi-drop bus protocol is its address bit. The address bit signifies to the slaves that the master is transmitting an address word (all read it) or a data word (only the addressed slave reads its). The UART hardware provides an efficient method of handling the situation described with the use of `UART_STAT.ASTKY` bit.

NOTE: The UART module uses the `UART_STAT.ASTKY` bit in multi-drop bus mode to indicate when an address operation for a peripheral is occurring. The `UART_STAT.ASTKY` bit is a sticky version of the `UART_STAT.ADDR` bit. Hardware sets the bit whenever the `UART_STAT.ADDR` bit is set. Only software clears it with a W1C operation. With the `UART_STAT.ASTKY` bit set, words are received irrespective of the mode bit or address bit setting. With the `UART_STAT.ASTKY` bit cleared, only address words (mode bit =1) are received and words with mode bit =0 are ignored in MDB mode. This bit does not affect reception in non-MDB modes. (Words with mode bit =0 are not moved from the `UART_RSR` register to the receive FIFO.)

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

Refer to Autobaud Detection section for more information. The required steps are:

1. Ensure that the timer is disabled.

2. Configure the following bits: `UART_CTL.MOD` =00, `UART_CTL.LOOP_EN` =1, `UART_CTL.WLS` =11 (8-bit data), and `UART_CTL.EN` =1
3. Configure the following bits: `TIMER_TMR[n]_CFG.TMODE` =1101, `TIMER_TMR[n]_CFG.OUTDIS` =1, `TIMER_TMR[n]_CFG.IRQMODE` =10 and enable the timer.
4. Send test data through the host device and wait for the timer interrupt and disable the timer.

   The bit rate can be derived from the timer period register value according to the formula provided in the Auto-baud Detection section.

Using Common Initialization Steps

When using the core or the DMA to execute transfers, the following steps are common to all UART modes.

1. All UART signals are multiplexed and compete with other functions at pin level. First, program the port registers according to the guidelines in the PORT chapter.
2. Program the `UART_CLK` register. Refer to Bit Rate Generation topic.
3. Program the `UART_CTL` register and enable the UART clock.

Using Core Transfers

Write data into the `UART_THR` register, when the `UART_STAT.THRE` bit is set, to initiate a core transmit operation. If the `UART_STAT.DR` bit is set, received data can be read from the `UART_RBR` register.

Using DMA Transfers

1. Make sure that the `UART_IMSK.ETBEI` or the `UART_IMSK.ERBFI` bits are cleared before configuring the DMA.
2. Configure the dedicated DMA channel.
3. Set the `UART_IMSK.ETBEI` or `UART_IMSK.ERBFI` bits to start the transfer.

Using Interrupts

Each UART features three interrupt signal outputs.

1. Enable individual interrupts in the system event controller (SEC).
2. Register IRQ handlers.
3. Use the interrupts mask registers to enable specific IRQ events.

Setting Up Hardware Flow Control

1. Configure automatic or manual hardware flow control for the receiver through the `UART_CTL.ARTS` bit, or the transmitter through the `UART_CTL.ACTS` bit.
2. Configure $\overline{\texttt{UART\_CTS}}$ and $\overline{\texttt{UART\_RTS}}$ polarity through the `UART_CTL.FCPOL` bit.

ADSP-SC58x UART Register Descriptions

UART (UART) contains the following registers.

Table 17-9: ADSP-SC58x UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

Clock Rate Register

The `UART_CLK` register divides the system clock (SCLK0_0) down to the bit clock.

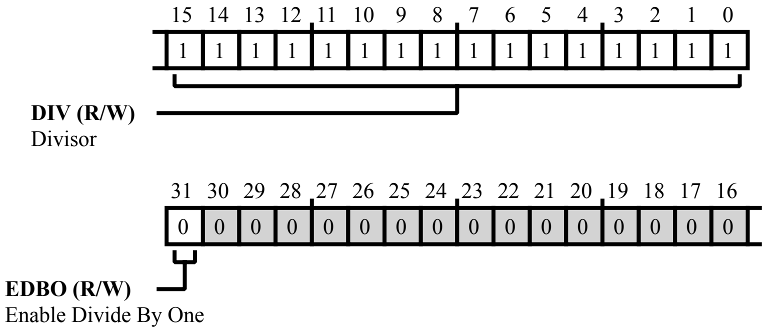

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
DIV (R/W)
Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EDBO (R/W)
Enable Divide By One

Figure 17-11: UART_CLK Register Diagram

Table 17-10: UART_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EDBO	Enable Divide By One. The `UART_CLK.EDBO` bit enables the bypassing of the divide-by-16 prescaler in bit clock generation. This functionality improves bit rate granularity, especially at high bit rates. Do not set this bit in IrDA mode.	
		0	Bit clock prescaler = 16
		1	Bit clock prescaler = 1
15:0 (R/W)	DIV	Divisor. The `UART_CLK.DIV` provides the divisor for the UART's clock bit rate calculation. The bit rate is defined by: Bit Rate = SCLK0_0 / ($16^{(1\text{-EDBo})}$ x `UART_CLK.DIV`)	

Control Register

The `UART_CTL` register provides enable and disable control for internal UART and for the IrDA mode of operation. This register also provides UART line control, permitting selection of the format of received and transmitted character frames. Modem feature control also is available from this register, including partial modem functionality to allow for hardware flow control and loopback mode.

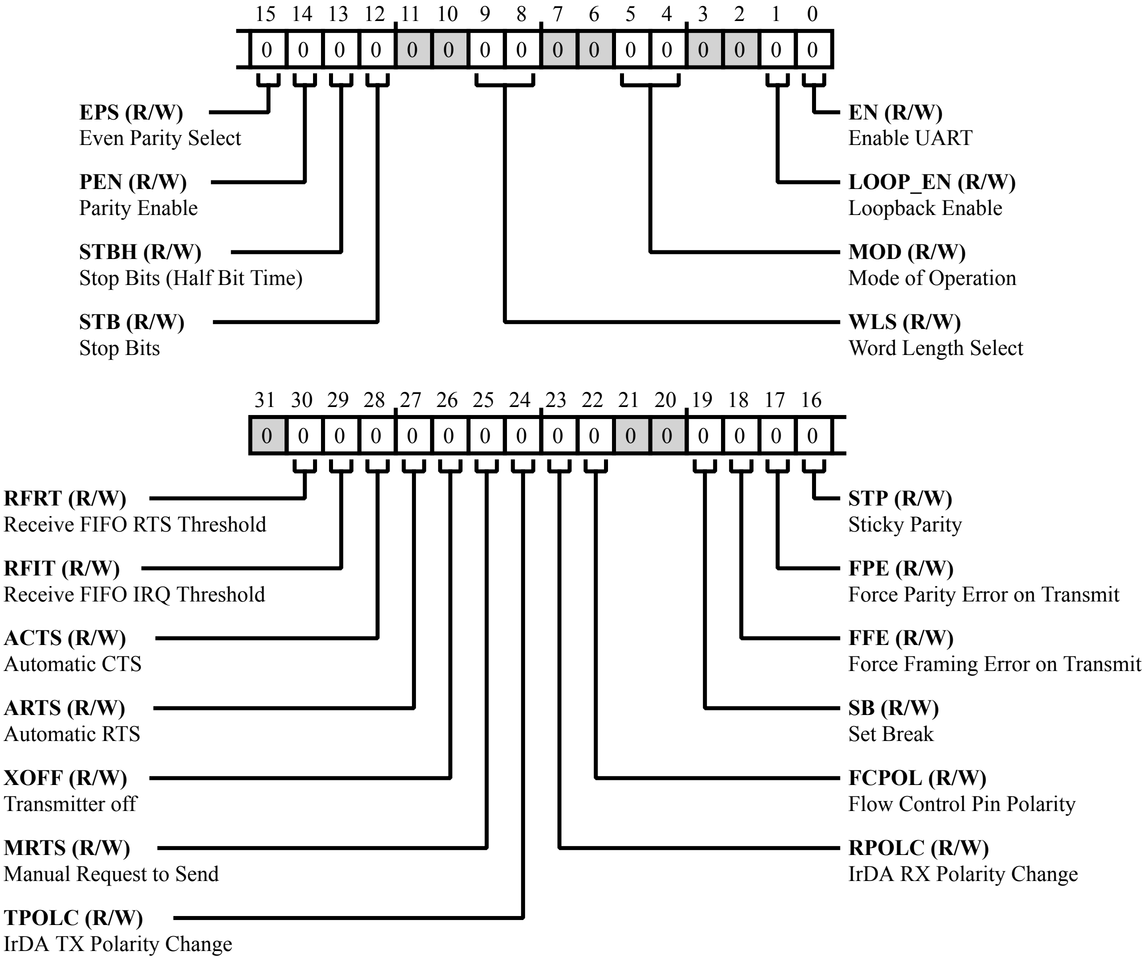

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EPS (R/W)
Even Parity Select
PEN (R/W)
Parity Enable
STBH (R/W)
Stop Bits (Half Bit Time)
STB (R/W)
Stop Bits
EN (R/W)
Enable UART
LOOP_EN (R/W)
Loopback Enable
MOD (R/W)
Mode of Operation
WLS (R/W)
Word Length Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RFRT (R/W)
Receive FIFO RTS Threshold
RFIT (R/W)
Receive FIFO IRQ Threshold
ACTS (R/W)
Automatic CTS
ARTS (R/W)
Automatic RTS
XOFF (R/W)
Transmitter off
MRTS (R/W)
Manual Request to Send
TPOLC (R/W)
IrDA TX Polarity Change
STP (R/W)
Sticky Parity
FPE (R/W)
Force Parity Error on Transmit
FFE (R/W)
Force Framing Error on Transmit
SB (R/W)
Set Break
FCPOL (R/W)
Flow Control Pin Polarity
RPOLC (R/W)
IrDA RX Polarity Change

Figure 17-12: UART_CTL Register Diagram

Table 17-11: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RFRT	Receive FIFO RTS Threshold. The `UART_CTL.RFRT` bit controls `UART_RTS` pin assertion and deassertion timing. This bit is ignored if `UART_CTL.ARTS` is cleared. If set, the `UART_RTS` pin is deasserted when the receive buffer already holds seven words and an eighth start bit is detected. It is reasserted when the FIFO contains seven words or less. If cleared, the `UART_RTS` pin is deasserted when the RX buffer already holds four words and a fifth start bit is detected. The `UART_RTS` pin is reasserted when the RX buffer contains no more than 4 words.	
		0	Deassert RTS if RX FIFO word count > 4; assert if <= 4
		1	Deassert RTS if RX FIFO word count > 7; assert if <= 7
29 (R/W)	RFIT	Receive FIFO IRQ Threshold. The `UART_CTL.RFIT` bit controls the timing of the `UART_STAT.RFCS` bit. If `UART_CTL.RFIT` is cleared, the receive threshold is two. If `UART_CTL.RFIT` is set, the threshold is four words in the receive buffer.	
		0	Set RFCS=1 if RX FIFO count >= 4
		1	Set RFCS=1 if RX FIFO count >= 7
28 (R/W)	ACTS	Automatic CTS. The `UART_CTL.ACTS` bit must be set to enable the `UART_CTS` input pin for `UART_TX` handshaking. If enabled, the `UART_STAT.CTS` bit holds the value (if `UART_CTL.FCPOL` is set) or complement value (if `UART_CTL.FCPOL` is cleared) of the `UART_CTS` input pin. The `UART_STAT.CTS` bit can be used to determine whether the external device is ready to receive data (if `UART_STAT.CTS` is set) or whether it is busy (if `UART_STAT.CTS` is cleared). If `UART_CTL.ACTS` is cleared, the `UART_TX` handshaking protocol is disabled, and the `UART_TX` line transmits data whenever there is data to send, regardless of the value of `UART_CTS`. Software can pause ongoing transmission by setting the `UART_CTL.XOFF` bit.	
		0	Disable TX handshaking protocol
		1	Enable TX handshaking protocol
27 (R/W)	ARTS	Automatic RTS. The `UART_CTL.ARTS` bit must be set to enable the `UART_RTS` input pin for `UART_TX` handshaking. If set, the hardware guarantees a minimal `UART_RTS` pin deassertion pulse width of at least the number of data bits defined by the `UART_CTL.WLS` bit field. If cleared, the `UART_RTS` pin is not generated automatically by hardware. The `UART_RTS` pin can still be manually controlled by the `UART_CTL.MRTS` bit, and software is responsible for `UART_RTS` pulse width control (if needed).	
		0	Disable RX handshaking protocol.
		1	Enable RX handshaking protocol.

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	XOFF	Transmitter off. The `UART_CTL.XOFF` bit (if set) turns off transmission (XOFF) by preventing the content of THR from being continued to TSR. When set, this bit turns on transmission (XON). The state of the `UART_CTL.XOFF` bit is ignored if the `UART_CTL.ACTS` bit is set.	
		0	Transmission ON, if ACTS=0
		1	Transmission OFF, if ACTS=0
25 (R/W)	MRTS	Manual Request to Send. The `UART_CTL.MRTS` bit controls the state of the `UART_RTS` output pin when the `UART_CTL.ARTS` bit is cleared. When `UART_CTL.MRTS` is cleared, the UART deasserts the `UART_RTS` pin, signaling to the external device that the UART is not ready to receive. When `UART_CTL.MRTS` is set, the UART asserts the `UART_RTS` pin, signaling to the external device that the UART is ready to receive.	
		0	Deassert RTS pin when ARTS=0
		1	Assert RTS pin when ARTS=0
24 (R/W)	TPOLC	IrDA TX Polarity Change. The `UART_CTL.TPOLC` bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the `UART_TX` pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the `UART_TX` pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low TX polarity setting
		1	Active-high TX polarity setting
23 (R/W)	RPOLC	IrDA RX Polarity Change. The `UART_CTL.RPOLC` bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the `UART_RX` pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the `UART_RX` pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low RX polarity setting
		1	Active-high RX polarity setting

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	FCPOL	Flow Control Pin Polarity. The `UART_CTL.FCPOL` bit selects the polarities of the `UART_CTS` and `UART_RTS` pins. When the `UART_CTL.FCPOL` bit is cleared, the `UART_RTS` and `UART_CTS` pins are active low, and the UART is halted when the `UART_RTS` and `UART_CTS` pin state is high. When `UART_CTL.FCPOL` bit is set, the `UART_RTS` and `UART_CTS` pins are active high, and the UART is halted when the `UART_RTS` and `UART_CTS` pin state is low.	
		0	Active low CTS/RTS
		1	Active high CTS/RTS
19 (R/W)	SB	Set Break. If set, the `UART_CTL.SB` bit forces the `UART_TX` pin to low asynchronously, regardless of whether or not data is currently transmitted. This bit functions even when the UART clock is disabled. Because the `UART_TX` pin normally drives high, it can be used as a flag output pin, if the UART is not used. (For example, if `UART_CTL.TPOLC` is cleared, drive `UART_TX` pin low; or if `UART_CTL.TPOLC` is set, drive `UART_TX` pin high.)	
		0	No force
		1	Force TX pin to 0
18 (R/W)	FFE	Force Framing Error on Transmit. The `UART_CTL.FFE` bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force error
17 (R/W)	FPE	Force Parity Error on Transmit. The `UART_CTL.FPE` bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force parity error

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	STP	Sticky Parity. The `UART_CTL.STP` bit controls whether the parity is generated by hardware based on the data bits or whether it is set to a fixed value. If this bit is cleared, the hardware calculates the parity bit value based on the data bits. Then, the EPS bit determines whether odd or even parity mode is chosen. If this bit is set, odd parity is used. That means that the total count of logical-1 data bits including the parity bit must be an odd value. Even parity is chosen by `UART_CTL.STP` cleared and `UART_CTL.EPS` set. Then, the count of logical-1 bits must be a even value. If the `UART_CTL.STP` bit is set, hardware parity calculation is disabled. In this case, the sent and received parity equals the inverted `UART_CTL.EPS` bit.	
		0	No forced parity
		1	Force (Stick) parity to defined value (if PEN=1)
15 (R/W)	EPS	Even Parity Select.	
		0	Odd parity
		1	Even parity
14 (R/W)	PEN	Parity Enable. The `UART_CTL.PEN` bit enables parity transmission and parity check. The `UART_CTL.PEN` bit inserts one additional bit between the most significant data bit and the first stop bit. The polarity of this so-called parity bit depends on data and the `UART_CTL.STP` and `UART_CTL.EPS` control bits. Both transmitter and receiver calculate the parity value. The receiver compares the received parity bit with the expected value and issues a parity error if they do not match. If the `UART_CTL.PEN` bit is cleared, the `UART_CTL.STP` and the `UART_CTL.EPS` bits are ignored.	
		0	Disable
		1	Enable parity transmit and check
13 (R/W)	STBH	Stop Bits (Half Bit Time).	
		0	0 half-bit-time stop bit
		1	1 half-bit-time stop bit
12 (R/W)	STB	Stop Bits. The `UART_CTL.STB` bit controls how many stop bits are appended to transmitted data.	
		0	1 stop bit
		1	2 stop bits

Table 17-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	WLS	Word Length Select. The `UART_CTL.WLS` field determines whether the transmitted and received UART word consists of 5, 6, 7, or 8 data bits.	
		0	5-bit word
		1	6-bit word
		2	7-bit word
		3	8-bit word
5:4 (R/W)	MOD	Mode of Operation. The `UART_CTL.MOD` selects the UART operation mode (UMOD).	
		0	UART mode
		1	MDB mode
		2	IrDA SIR mode
1 (R/W)	LOOP_EN	Loopback Enable. The `UART_CTL.LOOP_EN` bit enables UART loopback mode. When set, this bit disconnects the input of the receiver from the `UART_RX` pin, and internally redirects the transmit output to the receiver. The `UART_TX` pin remains active and continues to transmit data externally as well. Loopback mode also forces the `UART_RTS` pin to its deassertive state, disconnects the `UART_CTS` bit from the `UART_CTS` input pin, and directly connects the `UART_CTL.MRTS` bit to the `UART_STAT.CTS` bit. In loopback mode, setting the `UART_CTL.MRTS` bit sets the `UART_STAT.CTS` bit and enables the transmitter of the UART. Clearing the `UART_CTL.MRTS` bit clears the `UART_STAT.CTS` bit and disables the transmitter of the UART.	
		0	Disable
		1	Enable
0 (R/W)	EN	Enable UART. The `UART_CTL.EN` enables the UART clocks. This bit also resets the state machine and control registers when cleared. Using this bit to disable the UART -- when not used -- reduces power consumption.	
		0	Disable
		1	Enable

Interrupt Mask Register

The `UART_IMSK` register indicates the interrupt mask status (unmasked, if set, or masked, if cleared) of the UART status interrupt requests. This register is not a data register. Instead, it is controlled by the `UART_IMSK_SET` and `UART_IMSK_CLR` register pair. Writing ones to the `UART_IMSK_SET` register enables (unmasks) interrupt requests, and writing ones to the `UART_IMSK_CLR` register disables (masks) them. Reads from either register return the enabled bits.

The `UART_IMSK` register is used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` bits are normally set. Setting this register without enabling system DMA causes the UART to notify the processor of the data inventory state using interrupts. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present.

Each UART features three separate interrupt channels to handle the data transmit, data receive, and line status events independently, regardless of whether DMA is enabled or not. If no DMA channels are assigned to the UART, set the `UART_IMSK.ELSI` bit to reroute the transmit and receive interrupts to the status interrupt request output.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available for receive and transmit operations. Line error handling can be configured independently from the receive or transmit setup.

The DMA of the UART is enabled by first setting up the system DMA control registers and then enabling the `UART_IMSK.ERBFI` and `UART_IMSK.ETBEI` interrupts. This configuration is because the interrupt request lines double as DMA request lines. Depending on whether DMA is enabled or not, upon receiving these requests, the DMA control unit either generates a direct memory access or passes the UART interrupt on to the system interrupt handling unit(s). However, the error interrupt for the UART goes directly to the system interrupt handling unit(s), bypassing the DMA unit completely.

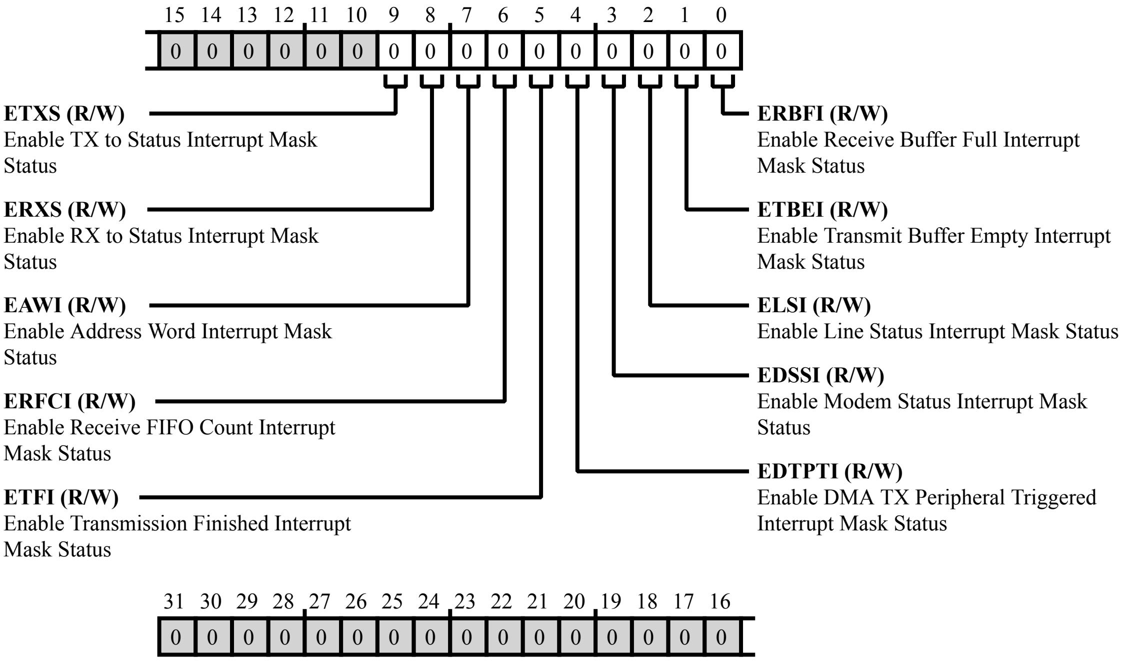

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ETXS (R/W)
Enable TX to Status Interrupt Mask Status
ERXS (R/W)
Enable RX to Status Interrupt Mask Status
EAWI (R/W)
Enable Address Word Interrupt Mask Status
ERFCI (R/W)
Enable Receive FIFO Count Interrupt Mask Status
ETFI (R/W)
Enable Transmission Finished Interrupt Mask Status
ERBFI (R/W)
Enable Receive Buffer Full Interrupt Mask Status
ETBEI (R/W)
Enable Transmit Buffer Empty Interrupt Mask Status
ELSI (R/W)
Enable Line Status Interrupt Mask Status
EDSSI (R/W)
Enable Modem Status Interrupt Mask Status
EDTPTI (R/W)
Enable DMA TX Peripheral Triggered Interrupt Mask Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-13: UART_IMSK Register Diagram

Table 17-12: UART_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	ETXS	Enable TX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ETXS` bit indicates re-direction of the TX interrupt requests to status interrupt output. If cleared, TX interrupt requests are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked
8 (R/W)	ERXS	Enable RX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ERXS` bit indicates re-direction of RX interrupt requests to status interrupt output. If cleared, RX interrupt requests are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 17-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EAWI	Enable Address Word Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.EAWI` bit indicates generation of a status interrupt request when an Address word in MDB-mode is present in the `UART_RBR`. A received word is an address word if the `UART_STAT.ADDR` bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
6 (R/W)	ERFCI	Enable Receive FIFO Count Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ERFCI` bit indicates enabling of the receive buffer threshold interrupt request if signaled by the `UART_STAT.RFCS` bit. Read the `UART_RBR` register sufficient times to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
5 (R/W)	ETFI	Enable Transmission Finished Interrupt Mask Status. If set (interrupt unmasked) the `UART_IMSK.ETFI` bit indicates enabling of interrupt generation on the status interrupt channel when the transmit buffer register, the transmit address register, and the transmit shift register are all empty as indicated by the `UART_STAT.TFI`. The `UART_IMSK.ETFI` interrupt can be used to avoid expensive polling of the `UART_STAT.TEMT` bit, when the UART clock or line drivers should be disabled after transmission has completed. W1C the `UART_STAT.TFI` bit to clear the interrupt request. In DMA operation, the `UART_IMSK.ETFI` bits functionality might be preferred.	
		0	Interrupt is masked
		1	Interrupt is unmasked
4 (R/W)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.EDTPTI` bit indicates enabling of the DMA completion interrupt request to be delayed until the data has left the UART completely. This bit is required for DMA transmit operation only. If set, the UART can generate a DMA interrupt request by the time the `UART_STAT.TEMT` bit goes high after the last DMA data word is transmitted. When `UART_IMSK.EDTPTI` is set, usually the `DMA_CFG.INT` field is cleared to 00 in a STOP mode DMA. This set up suppresses the normal completion interrupt request, and the `UART_STAT.TEMT` event is signaled through the DMA controller and triggers the DMA interrupt. If both (`DMA_CFG.INT` not 00 and `UART_IMSK.EDTPTI` set), two interrupts are requested at the end of a STOP mode DMA.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 17-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EDSSI	Enable Modem Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.EDSSI` bit indicates enabling of a modem status interrupt request on the same status interrupt channel when the `UART_STAT.SCTS` bit is set. This indicates `UART_CTS` pin re-assertion. Write-1-to-clear (W1C) the `UART_STAT.SCTS` bit to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
2 (R/W)	ELSI	Enable Line Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ELSI` bit indicates that redirection of TX and RX interrupt requests to the status interrupt output of the UART by OR'ing them with the `UART_STAT.OE`, `UART_STAT.PE`, `UART_STAT.FE`, and `UART_STAT.BI` interrupt requests. Set this bit when no DMA channel is associated with the UART. Enabling `UART_IMSK.ELSI` disables the RX/TX interrupt channels and negates the `UART_IMSK.EDTPTI` bit.	
		0	Interrupt is masked
		1	Interrupt is unmasked
1 (R/W)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ETBEI` bit indicates generation of a TX interrupt request if the `UART_STAT.THRE` bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
0 (R/W)	ERBFI	Enable Receive Buffer Full Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ERBFI` indicates generation of an RX interrupt request if the `UART_STAT.DR` bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Interrupt Mask Clear Register

The UART_IMSK indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupt requests, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

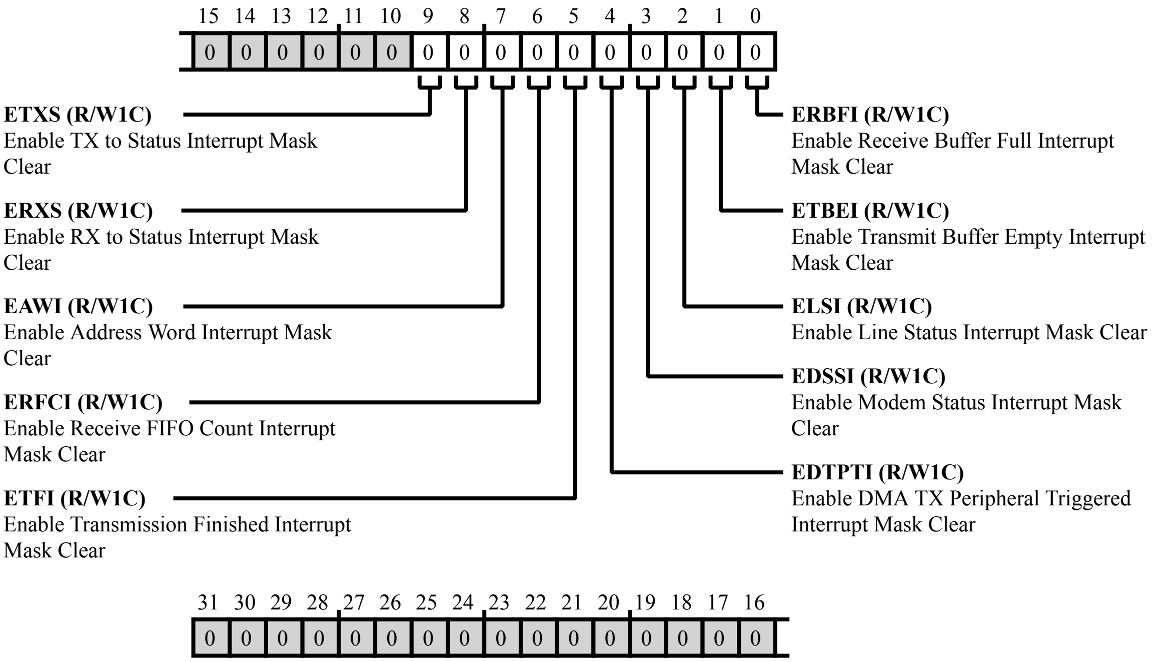

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ETXS (R/W1C)
Enable TX to Status Interrupt Mask Clear
ERXS (R/W1C)
Enable RX to Status Interrupt Mask Clear
EAWI (R/W1C)
Enable Address Word Interrupt Mask Clear
ERFCI (R/W1C)
Enable Receive FIFO Count Interrupt Mask Clear
ETFI (R/W1C)
Enable Transmission Finished Interrupt Mask Clear
ERBFI (R/W1C)
Enable Receive Buffer Full Interrupt Mask Clear
ETBEI (R/W1C)
Enable Transmit Buffer Empty Interrupt Mask Clear
ELSI (R/W1C)
Enable Line Status Interrupt Mask Clear
EDSSI (R/W1C)
Enable Modem Status Interrupt Mask Clear
EDTPTI (R/W1C)
Enable DMA TX Peripheral Triggered Interrupt Mask Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-14: UART_IMSK_CLR Register Diagram

Table 17-13: UART_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ETXS	Enable TX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
8 (R/W1C)	ERXS	Enable RX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
7 (R/W1C)	EAWI	Enable Address Word Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Table 17-13: UART_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	ERFCI	Enable Receive FIFO Count Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
5 (R/W1C)	ETFI	Enable Transmission Finished Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
4 (R/W1C)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
3 (R/W1C)	EDSSI	Enable Modem Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
2 (R/W1C)	ELSI	Enable Line Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
1 (R/W1C)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
0 (R/W1C)	ERBFI	Enable Receive Buffer Full Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Interrupt Mask Set Register

The UART_IMSK indicates interrupt request mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupt requests, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

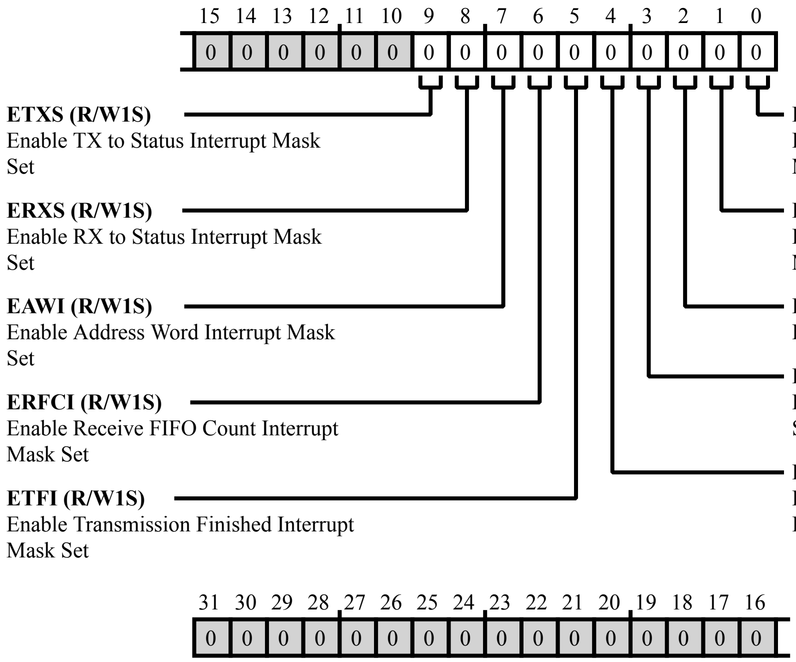

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ETXS (R/W1S)
Enable TX to Status Interrupt Mask Set
ERXS (R/W1S)
Enable RX to Status Interrupt Mask Set
EAWI (R/W1S)
Enable Address Word Interrupt Mask Set
ERFCI (R/W1S)
Enable Receive FIFO Count Interrupt Mask Set
ETFI (R/W1S)
Enable Transmission Finished Interrupt Mask Set
ERBFI (R/W1S)
Enable Receive Buffer Full Interrupt Mask Set
ETBEI (R/W1S)
Enable Transmit Buffer Empty Interrupt Mask Set
ELSI (R/W1S)
Enable Line Status Interrupt Mask Set
EDSSI (R/W1S)
Enable Modem Status Interrupt Mask Set
EDTPTI (R/W1S)
Enable DMA TX Peripheral Triggered Interrupt Mask Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-15: UART_IMSK_SET Register Diagram

Table 17-14: UART_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1S)	ETXS	Enable TX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
8 (R/W1S)	ERXS	Enable RX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
7 (R/W1S)	EAWI	Enable Address Word Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Table 17-14: UART_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	ERFCI	Enable Receive FIFO Count Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
5 (R/W1S)	ETFI	Enable Transmission Finished Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
4 (R/W1S)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
3 (R/W1S)	EDSSI	Enable Modem Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
2 (R/W1S)	ELSI	Enable Line Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
1 (R/W1S)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
0 (R/W1S)	ERBFI	Enable Receive Buffer Full Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Receive Buffer Register

The read-only `UART_RBR` register is the UART's receive buffer. It is updated when there is pending data in the receive FIFO. Newly available data is signaled by the `UART_STAT.DR` bit.

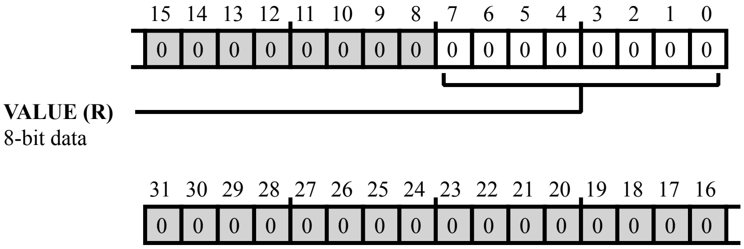

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
8-bit data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-16: UART_RBR Register Diagram

Table 17-15: UART_RBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	8-bit data.

Receive Shift Register

The read only UART_RSR register which returns the content of the UART's receive shift register.

The frame data is moved into this shift register after polarity inversion, if any (including the native polarity inversion in the IrDA case).

In the case of the longest frame (MDB, with parity mode, and 8 bit data word-length), the start bit may be shifted out and not available for reading at the end of the frame reception. This register is NOT reset at the start of frame. If read, in the middle of a frame reception, data corresponding the previous frame may not have entirely shifted out (for example, the read data that have been read may NOT correspond entirely to the frame being received).

Because the UART is receiving only 1 stop bit, the UART_RSR contains only 1 stop bit even if more than one stop bit is present in the actual transfer. This register may be considered as storing the 10 most recently received bits (taking into consideration the stop bit receive limitation above).

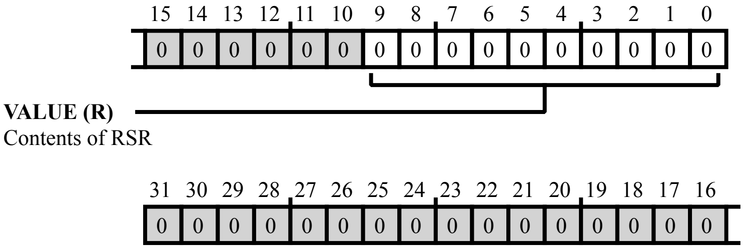


Figure 17-17: UART_RSR Register Diagram

Table 17-16: UART_RSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	VALUE	Contents of RSR.

Receive Counter Register

The `UART_RXCNT` register returns the content of 16-bit counter in the UART receiver. This count is used for baud rate clock generation (the lower [15:0] is the count data).

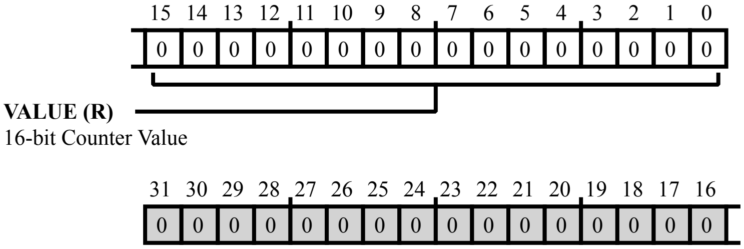

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
16-bit Counter Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-18: UART_RXCNT Register Diagram

Table 17-17: UART_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

Scratch Register

The `UART_SCR` registers contain 8-bit scratch pad data. These registers are used for general purpose data storage and do not control the UART hardware in any way.

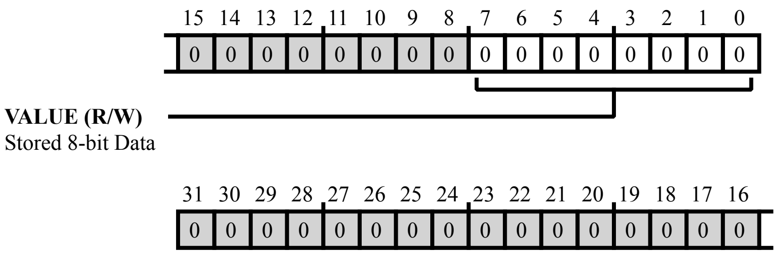

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Stored 8-bit Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-19: UART_SCR Register Diagram

Table 17-18: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Stored 8-bit Data.

Status Register

The UART_STAT register contains the UART line status and UART modem status, as indicated by the current states of the UART's UART_CTS pin and internal receive buffers. Writes to this register can perform write-one-to-clear (W1C) operations on most status bits. Reading this register has no side effects.

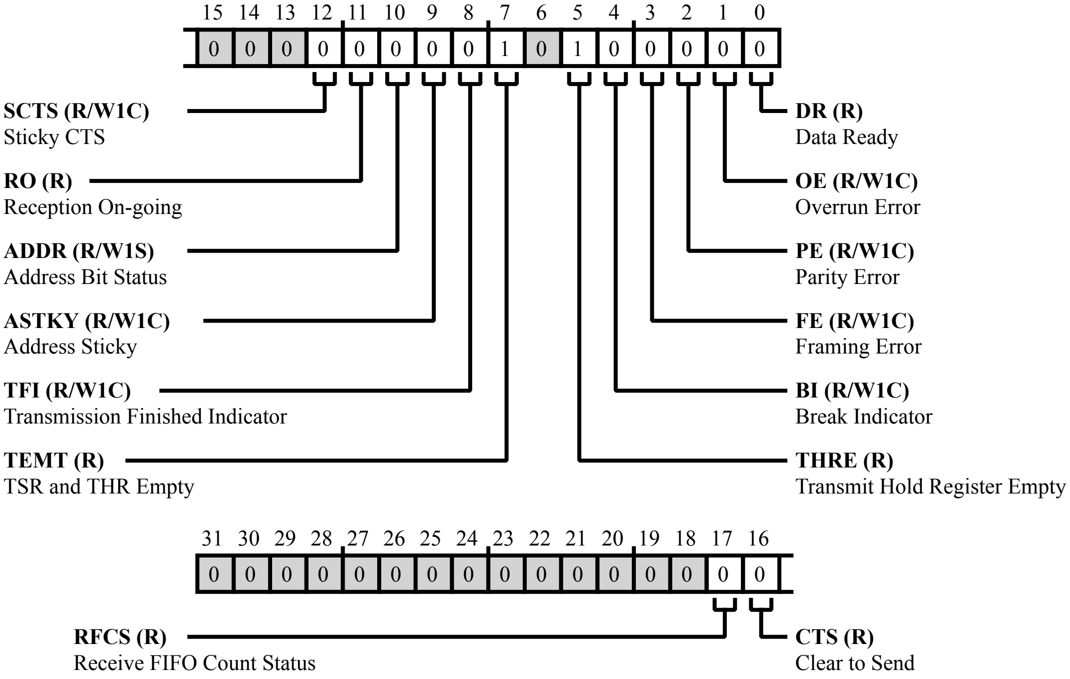

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
SCTS (R/W1C)
Sticky CTS
RO (R)
Reception On-going
ADDR (R/W1S)
Address Bit Status
ASTKY (R/W1C)
Address Sticky
TFI (R/W1C)
Transmission Finished Indicator
TEMT (R)
TSR and THR Empty
DR (R)
Data Ready
OE (R/W1C)
Overrun Error
PE (R/W1C)
Parity Error
FE (R/W1C)
Framing Error
BI (R/W1C)
Break Indicator
THRE (R)
Transmit Hold Register Empty
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RFCS (R)
Receive FIFO Count Status
CTS (R)
Clear to Send

Figure 17-20: UART_STAT Register Diagram

Table 17-19: UART_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	RFCS	Receive FIFO Count Status. The UART_STAT.RFCS bit is set when the receive buffer holds more or equal entries than a certain threshold. The threshold is controlled by the UART_CTL.RFIT bit. If UART_CTL.RFIT is cleared, the threshold is four entries. If UART_CTL.RFIT is set, the threshold is seven entries. The UART_STAT.RFCS bit is cleared when the UART_RBR register is read sufficient times until the buffer is drained below the threshold. The UART_STAT.RFCS bit can trigger a status interrupt request if enabled by the UART_IMSK_SET.ERFCI bit.	
		0	RX FIFO has less than 4 (7) entries when RFIT=0 (1)
		1	RX FIFO has at least 4 (7) entries when RFIT=0 (1)

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	CTS	Clear to Send. The `UART_STAT.CTS` bit holds the value (if `UART_CTL.FCPOL` set) or the complement value (if `UART_CTL.FCPOL` cleared) of the `UART_CTS` input pin. The `UART_CTL.ACTS` bit must be set to enable this feature. The core can read the value of the `UART_STAT.CTS` bit to determine whether the external device is ready to receive (`UART_STAT.CTS` set) or if it is busy (`UART_STAT.CTS` cleared). If `UART_CTL.ACTS` is cleared, the `UART_TX` handshaking protocol is disabled, and the UART transmits data as long as there is data to transmit, regardless of the value of `UART_STAT.CTS`. When `UART_CTL.ACTS` is cleared, the software can pause transmission temporarily by setting the XOFF bit. Note that in loopback mode (`UART_CTL.LOOP_EN` set), the `UART_STAT.CTS` bit is disconnected from the `UART_CTS` input pin. Instead, the bit is directly connected to the `UART_CTL.MRTS` bit.	
		0	Not clear to send (External device not ready to receive)
		1	Clear to send (External device ready to receive)
12 (R/W1C)	SCTS	Sticky CTS. The `UART_STAT.SCTS` bit is a sticky bit that is set when `UART_STAT.CTS` transitions from 0 to 1. The `UART_STAT.SCTS` bit is cleared by software with a W1C operation. This bit can trigger a line status interrupt request if enabled by the `UART_IMSK_SET.EDSSI` bit.	
		0	CTS has not transitioned from low to high
		1	CTS has transitioned from low to high
11 (R/NW)	RO	Reception On-going.	
		0	No data reception in progress
		1	Data reception in progress
10 (R/W1S)	ADDR	Address Bit Status. The `UART_STAT.ADDR` bit is used to mirror the address bit of the word in `UART_RBR` in multi-drop bus protocol, and is enabled only in MDB mode. The `UART_STAT.ADDR` bit is updated by hardware upon detecting a received word with the address bit in `UART_RBR` set or cleared. Additionally, software can set the ADDR bit with a write-1-to-set (W1S) operation.	
		0	Address bit is low
		1	Address bit is high

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ASTKY	Address Sticky. The `UART_STAT.ASTKY` bit is used in multi-drop bus mode to indicate whether a peripheral is currently being addressed. This bit is a sticky version of the `UART_STAT.ADDR` bit and is set by hardware when setting the `UART_STAT.ADDR` bit. The `UART_STAT.ASTKY` bit can only be cleared by software with a write-one-to-clear (W1C) operation. With the `UART_STAT.ASTKY` bit set, words will be received irrespective of the `UART_CTL.MOD` bit or `UART_STAT.ADDR` bit selection. With the `UART_STAT.ASTKY` bit cleared, only address words (`UART_CTL.MOD` bit set) will be received and words with `UART_CTL.MOD` bit cleared are ignored (not moved from the RSR to the RX FIFO) in MDB mode. The `UART_STAT.ASTKY` bit does not affect reception in non-MDB modes.	
		0	ADDR bit has not been set
		1	ADDR bit has been set
8 (R/W1C)	TFI	Transmission Finished Indicator. The `UART_STAT.TFI` bit is a sticky version of the `UART_STAT.TEMT` bit. While `UART_STAT.TEMT` is automatically cleared by hardware when new data is written to the `UART_THR` register, the sticky `UART_STAT.TFI` bit remains set, until it is cleared by software (W1C). The `UART_STAT.TFI` bit enables more flexible transmit interrupt request timing.	
		0	TEMT did not transition from 0 to 1
		1	TEMT transition from 0 to 1
7 (R/NW)	TEMT	TSR and THR Empty. The `UART_STAT.TEMT` bit indicates that the `UART_THR` and `UART_TAIP` registers and the `UART_TSR` register are empty. In this case, the program is permitted to write to the `UART_THR` and `UART_TAIP` registers twice without losing data. The `UART_STAT.TEMT` bit can also be used as indicator that pending UART transmission is completed. At that time, it is safe to disable the `UART_CTL.EN` bit or to three-state the off-chip line driver.	
		0	Not empty TSR/THR
		1	TSR/THR Empty

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	THRE	Transmit Hold Register Empty. The UART_STAT.THRE bit indicates that the UART transmit channel is ready for new data and software can write to the UART_THR and UART_TAIP registers. Writes to the UART_THR and UART_TAIP registers clear the UART_STAT.THRE. The bit is set again when the UART_THR and UART_TAIP registers are empty and ready to accept data.	
		0	Not empty THR/TAIP
		1	Empty THR/TAIP
4 (R/W1C)	BI	Break Indicator. The UART_STAT.BI bit indicates that the first stop bit is sampled low and the entire data word, including parity bit, consists of low bits only. (This condition indicates that UART_RX was held low for more than the maximum word length.) The UART_STAT.BI bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The bit is sticky and can be cleared by W1C operations.	
		0	No break interrupt
		1	Break interrupt this indicates UARTxRX was held low(RPOLC=0) / high (RPOLC=1) for more than the maximum word length
3 (R/W1C)	FE	Framing Error. The UART_STAT.FE bit indicates that the first stop bit is sampled. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.FE bit is sticky and can be cleared by W1C operations. Note that invalid stop bits can be simulated by setting the UART_CTL.FFE bit.	
		0	No error
		1	Invalid stop bit error
2 (R/W1C)	PE	Parity Error. The UART_STAT.PE bit indicates that the received parity bit does not match the expected value. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.PE bit is sticky and can be cleared by W1C operations. Note that invalid parity bits can be simulated by setting the UART_CTL.FPE bit.	
		0	No parity error
		1	Parity error

Table 17-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	OE	Overrun Error. The `UART_STAT.OE` bit indicates that further data is received while the internal receive buffer was full. This bit is set when sampling the stop bit of the sixth data word. To avoid overruns, read the `UART_RBR` register in time. In DMA receive mode, overruns are very unlikely to happen ever. After an overrun occurs, the `UART_RBR` and receive FIFO are protected from being overwritten by new data until the `UART_STAT.OE` bit is cleared by software. The content of the `UART_RSR` register is lost as soon as the overrun occurs. The `UART_STAT.OE` bit is sticky and can be cleared by W1C operations.	
		0	No overrun
		1	Overrun error
0 (R/NW)	DR	Data Ready. The `UART_STAT.DR` bit indicates that data is available in the receiver and can be read from the `UART_RBR` register. The bit is set by hardware when the receiver detects the first valid stop bit. The bit is cleared by hardware when the `UART_RBR` register is read.	
		0	No new data
		1	New data in RBR

Transmit Address/Insert Pulse Register

The `UART_TAIP` register and the `UART_THR` register share the same physical register, but `UART_TAIP` has different effect than the `UART_THR` register when `UART_TAIP` is written to in MDB and UART modes.

In MDB mode, data written to the `UART_TAIP` register is transmitted as an address frame (as with the `UART_CTL.MOD` bit set).

In UART mode, a write to `UART_TAIP` causes a pulse of value `UART_TAIP` [7] for a duration of `UART_TAIP` [6:0] x bit time. (There is additional inversion if the `UART_CTL.TPOLC` bit is set).

Bit time is defined by the `UART_CLK` register. The transmission of the pulse is followed by stop bit transmission as specified by the `UART_CTL.STB` and `UART_CTL.STBH` bits. This could be used for supporting line break command and inter-frame gap.

In IrDA mode, writes to `UART_TAIP` is treated the same as writes to `UART_THR`.

Accesses to the `UART_TAIP` register have the same affects as the `UART_THR` register with respect to the `UART_STAT.THRE`, `UART_STAT.TEMT`, and `UART_STAT.TFI` flags.

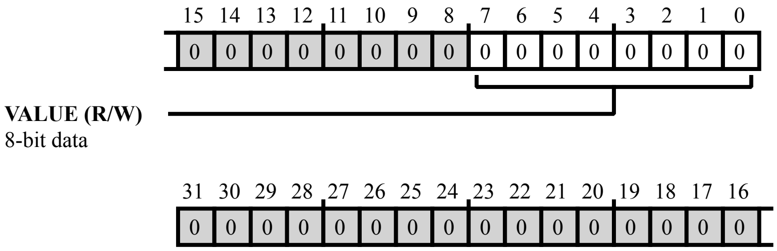


Figure 17-21: UART_TAIP Register Diagram

Table 17-20: UART_TAIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8-bit data.

Transmit Hold Register

The write-only `UART_THR` register is the UART's transmit buffer. The `UART_STAT.THRE` bit indicates whether data can be written to `UART_THR`. Writes to this register automatically propagate to the internal `UART_TSR` register as soon as `UART_TSR` is ready. Then, transmit operation is initiated immediately.

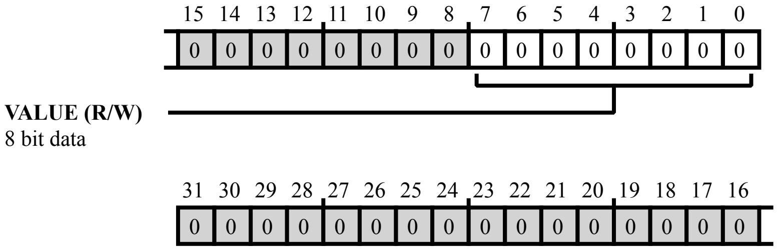

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
8 bit data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-22: UART_THR Register Diagram

Table 17-21: UART_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8 bit data.

Transmit Shift Register

The read only `UART_TSR` register which returns the content of the UART's transmit shift register.

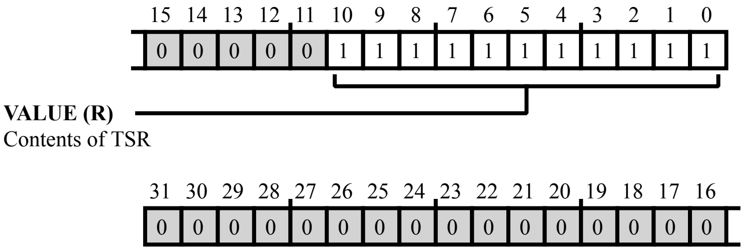

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
VALUE (R)
Contents of TSR
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-23: UART_TSR Register Diagram

Table 17-22: UART_TSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Contents of TSR.

Transmit Counter Register

The `UART_TXCNT` read only register returns the content of 16-bit counter in the UART transmitter. This count is used for baud rate clock generation (the lower [15:0] is the count data).

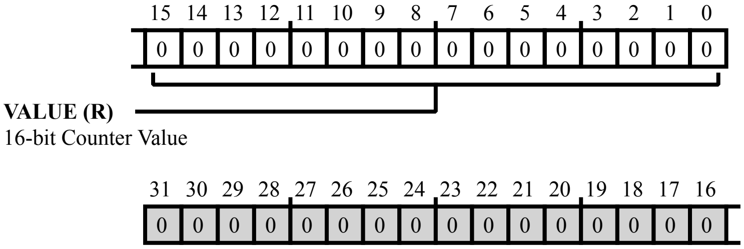

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
16-bit Counter Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-24: UART_TXCNT Register Diagram

Table 17-23: UART_TXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

18 Enhanced Parallel Peripheral Interface (EPPI)

The Enhanced Parallel Peripheral Interface (EPPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins. It can support direct connections to active TFT LCDs, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. Each EPPI has two DMA channels associated with it. Moreover, in some modes, an EPPI can use an extra DMA channel.

EPPI Features

The EPPI module supports the following features.

- Programmable data length from 8 bits to up to 24 bits per clock cycle (depending on the product model)
- Bidirectional and half-duplex port
- Internal or external clock source
- Clock gating by an external device asserting the clock gating control signal
- Various framed and non-framed operating modes, as well as internal or external frame syncs
- Various general-purpose modes with 0, 1, 2, and 3 frame sync modes for both receive and transmit
- Ignores premature external frame syncs for data consistency
- SMPTE274M and SMPTE 296M high definition format support
- ITU-656, SMPTE 296M and SMPTE 274M status word error detection and correction for ITU-656 receive modes
- ITU-656, SMPTE 296M and SMPTE 274M receive modes – active video only, vertical blanking only, and entire field
- ITU-656, SMPTE 296M and SMPTE 274M preamble and status word decode
- Optional packing and unpacking of data to or from 32 bits from or to 8, 16 bits and 24 bits. If packing or unpacking is enabled, endianness can be altered to change the order of packing or unpacking of bytes/words
- Optional sign extension or zero-fill and alternate even or odd data sample filter for receive modes

- RGB888 to RGB666 or RGB565 conversion for transmit modes
- 4:2:2 YCrCb data Tx/Rx interleaving or de-interleaving modes
- Configurable LCD data enable (DEN) output available on frame sync 3
- Delayed start of PPI frame syncs
- Data clipping and mirroring
- Horizontal and vertical windowing for general purpose 2 and 3 frame sync modes
- Preamble, blanking and stripping support
- Multiplexing dual input

EPPI Functional Description

The EPPI has the following functionality.

RGB data formats

For transmit modes, the EPPI can convert RGB888 data in memory to either RGB565 or RGB666 at the output using bits in the Control register.

Data clipping

The EPPI contains two registers to define the lower and upper limits for the Luma and Chroma components. This functionality is used for clipping data values during 8-bit, 10-bit, 12-bit or 16-bit transmit modes.

Data mirroring

A data mirroring feature is available which mirrors the EPPI data bits 15–0. This functionality is available in both transmit and receive modes.

Windowing

The EPPI supports windowing for general-purpose input modes.

Preamble, blanking and stripping support

The EPPI can embed blanking information and clip active data to be transmitted. This functionality is available for single channel data, interleaved data, and parallel data and supports data lengths equal to 16 bits, 20 bits or 24 bits.

ADSP-SC58x EPPI Register List

The EPPI is a half-duplex, bidirectional parallel port. It comprises a clock pin, 3 frame sync pins, and a set of data pins. For more information on EPPI functionality, see the EPPI register descriptions.

Table 18-1: ADSP-SC58x EPPI Register List

Name	Description
`EPPI_CLKDIV`	Clock Divide Register
`EPPI_CTL`	Control Register
`EPPI_CTL2`	Control Register 2 Register
`EPPI_EVENCLIP`	Clipping Register for EVEN (Luma) Data Register
`EPPI_FRAME`	Lines Per Frame Register
`EPPI_FS1_DLY`	Frame Sync 1 Delay Value Register
`EPPI_FS1_PASPL`	FS1 Period Register / EPPI Active Samples Per Line Register
`EPPI_FS1_WLHB`	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
`EPPI_FS2_DLY`	Frame Sync 2 Delay Value Register
`EPPI_FS2_PALPF`	FS2 Period Register / EPPI Active Lines Per Field Register
`EPPI_FS2_WLVB`	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
`EPPI_HCNT`	Horizontal Transfer Count Register
`EPPI_HDLY`	Horizontal Delay Count Register
`EPPI_IMSK`	Interrupt Mask Register
`EPPI_LINE`	Samples Per Line Register
`EPPI_ODDCLIP`	Clipping Register for ODD (Chroma) Data Register
`EPPI_STAT`	Status Register
`EPPI_VCNT`	Vertical Transfer Count Register
`EPPI_VDLY`	Vertical Delay Count Register

ADSP-SC58x EPPI Interrupt List

Table 18-2: ADSP-SC58x EPPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
81	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Level	28
82	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Level	29
83	EPPI0_STAT	EPPI0 Status	Level	
215	EPPI0_CH0_DMA_ERR	EPPI0 DMA Channel 0 Error		

Table 18-2: ADSP-SC58x EPPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
216	EPPI0_CH1_DMA_ERR	EPPI0 DMA Channel 1 Error		

ADSP-SC58x EPPI Trigger List

Table 18-3: ADSP-SC58x EPPI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
62	`EPPI0_CH0_DMA`	EPPI0 Channel 0 DMA	Edge
63	`EPPI0_CH1_DMA`	EPPI0 Channel 1 DMA	Edge

Table 18-4: ADSP-SC58x EPPI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
46	`EPPI0_CH0_DMA`	EPPI0 Channel 0 DMA	Pulse
47	`EPPI0_CH1_DMA`	EPPI0 Channel 1 DMA	Pulse

RGB Data Formats

For transmit modes, the EPPI can convert RGB888 data in memory to RGB666 at the output when the `EPPI_CTL.RGBFMTEN` bit is set and the `EPPI_CTL.DLEN` value is equal to 18 bits. Similarly, the EPPI can convert RGB888 data in memory to RGB565 at the output when the `EPPI_CTL.RGBFMTEN` bit is set and `EPPI_CTL.DLEN` is equal to 16 bits.

This conversion is performed as follows:

- If `EPPI_CTL.PACKEN` =1, the EPPI first unpacks according to the `EPPI_CTL.SWAPEN` bit setting, and the three 32-bit data words from the DMA are broken into four 24-bit data words to be transmitted out, as described earlier.
- If `EPPI_CTL.PACKEN` =0, the EPPI takes the lower 24 bits of the 32-bit DMA as the data to be transmitted. Then, the EPPI truncates this 24-bit data word to the required data width. It removes the lower 2 bits of G and the lower 2 bits or 3 bits of R and B.

Data Clipping

The EPPI contains two registers to define the lower and upper limits for the Luma and Chroma components. It uses these registers for clipping data values during 8-bit, 10-bit, 12-bit or 16-bit transmit modes. All data values for odd samples which are less than the value in the `EPPI_ODDCLIP.LOWODD` bit field are replaced with the value in the `EPPI_ODDCLIP.LOWODD` field. All data values for even samples which are less than the value in the `EPPI_EVENCLIP.LOWEVEN` field are replaced with the value in the `EPPI_EVENCLIP.LOWEVEN` field.

In the same manner, all data values for odd samples which are more than the value in the `EPPI_ODDCLIP.HIGHODD` bit field are replaced with the value in the `EPPI_ODDCLIP.HIGHODD` field. All data values for even samples which are more than the values in the `EPPI_EVENCLIP.HIGHEVEN` field are replaced with the values in the `EPPI_EVENCLIP.HIGHEVEN` field.

Depending on the programmed EPPI length, only the corresponding bits (least aligned) are considered for clipping. For example, if the EPPI is programmed in 10-bit mode, bits 9:0 and bits 25:16 constitute the clipping thresholds. The higher bits are ignored. The EPPI supports 8-bit, 10-bit, 12-bit, and 16-bit clipping thresholds.

For the 4:2:2 YCrCb color space, Luma and Chroma typically have different lower and upper thresholds. Separate thresholds can be required for even and odd data samples. For monochrome (Y only) or some non-video clipping applications, the value in the `EPPI_ODDCLIP.LOWODD` field can be the same as the value in the `EPPI_EVENCLIP.LOWEVEN` field. The value in the `EPPI_ODDCLIP.HIGHODD` field can be the same as the value in the `EPPI_EVENCLIP.HIGHEVEN` field.

In GP 0 FS mode with internal blanking generation, clipping is valid only for the active video part of the transmitted data. ITU-R 656 preambles, status words, and blanking data bypass the clipping logic.

If the EPPI is programmed in 16, 20-bit or 24-bit mode with the `EPPI_CTL.SPLTWRD` bit set, the YDATA (luma data) gets the clipping threshold levels of the `EPPI_EVENCLIP` register. The CDATA (chroma data) gets the clipping threshold levels of the `EPPI_ODDCLIP` register.

The clipping registers are ignored when the `EPPI_CTL.RGBFMTEN` bit is set.

Data Mirroring

To increase the pin multiplexing options for the EPPI data pins, a data mirroring feature is available which mirrors the EPPI data bits 15:0. This feature is available in both transmit and receive modes. It is enabled by setting the `EPPI_CTL.DMIRR` bit.

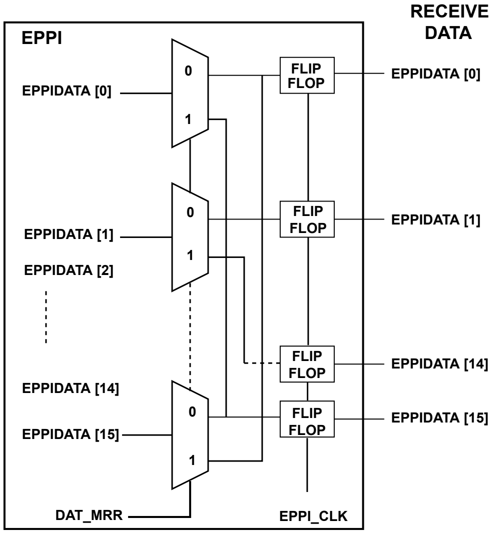


Figure 18-1: Data Mirroring Receive

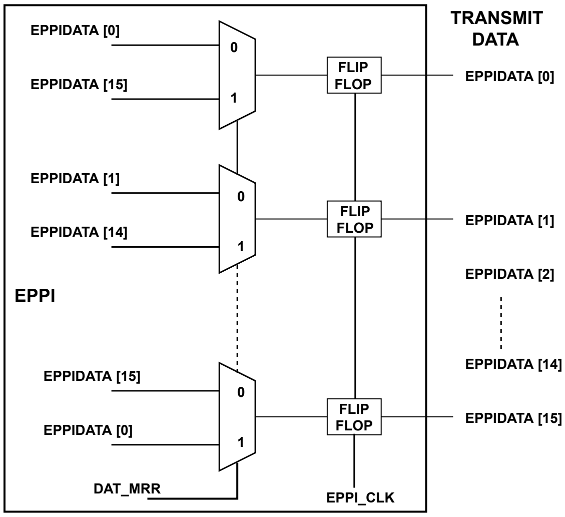

EPPIDATA [0]
EPPIDATA [15]
0
1
FLIP FLOP
TRANSMIT DATA
EPPIDATA [0]
EPPIDATA [1]
EPPIDATA [14]
0
1
FLIP FLOP
EPPIDATA [1]
EPPIDATA [2]
EPPI
EPPIDATA [14]
EPPIDATA [15]
EPPIDATA [0]
0
1
FLIP FLOP
EPPIDATA [15]
DAT_MRR
EPPI_CLK

Figure 18-2: Data Mirroring Transmit

Windowing

The EPPI supports windowing for general-purpose input modes. The module can be configured to bring in a region of interest instead of the entire frame of data, which helps reduce bandwidth requirements.

Preamble, Blanking and Stripping Support

The EPPI supports embedding blanking information and clipping of active data for transmission. This functionality is available for single-channel data, interleaved data, and parallel data and supports a data length (`EPPI_CTL.DLEN`) of 16, 20 bits or 24 bits.

Support of preamble generation or detection and stripping of blanking information is also provided for ITU656 mode and the SMPTE 274M and 296M HD formats. The ***Video Mode Comparison*** table shows the SMPTE standards in comparison with the ITU656 modes. Preambles for SMPTE and ITU656 modes are identical. Extension of preamble to 12 bits is also supported.

Table 18-5: Video Mode Comparison

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
ITU656 (NTSC)	30	1716x525	720x480	27.02	Y-C interleaved
ITU656 (PAL)	25	1728x625	720x576	27.00	Y-C interleaved
SMPTE 296M	30	3300x750	1280x720	74.25	Y,C separate
	60	1650x750	1280x720	74.25	Y,C separate

Table 18-5: Video Mode Comparison (Continued)

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
SMPTE 274M	30	2200x1125	1920x1080	74.25	Y,C separate
	60	2200x1125	1920x1080	148.50	Y,C separate
	25	2640x1125	1920x1080	74.25	Y,C separate
	50	2640x1125	1920x1080	148.50	Y,C separate
	24	2750x1125	1920x1080	74.25	Y,C separate

See the clock operating conditions section of the data sheet for the maximum sampling frequency for this product.

EPPI Definitions

The following definitions are helpful when using the EPPI module.

ITU-R BT.-656

Description of a digital video protocol for interfaces and data stream format required to send uncompressed PAL or NTSC standard definition TV (525 or 625 lines) signals.

YUV422

YUV is a color space where luminance (Y) and chrominance (UV) components define the pixels. The suffix signifies how the chrominance components have been decimated and provide formatting information. In this case, the YUV422 format has the chrominance decimated by two, meaning only half of each chrominance component is available. Typical YUV422 formatting interleaves the luminance and chrominance (for example, U1Y1V1Y2U2Y3V2Y4).

RGB888

RGB is a color space where three color values, one red (R), one green (G) and one blue (B), define the pixels. The suffix signifies the bit widths for these color components. In this case, RGB888 means that each red, green, and blue value is 8 bits.

RGB565

RGB is a color space where three color values, one red (R), one green (G) and one blue (B) define the pixels. The suffix signifies the bit widths for these color components. In this case, RGB565 means that the red (R) and blue (B) are 5 bits each while the green (G) is 6 bits. When packed together, each RGB565 pixel can be represented in a 16-bit data word. LCD display panels commonly use this format.

SMPTE 274M

An HD standard defining the spatial resolution (image sample structure) and frame rates for 1920x1080.

SMPTE 296M

An HD standard for defining the spatial resolution (image sample structure) and frame rates fro 1280x720.

EPPI Block Diagram

The *EPPI Block Diagram* figure shows the functional blocks within the EPPI.

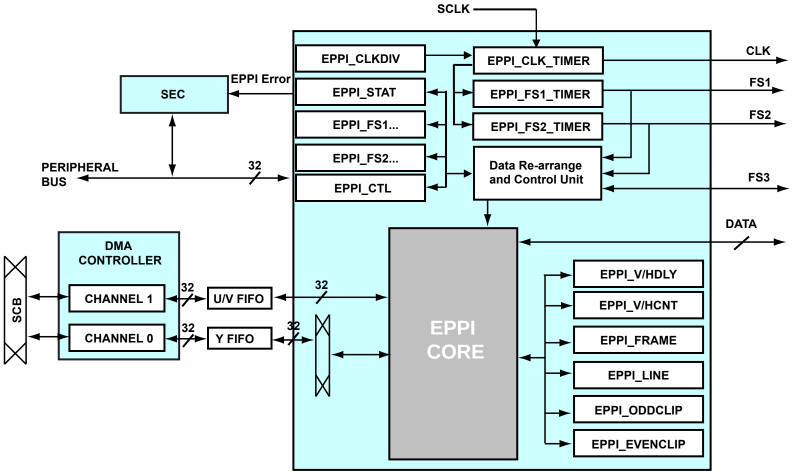

SCLK
EPPI_CLKDIV
EPPI_CLK_TIMER
CLK
SEC
EPPI Error
EPPI_STAT
EPPI_FS1_TIMER
FS1
EPPI_FS1...
EPPI_FS2_TIMER
FS2
PERIPHERAL BUS
32
EPPI_FS2...
Data Re-arrange and Control Unit
EPPI_CTL
FS3
DATA
DMA CONTROLLER
SCB
CHANNEL 1
32
U/V FIFO
32
CHANNEL 0
32
Y FIFO
32
EPPI CORE
EPPI_V/HDLY
EPPI_V/HCNT
EPPI_FRAME
EPPI_LINE
EPPI_ODDCLIP
EPPI_EVENCLIP

Figure 18-3: EPPI Block Diagram

EPPI Architectural Concepts

The following sections describe the architectural concepts.

- EPPI Interface
- Reset Operation
- Frame Sync Polarity and Sampling Edge
- Direct Memory Access (DMA)
- EPPI Clock

EPPI Interface

A block diagram of the architecture for the EPPI interface is shown in the *EPPI DMA Interface* figure.

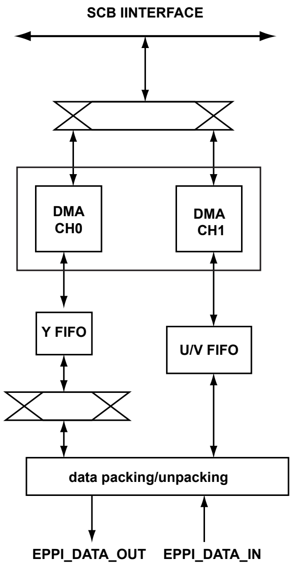

SCB IINTERFACE
DMA CH0
DMA CH1
Y FIFO
U/V FIFO
data packing/unpacking
EPPI_DATA_OUT
EPPI_DATA_IN

Figure 18-4: EPPI DMA Interface

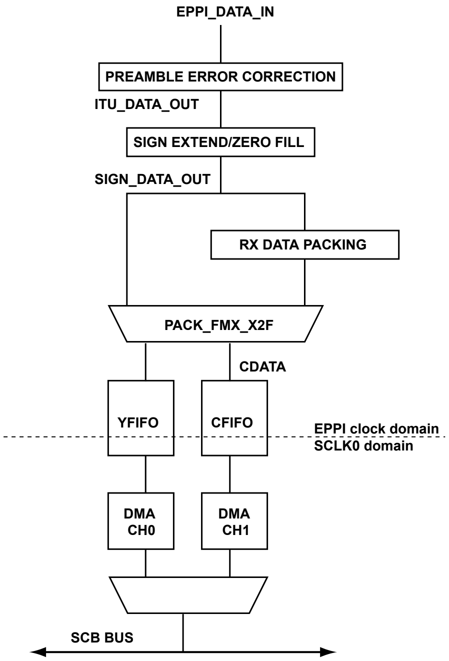

EPPI_DATA_IN
PREAMBLE ERROR CORRECTION
ITU_DATA_OUT
SIGN EXTEND/ZERO FILL
SIGN_DATA_OUT
RX DATA PACKING
PACK_FMX_X2F
CDATA
YFIFO
CFIFO
EPPI clock domain
SCLK0 domain
DMA CH0
DMA CH1
SCB BUS

Figure 18-5: Receive Data Path

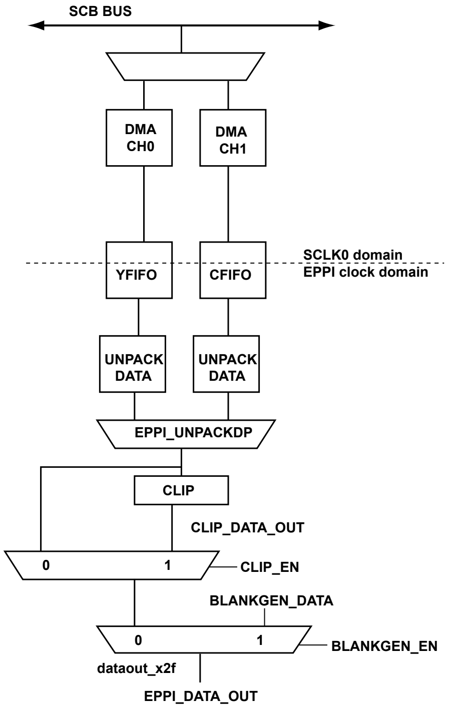

SCB BUS
DMA CH0
DMA CH1
YFIFO
CFIFO
SCLK0 domain
EPPI clock domain
UNPACK DATA
UNPACK DATA
EPPI_UNPACKDP
CLIP
CLIP_DATA_OUT
0
1
CLIP_EN
BLANKGEN_DATA
0
1
BLANKGEN_EN
dataout_x2f
EPPI_DATA_OUT

Figure 18-6: Transmit Data Path

Reset Operation

On a hardware reset, the entire EPPI is reset. All MMRs return to their default values. EPPI interrupt and DMA requests become inactive and internally generated `EPPI_CLK` and frame syncs are aborted.

In software, write 0 to the `EPPI_CTL.EN` bit to reset and reconfigure the EPPI. When disabled in this manner, only the `EPPI_STAT` register is cleared to its reset value. Interrupts and DMA requests become inactive and internally generated clock and frame syncs are aborted.

Frame Sync Polarity and Sampling Edge

The `EPPI_CTL.POLS` and `EPPI_CTL.POLC` bits provide a mechanism to select the active level of the frame syncs and the sampling or driving edge of the EPPI clock, respectively. This functionality allows the EPPI to connect to data sources and receivers with a wide array of control signal polarities. Often, the remote data source or receiver also offer configurable signal polarities. In these cases, the `EPPI_CTL.POLS` and `EPPI_CTL.POLC` bits add flexibility.

Table 18-6: Frame Sync Polarity Selections and Frame Sync Pin States

Bit Setting	Frame Sync 2	Frame Sync 1
POLS = b#00	Active high	Active high
POLS = b#01	Active high	Active low
POLS = b#10	Active low	Active high
POLS = b#11	Active low	Active low

Table 18-7: Clock Polarity Selections and Receive/Transmit Pin States

	Receive		Transmit	
Bit Setting	Sample Data	Sample/Drive Syncs	Drive Data	Sample/Drive Syncs
POLC = b#00	Falling edge	Falling edge	Rising edge	Rising edge
POLC = b#01	Falling edge	Rising edge	Rising edge	Falling edge
POLC = b#10	Rising edge	Falling edge	Falling edge	Rising edge
POLC = b#11	Rising edge	Rising edge	Falling edge	Falling edge

Direct Memory Access (DMA)

The EPPI has a native DMA controller with two channels. A local arbiter arbitrates between these channels and requests are forwarded to the system crossbar. The EPPI has one connection to the fabric.

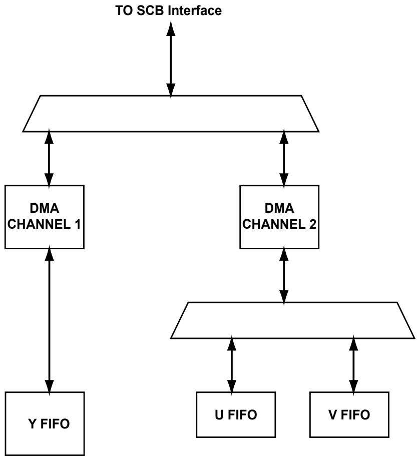


Figure 18-7: EPPI DMA Interface

The EPPI must be used with DMA. Configuring the EPPI DMA channels is a necessary step toward using the EPPI interface. The channels can be configured for either transmit or receive operation, and have a maximum throughput of (`EPPI_CLK`) x (32 bits/transfer). In modes where data lengths permit, packing can increase transfer bandwidth.

The DMA engine generates interrupts at the completion of a row, frame, or partial-frame transfer. The DMA engine also coordinates the source or destination point for the data that is transferred through the EPPI.

The 2D DMA capability allows the processor to be interrupted at the end of a line or after a frame of video is transferred, or if a DMA error occurs. The `DMA_XCNT` and `DMA_YCNT` registers allow for flexible data interrupt points. For example, assume the `DMA_XMOD` = `DMA_YMOD` =1. If a data frame contains 320 × 240 bytes (240 rows of 320 bytes each), the following conditions hold.

- Setting `DMA_XCNT` =320, `DMA_YCNT` = 240, and `DMA_CFG.INT` =1 interrupts on every row transferred, for the entire frame.
- Setting `DMA_XCNT` =320, `DMA_YCNT` =240, and `DMA_CFG.INT` =2 interrupts only on the completion of the frame (when 240 rows of 320 bytes have been transferred).

- Setting `DMA_XCNT` =38,400 (320 x 120), `DMA_YCNT` =2, and `DMA_CFG.INT` =1 causes an interrupt when half of the frame is transferred, and again when the whole frame is transferred.

The following is the general procedure for setting up DMA operation with the EPPI.

1. Configure the DMA registers as appropriate for the desired DMA operating mode.
2. Enable the DMA channel for operation.
3. Configure appropriate EPPI registers.
4. Enable the EPPI by writing 1 to the `EPPI_CTL.EN` bit.

EPPI Clock

The EPPI can be supplied with an external clock, or the clock can be generated internally and supplied to external devices. For information on the maximum PPI_CLK specification in internal and external clock modes, see the product-specific data sheet.

When using an external `EPPI_CLK`, there can be up to two cycles latency before valid data is received or transmitted.

The internal clock can be generated from SCLK1_0 when the `EPPI_CTL.ICLKGEN` bit is set. The value in the `EPPI_CLKDIV` register determines the generated clock frequency. The internally generated EPPI clock frequency is:

$f_{PCLK} = f_{SCLK0}/(\texttt{EPPI\_CLKDIV} + 1)$

where:

f_{PCLK} – frequency of internally generated EPPI clock

f_{SCLK} – frequency of SCLK1_0

`EPPI_CLKDIV` – Clock division value programmed in the `EPPI_CLKDIV` register.

The ***Relationship Between CLKDIV and the Ratio of SCLK0 to EPPI Clock*** table gives a few examples.

Table 18-8: Relationship Between CLKDIV and the Ratio of SCLK0 to EPPI Clock

CLKDIV15–0	EPPI/SCLK0 Clock Ratio
0x0002	1:3
0x0003	1:4
0x0004	1:5
0x0005	1:6
...	...

EPPI Operating Modes

The EPPI supports various receive and transmit modes of operation which include the detection and generation of preamble data. Specifically, the EPPI supports data formats described in the specifications ITU656, SMPTE 274M and SMPTE 296M. In addition to these modes, the EPPI also supports general-purpose receive and transmit using up to three frame syncs (FS).

The control register (`EPPI_CTL`) includes most of the bits used for configuring operating modes. The “Register Descriptions” section of this chapter provides complete descriptions of these bits.

ITU-R 656 Modes

The EPPI supports three input modes and one output mode for ITU-R 656 framed data. This section describes these modes.

ITU-R 656 Background

In ITU-R 656 mode, the horizontal (H), vertical (V), and field (F) signals are sent as an embedded part of the video data stream. The signals are sent in a series of bytes that form a control word. ITU-R 656 was formerly known as CCIR-656.

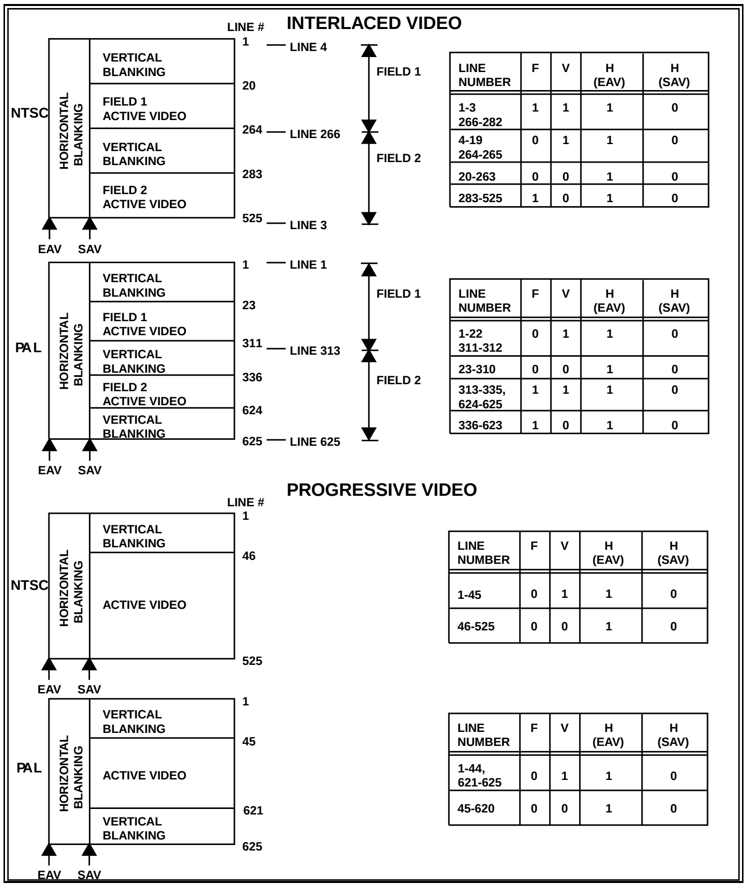
The letter H is used to distinguish between the *start of active video* (SAV) and *end of active video* (EAV) signals. These signals indicate the beginning and end of active video data in each line. The SAV occurs on a 1-to-0 transition of H, and EAV occurs on a 0-to-1 transition of H. The space between EAV and SAV is filled with horizontal blanking data. Therefore, H = 1 during the horizontal blanking portion of the data stream, and H = 0 during the active video portion of the data stream.

The letter V is used to denote the vertical blanking portion of the data stream. A transition in V can occur only in the EAV sequence. When V = 1, the data stream contains vertical blanking data, and when V = 0, the data stream contains active video data.

The letter F is used to distinguish Field 1 from Field 2. Interlaced video has two fields in a frame of data. It requires each field to be handled uniquely, and alternate rows of each field combined to create the actual video image.

For interlaced video, F = 0 represents Field 1 (*Odd Field*) and F = 1 represents Field 2 (*Even Field*). Progressive video makes no distinction between Field 1 and Field 2, and F is always 0 for progressive video. Interlaced video requires each field to be handled uniquely because alternate rows of each field combine to create the actual video image.

An entire field of video is comprised of active video plus horizontal blanking (the space between an EAV and SAV code) and vertical blanking (the space where V = 1). A field of video commences on a transition of the F bit.


INTERLACED VIDEO
LINE #
NTSC
HORIZONTAL BLANKING
VERTICAL BLANKING
FIELD 1 ACTIVE VIDEO
VERTICAL BLANKING
FIELD 2 ACTIVE VIDEO
1
20
264
283
525
LINE 4
LINE 266
LINE 3
FIELD 1
FIELD 2
EAV
SAV
PAL
HORIZONTAL BLANKING
VERTICAL BLANKING
FIELD 1 ACTIVE VIDEO
VERTICAL BLANKING
FIELD 2 ACTIVE VIDEO
VERTICAL BLANKING
1
23
311
336
624
625
LINE 1
LINE 313
LINE 625
FIELD 1
FIELD 2
EAV
SAV
PROGRESSIVE VIDEO
LINE #
NTSC
HORIZONTAL BLANKING
VERTICAL BLANKING
ACTIVE VIDEO
1
46
525
EAV
SAV
PAL
HORIZONTAL BLANKING
VERTICAL BLANKING
ACTIVE VIDEO
VERTICAL BLANKING
1
45
621
625
EAV
SAV

LINE NUMBER	F	V	H (EAV)	H (SAV)
1-3 266-282	1	1	1	0
4-19 264-265	0	1	1	0
20-263	0	0	1	0
283-525	1	0	1	0

LINE NUMBER	F	V	H (EAV)	H (SAV)
1-22 311-312	0	1	1	0
23-310	0	0	1	0
313-335, 624-625	1	1	1	0
336-623	1	0	1	0

LINE NUMBER	F	V	H (EAV)	H (SAV)
1-45	0	1	1	0
46-525	0	0	1	0

LINE NUMBER	F	V	H (EAV)	H (SAV)
1-44, 621-625	0	1	1	0
45-620	0	0	1	0

Figure 18-8: Typical Video Frame Partitioning for NTSC/PAL Systems in Interlaced and Progressive ITU-R BT.656 Systems

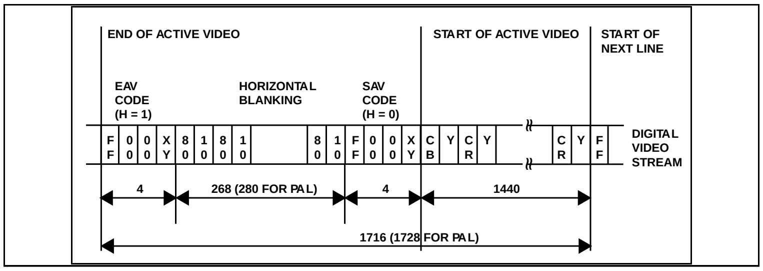

END OF ACTIVE VIDEO
START OF ACTIVE VIDEO
START OF NEXT LINE
EAV CODE (H = 1)
HORIZONTAL BLANKING
SAV CODE (H = 0)
F F
0 0
0 0
X Y
8 0
1 0
8 0
1 0
8 0
1 0
F F
0 0
0 0
X Y
C B
Y
C R
Y
C R
Y
F F
DIGITAL VIDEO STREAM
4
268 (280 FOR PAL)
4
1440
1716 (1728 FOR PAL)

Figure 18-9: ITU-R 656 8-Bit Parallel Data Stream from NTSC (PAL) Systems

NOTE: Refer to the ***Control Sequences for 8-Bit and 10-Bit ITU-R 656 Video*** table. There is a defined preamble of three data elements (for example, in the case of 8-bit video: 0xFF, 0x00, 0x00), followed by the XY status word. The status word contains four protection bits for error detection and correction excluding the F (field), V (vertical blanking), and H (horizontal blanking) bits. F and V are only allowed to change as part of EAV sequences (that is, transition from H = 0 to H = 1).

The bit definitions are as follows:

- F = 0 for field 1
- F = 1 for field 2

- V = 1 during vertical blanking
- V = 0 when not in vertical blanking
- H = 0 at SAV
- H = 1 at EAV
- P3 = V XOR H
- P2 = F XOR H
- P1 = F XOR V
- P0 = F XOR V XOR H

P3–P0 are protection bits that enable 1-bit and 2-bit error detection, and 1-bit error correction at the receiver. The EPPI corrects the error if it detects 1-bit errors in F, V, or H. Errors in the protection bits themselves are detected but not corrected.

Table 18-9: Control Sequences for 8-Bit and 10-Bit ITU-R 656 Video

	8-Bit Data								**10-Bit Data**	
	D9 (MSB	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Preamble	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
Control Byte	1	F	V	H	P3	P2	P1	P0	0	0

The `EPPI_STAT` register contains 2 bits, `EPPI_STAT.ERRDET` and `EPPI_STAT.ERRNCOR`, that are used to report the status of error detected and error not corrected, respectively.

The `EPPI_STAT.ERRDET` bit is set whenever an error is detected in the status word. However, this bit does not generate an interrupt. The `EPPI_STAT.ERRNCOR` bit is set when more than a 1-bit error is detected in the status word. An interrupt is generated when the `EPPI_STAT.ERRNCOR` bit is set. It can be cleared by clearing the `EPPI_STAT.ERRNCOR` and `EPPI_STAT.ERRDET` bits. Both bits are sticky and W1C.

In many applications, video streams other than the standard NTSC/PAL formats (for example, CIF, QCIF) can be employed. The processor interface is flexible enough to accommodate different row and field lengths. In general, as long as the incoming video has the proper EAV/SAV codes, the EPPI can read it in. A CIF image could be formatted to be 656-compliant, where EAV and SAV values define the range of the image for each line. The V and F codes are used to delimit fields and frames.

The following sections provide descriptions of EPPI operations.

Table 18-10: Operating Modes and Generic EPPI Operation

		How to configure	Useful for	How to configure in ITU R 656 Tx Mode
ITU-R BT.656 Rx	Entire field	DIR= 0 XFRTYPE = b#01		
	Active video	DIR = 0 XFRTYPE = b#00		
	Blanking only	DIR = 0 XFRTYPE = b#10		
GP 0 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#00	Applications where periodic frame syncs are not used to frame the data	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#00		
GP 1 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG =b#01	Interfacing with ADCs, DACs, and other general-purpose devices	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#01		
GP 2 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#10	Video applications that use two hardware synchronization signals, HSYNC and VSYNC	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FS_CFG = b#10		
GP 3 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#11	Video applications that use three hardware sync signals, HSYNC, VSYNC, and FIELD	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#11		

ITU-R 656 Input Modes

In the ITU-R 656 input modes, the video source provides the clock or the system supplies it externally.

As shown in the ***ITU-R 656 Input Submodes*** figure and described in the following sections, there are three submodes supported for ITU-R 656 inputs: entire field, active video only, and vertical blanking interval only.

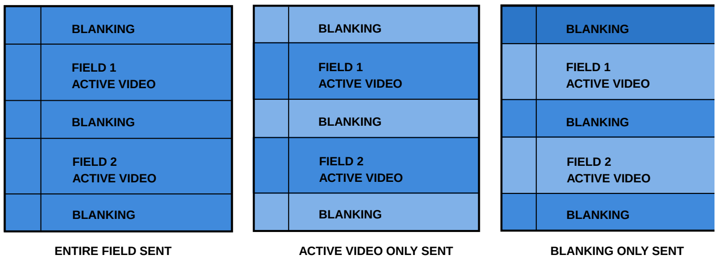


Figure 18-10: ITU-R 656 Input Submodes

Entire Field

In this mode, the EPPI reads the entire incoming bit stream. This stream includes active video as well as control byte sequences and ancillary data that can be embedded in horizontal and vertical blanking intervals.

Data transfer starts immediately after Field 1 synchronization occurs. The transfer does not include the first EAV code that contains the F = 0 assignment for interlaced video or the V = 0 assignment for progressive video.

Active Video

The EPPI uses this mode when only the active video portion of a field is of interest. The EPPI ignores (does not read in) all data between EAV and SAV, as well as all data present when V = 1. Furthermore, the control byte sequences are not stored to memory. The EPPI filters the sequences. After the start of Field 1 synchronizes, the EPPI ignores incoming samples until it sees an SAV.

In active video mode, programs must specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register. Programs must specify the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.

In this mode, any input data sequence that is considered part of the preamble is not sent to memory such as in 8-bit ITU mode. If 0xFF or 0x00 appear in the input data stream, these values are considered part of the preamble. The part of the preamble can appear individually and not be tagged along with the preamble sequence FF, 00, 00. This functionality also applies to vertical blanking interval mode.

Vertical Blanking Interval (VBI)

In this mode, data transfer is only active while V = 1 is in the control byte sequence. This functionality indicates that the video source is in the midst of the vertical blanking interval (VBI), which is sometimes used for ancillary data transmission. The ITU-R 656 recommendation specifies the format for these ancillary data packets, but the EPPI is not equipped to decode the packets themselves. Software must handle this task. Horizontal blanking data is logged where it coincides with the rows of the VBI.

The VBI is split into two regions within each field. The EPPI considers these two separate regions as one contiguous space. However, frame synchronization begins at the start of Field 1, which does not necessarily correspond to the start of vertical blanking. For instance, in 525/60 systems, the start of Field 1 (F = 0) corresponds to line 4 of the VBI.

In VBI mode, the program must specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register. The program must specify the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.

In this mode, any input data sequence that is considered as part of the preamble is not sent to memory such as in 8-bit ITU mode. If 0xFF or 0x00 appears in the input data stream, these values are considered part of the preamble. The part of the preamble can appear individually, and not be tagged along with the preamble sequence FF, 00, 00. This functionality applies to active video mode too.

ITU-R 656 Output in General-Purpose Transmit Modes

In GP transmit mode, the EPPI frames an ITU-R 656 output stream with the proper preambles and blanking intervals by setting the `EPPI_CTL.BLANKGEN` bit. The EPPI fetches active data from memory through the DMA channel, saving DMA bandwidth. The EPPI generates and embeds the proper preamble, status word (EAV and SAV sequences), and blanking data along with the active video from memory. Program the `EPPI_FS1_PASPL`, `EPPI_FS2_WLVB`, `EPPI_FS2_PALPF`, and `EPPI_FS1_WLHB` registers to perform the desired functions. The EPPI can also drive out the frame syncs using the `EPPI_CTL.FSCFG` bit setting.

The ***16-Bit Transmit with Internal Blanking Generation*** figure shows the bit stream format in 16-bit transmit modes with blanking generation (`EPPI_CTL.BLANKGEN` enabled). Each 16-bit data sample consists of 8-bit luma (Y) and 8-bit chroma (Cr or Cb) components. During transmission, the chroma data and blanking bytes of value 0x80 are placed on the upper half (MSBs) of the data lines. The luma data and blacking bytes of value 0x10 are placed on the lower half (LSBs) of the data lines.

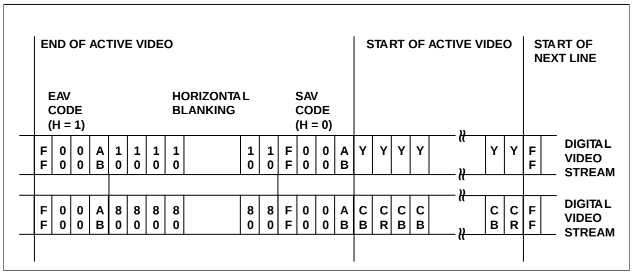


Figure 18-11: 16-Bit Transmit with Internal Blanking Generation

The ***Generated Blanking Preamble Sequence*** figure shows the data transmitted by the EPPI in this mode. After the EPPI is enabled, and if the EPPI FIFO is not empty, the transmission starts by sending out an EAV sequence for a vertical blanking line. For interlaced video, F starts at 1. For progressive video, F is always 0.

NOTE: Internal blanking generation functionality is valid only when the data length is 8, 10, or 16 bits and when the EPPI is in GP transmit modes. The `EPPI_CTL.BLANKGEN` bit generates preambles even in GP 2FS mode.

The internal blanking generation functionality of the ITU-R 656 output mode can also be bypassed by clearing the `EPPI_CTL.BLANKGEN` bit. (For example, if sending ancillary data in the blanking interval). The `EPPI_CTL.BLANKGEN` bit generates preambles even in GP 2FS mode.

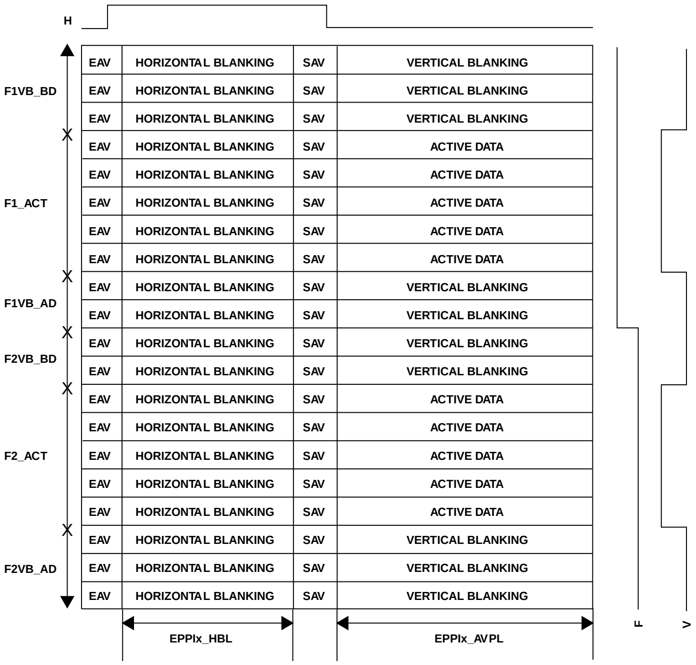


Figure 18-12: Generated Blanking Preamble Sequence

Frame Synchronization in ITU-R 656 Modes

For interlaced video, the start of frame synchronization occurs when a high-to-low transition is detected in F, the field indicator. For progressive video, the start of frame synchronization occurs when a high-to-low transition is detected in V, the vertical blanking indicator. These transitions in F and V can occur only in the EAV sequence. A start of line is detected on a low-to-high transition in H, the horizontal blanking indicator, which occurs in the EAV sequence as well.

For interlaced video, the start of frame corresponds to the start of field 1. Therefore, up to two fields can be ignored before the EPPI receives data. (For example, if field 1 started before the EPPI-to-camera channel was established). For progressive video, the start of frame corresponds to the start of active video.

Because all H and V signaling is embedded in the data stream in ITU-R 656 modes, the EPPI ignores the count registers (`EPPI_HCNT`, `EPPI_VCNT`). However, the EPPI still uses the `EPPI_FRAME` register to check for synchronization errors. Therefore, program this MMR with the number of lines expected in each frame of video.

The EPPI monitors the number of EAV-to-SAV transitions that occur from the start of a frame until it decodes the end of frame condition. (For example, a transition from F = 1 to F = 0 for interlaced video and a transition from V = 1 to V = 0 for progressive video).

At the end of frame condition, the actual number of lines processed is compared against the value in `EPPI_FRAME`. If there is a mismatch, a frame track error is asserted in the `EPPI_STAT` register. For example, if an SAV transition was missed, the current field only has NUM_ROWS – 1 rows. But, resynchronization occurs at the start of the next frame. When the EPPI receives the entire field, the field status bit is toggled in the `EPPI_STAT` register. This way, an interrupt service routine (ISR) can discern which field was previously read in.

General-Purpose EPPI Modes

The general-purpose (GP) EPPI modes accommodate a wide variety of data capture and transmission applications.

Each EPPI has three bidirectional frame sync pins. The EPPI internally generates frame syncs, or an external device communicating with the EPPI generates them.

GP modes differ based on the number of frame syncs used and the EPPI supports GP 0 FS—GP 3 FS modes.

All the GP modes, except 0 FS mode, support horizontal windowing. GP modes with 2 and 3 frame syncs also support vertical windowing.

For GP transmit modes with internal clock or frame syncs, the EPPI starts generating the clock or frame syncs only when the EPPI FIFO is full for the first time. For GP 0 FS transmit mode, the EPPI only starts transmitting when the EPPI FIFO is full for the first time.

General-Purpose 0 Frame Sync Mode

This mode is useful for applications where periodic frame syncs are not used to frame the data.

After the initial trigger, the EPPI receives or transmits data samples on every clock cycle. However, if the `EPPI_CTL.SKIPEN` bit is set for receive mode, the EPPI receives only alternate data samples.

The `EPPI_LINE`, `EPPI_FRAME`, `EPPI_HCNT`, `EPPI_HDLY`, `EPPI_VCNT`, and `EPPI_VDLY` registers are not valid for GP 0 FS mode. Therefore, windowing is not possible in this mode. Also, line and frame track errors are not applicable in this mode.

GP 0 FS receive mode is further divided into two submodes; internal trigger (`EPPI_CTL.FLDSEL` bit =0) and external trigger (`EPPI_CTL.FLDSEL` bit =1). The submodes are based on how the processor initiates data transmission or reception. GP 0 FS transmit mode is always internally triggered. DMA handles all subsequent data manipulation.

- ***Frame synchronization in GP 0 FS external trigger mode.*** When the EPPI is programmed in external trigger mode, it does not generate the `EPPI_FS1` signal and the external device must provide a trigger. The EPPI starts receiving the data as soon as an `EPPI_FS1` signal assertion is detected. After that, the DMA handles all subsequent data manipulation and any activity on `EPPI_FS1` is ignored.
- ***Frame synchronization in GP 0 FS internal trigger mode.*** When the EPPI is programmed in internal trigger mode, it starts receiving or transmitting data as soon as the EPPI clock is enabled and synchronized. There can be up to four PPI clock cycles of latency before valid data is received or transmitted.

General-Purpose 1 Frame Sync Mode

This mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and other general-purpose devices. This mode works for both transmit and receive.

The `EPPI_FRAME`, `EPPI_VDLY`, and `EPPI_VCNT` registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not available.

General-Purpose 2 Frame Sync Mode

This mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC signal can be connected to `EPPI_FS1` and the VSYNC signal can be connected to `EPPI_FS2`.

Data Enable in General-Purpose 2 Frame Sync Transmit Mode

The `EPPI_FS3` pin functions as a data enable (DEN) pin, when EPPI is configured in GP 2 FS transmit mode and generating the frame sync internally. The bits `EPPI_CTL.MUXSEL` and `EPPI_CTL.CLKGATEN` are not enabled. The functionality of the DEN pin is described in the following two cases.

Case 1

Blanking generation is configured using the `EPPI_CTL.BLANKGEN` bit. EPPI data length (`EPPI_CTL.DLEN` bit) is configured for 8, 10, or 16-bit transfers. The `EPPI_FS3` pin asserts during the *active data* regions, aligned with `EPPI_CLK` according to the clock polarity (`EPPI_CTL.POLC` bit) settings. For this mode, the pin `EPPI_FS3` is driven based on the `EPPI_CTL.POLC` setting. The pin `EPPI_FS3` is driven out on the same EPPI clock edge that drives out data. The frame sync polarity (`EPPI_CTL.POLS`) setting does not apply here—`EPPI_FS3` is always active high in this mode.

Case 2

Blanking generation (`EPPI_CTL.BLANKGEN` =0) is disabled. Or blanking generation is enabled, but the EPPI data length (`EPPI_CTL.DLEN` bit) is configured for a transfer size other than 8, 10, or 16 bits. The `EPPI_FS3` pin asserts at the start of the active data region on each line, aligned with `EPPI_CLK` according to the `EPPI_CTL.POLC` bit settings. For this mode, the pin `EPPI_FS3` is driven based on the `EPPI_CTL.POLC` setting. The `EPPI_FS3` signal is driven out on the same EPPI clock edge that drives out data.

The `EPPI_CTL.POLS` bit setting does not apply for case 2. The `EPPI_FS3` signal is always active high in this mode. Once asserted, `EPPI_FS3` stays asserted for the number of clock cycles per line configured in the `EPPI_HCNT` register, then it deasserts. This behavior on each line continues for the total number of lines programmed in the `EPPI_VCNT` register per frame. The behavior repeats at the start of subsequent video frames.

In case 2, if transmission of valid data is held off due to delays programmed in the `EPPI_HDLY` or `EPPI_VDLY` registers, the assertion of `EPPI_FS3` is also held off. The delay is on a per-line or per-frame basis.

General-Purpose 3 Frame Sync Mode

This mode is useful for video applications that use three synchronization signals for hardware: HSYNC, VSYNC, and FIELD. The HSYNC connects to the `EPPI_FS1` pin, VSYNC connects to the `EPPI_FS2` pin, and FIELD connects to the `EPPI_FS3` pin.

GP 3 FS mode is similar in operation to GP 2 FS mode. However, the start of frame synchronization in GP 3 FS also considers the state of the `EPPI_FS3` pin. All the windowing register settings (`EPPI_FRAME`, `EPPI_LINE`, `EPPI_HDLY`, `EPPI_HCNT`, `EPPI_VDLY`, and `EPPI_VCNT` registers), as well as data reception or transmission and error generation are the same as for GP 2 FS mode. In addition, for GP 3 FS mode with internal frame syncs, the `EPPI_CTL.FLDSEL` bit setting specifies the condition under which the transfer begins.

The EPPI generates the `EPPI_FS3` signal and toggles during every assertion of `EPPI_FS2` or a combination of `EPPI_FS2` and `EPPI_FS1`. The toggle depends on the `EPPI_CTL.FLDSEL` bit setting. The EPPI skips an `EPPI_FS2` signal when the `EPPI_FS3` value is high. Because of this condition, program the `EPPI_FS2` period value to half of the total number of pixels in the frame as in GP 3 FS mode. When in GP 2 FS mode, program the `EPPI_FS2` period with the value equal to the number of pixels per frame.

Supported Data Formats

The following sections describe EPPI receive and transmit data formats.

Receive Data Formats

The ***EPPI Receive Data Formats*** table provides information about EPPI configuration for specific use models for receive data.

Table 18-11: EPPI Receive Data Formats

Input Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	`EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =1 if necessary to separate chroma components
	RGB sensor	No splitting possible. `EPPI_CTL.PACKEN` =1 – Four EPPI words are packed to 32-bit DMA data. `EPPI_CTL.PACKEN` =0 – Each EPPI word is sent as 8-bit data on the 32-bit DMA bus. This transfer consumes 4 times the DMA bandwidth of the 8-bit case with `EPPI_CTL.PACKEN` =1;
	ADCs	Gives I (in phase) and Q (quadrature) components. `EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =0 since there are only two components.

Table 18-11: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
10	NTSC/PAL data	Each EPPI word is zero filled or sign extended to 16 bits. `EPPI_CTL.SPLTEO` =1. `EPPI_CTL.SUBSPLTODD` =1 if necessary to separate chroma components.
	RGB sensor	No splitting possible. `EPPI_CTL.PACKEN` =1. Two EPPI words are zero filled or sign extended to 16 bits and packed to 32-bit DMA data. `EPPI_CTL.PACKEN` =0. Each EPPI word can be zero filled or sign extended to 16 bits and sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 10-bit case with `EPPI_CTL.PACKEN` =1;
	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. `EPPI_CTL.SPLTEO` =1 S`EPPI_CTL.SUBSPLTODD` = 0 since there are only two components.
12	RGB sensor	No splitting possible. `EPPI_CTL.PACKEN` =1. Two EPPI words are zero filled or sign extended to 16 bits and packed to 32-bit DMA data. `EPPI_CTL.PACKEN` =0. Each EPPI word can be zero filled or sign extended to 16 bits and sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 12-bit case with `EPPI_CTL.PACKEN` =1;
	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. `EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =0 since there are only two components.
14	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. `EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =0 since there are only two components.

Table 18-11: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
16	8-bit luma/chroma pair for NTSC or HD	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =1, `EPPI_CTL.SUBSPLTODD` =1 if necessary to separate chroma components.
	16-bit luma/chroma pair for NTSC or HD	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =0, `EPPI_CTL.SUBSPLTODD` =1 if necessary to separate chroma components.
	RGB565 sensor	No splitting possible. `EPPI_CTL.PACKEN` =1. Two EPPI words are packed to a 32-bit DMA data. `EPPI_CTL.RGBFMTEN` is valid only in transmit modes. So, RGB565 cannot be byte aligned in memory. `EPPI_CTL.PACKEN` =0. Each EPPI word is sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 16-bit case with `EPPI_CTL.PACKEN` =1
	8-bit ADCs I/Q pair	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =1, `EPPI_CTL.SUBSPLTODD` =0.
	16-bit ADCs I/Q pair	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =0, `EPPI_CTL.SUBSPLTODD` =0.

Transmit Data Formats

The ***EPPI Transmit Data Formats*** table provides information about EPPI configuration for specific use models for transmit data.

Table 18-12: EPPI Transmit Data Formats

Output Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	`EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =1 if the chroma components (U and V) come in separate DMA words.
	Serial RGB for lower-resolution LCDs	No splitting possible. `EPPI_CTL.PACKEN` =1. The 32-bit DMA data is unpacked to drive four EPPI words. `EPPI_CTL.PACKEN` =0. The lowest 8 bits of the DMA data is driven on the EPPI data and the rest of the 24 bits are discarded. This transfer consumes 4 times the DMA bandwidth of the 8-bit case with `EPPI_CTL.PACKEN` =1.
10	NTSC/PAL data	`EPPI_CTL.SPLTEO` =1 `EPPI_CTL.SUBSPLTODD` =1 if the chroma components (U and V) come in separate DMA words.
	DACs	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SUBSPLTODD` =0.

Table 18-12: EPPI Transmit Data Formats (Continued)

Output Data Width	Use Model	Splitting/Packing Options
12	DACs	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SUBSPLTODD` =0.
14	DACs	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SUBSPLTODD` =0.
16	8-bit luma/chroma pair for NTSC or HD	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =1, `EPPI_CTL.SUBSPLTODD` =1 if the chroma components (U and V) come in separate DMA words.
	16-bit luma/chroma pair for NTSC or HD	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =0, `EPPI_CTL.SUBSPLTODD` =1 if the chroma components (U and V) come in separate DMA words.
	RGB565 LCD	No splitting possible. `EPPI_CTL.RGBFMTEN` =1. Takes RGB888 data from the memory and drops the LSBs from each component to drive out RGB565 data.
	8-bit ADCs I/Q pair	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =1, `EPPI_CTL.SUBSPLTODD` = 0
	16-bit ADCs I/Q pair	`EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SPLTWRD` =0, `EPPI_CTL.SUBSPLTODD` =1
18	RGB666 LCD	No splitting possible. `EPPI_CTL.RGBFMTEN` =1. Takes RGB888 data from the memory and drops the 2 LSBs from each component to drive out RGB666 data.

Data Transfer Modes

The following sections describe EPPI data transfer modes, including receive or transmit data packing, sign extension, zero fill, receive or transmit split modes, clock gating, delayed start, and data consistency management.

Data Packing for Receive Modes

For receive modes, if the `EPPI_CTL.PACKEN` bit =1 and the DMA is 32 bits, the EPPI packs the incoming data into 32-bit words based on the `EPPI_CTL.DLEN` and `EPPI_CTL.SWAPEN` bit settings. When `EPPI_CTL.SWAPEN` =0, the EPPI puts the first data in the least significant bits and when `EPPI_CTL.SWAPEN` =1, the EPPI puts the first data in the most significant bits. The packing options for the `EPPI_CTL.DLEN` bits are as follows.

- When `EPPI_CTL.DLEN` =8, four 8-bit words can be packed into one 32-bit word.
- When `EPPI_CTL.DLEN` =16, two 16-bit words can be packed into one 32-bit word.
- For `EPPI_CTL.DLEN` values that are more than 8 bits but less than 16 bits, two such words are either sign-extended or zero-filled to 16 bits, and packed into one 32-bit word.
- When `EPPI_CTL.DLEN` =18, the EPPI sign-extends or zero-fills the 18-bit data to 24 bits and packs four 24-bit words into three 32-bit words.

- When `EPPI_CTL.DLEN` =24, the EPPI packs four 24-bit words into three 32-bit words.

When `EPPI_CTL.PACKEN` =0, the EPPI receives the incoming data and sends it on the bus as-is. If `EPPI_CTL.DLEN` is less than or equal to 16 bits, the DMA is a 16-bit DMA; otherwise it is a 32-bit DMA.

Data Packing for Transmit Modes

For transmit modes, if the `EPPI_CTL.DLEN` bit =1 and the DMA is a 32-bit DMA, the EPPI unpacks the 32-bit word according to the `EPPI_CTL.DLEN` and `EPPI_CTL.SWAPEN` bit settings.

If `EPPI_CTL.SWAPEN` =1, the EPPI transmits the most significant bits as the first data, and if `EPPI_CTL.SWAPEN` =0, the EPPI transmits the least significant bits as the first data. The unpacking options for the `EPPI_CTL.DLEN` bits are as follows.

- When `EPPI_CTL.DLEN` =8, the EPPI transmits one 32-bit word from memory as four 8-bit data words.
- For `EPPI_CTL.DLEN` values greater than 8 bits but less than or equal to 16 bits, the EPPI transmits one 32-bit word from memory as two 16-bit data words.
- When `EPPI_CTL.DLEN` =18 or 24, the EPPI transmits three 32-bit words from memory as four data words.

Sign-Extended and Zero-Filled Data

The following list describes the bit settings and functionality for sign-extending and zero-filling data.

- For `EPPI_CTL.DLEN` equal to 10, 12 or 14, data is zero-filled or sign-extended to 16 bits.
- For `EPPI_CTL.DLEN` equal to 18 bits, data is zero-filled or sign-extended to 24 bits if packing is enabled, and zero-filled or sign-extended to 32 bits if packing is disabled.
- For `EPPI_CTL.DLEN` equal to 24 bits, data is zero-filled or sign-extended to 32 bits if packing is disabled.
- For `EPPI_CTL.DLEN` equal to 8 bits, data is zero-filled or sign-extended to 16 bits if packing is disabled.
- If `EPPI_CTL.SIGNEXT` =1, then the data is sign-extended, otherwise it is zero-filled.

Split Receive Modes

The control register has three control bits for split receive modes: `EPPI_CTL.SPLTEO`, `EPPI_CTL.SUBSPLTODD`, and `EPPI_CTL.DMACFG`. Packing is not valid in split modes.

- If `EPPI_CTL.SPLTEO` =1, the EPPI splits the incoming data stream into two substreams, an even stream, and an odd stream, and packs them separately.
- The `EPPI_CTL.SUBSPLTODD` bit is available only when `EPPI_CTL.SPLTEO` =1. When `EPPI_CTL.SUBSPLTODD` =1, the EPPI subsplits the odd substream, and packs the streams separately.
- The `EPPI_CTL.DMACFG` bit is also available only if `EPPI_CTL.SPLTEO` =1. If `EPPI_CTL.DMACFG` =1, the EPPI uses two DMA channels and if `EPPI_CTL.DMACFG` =0, the EPPI uses only one DMA channel.

Split Transmit Modes

The `EPPI_CTL` register has three control bits for split transmit modes: `EPPI_CTL.SPLTEO`, `EPPI_CTL.SUBSPLTODD`, and `EPPI_CTL.DMACFG`. The DMA is always a 32-bit DMA. Packing is not valid in split modes.

- If `EPPI_CTL.SPLTEO` =1, the EPPI receives the Luma (Y3Y2Y1Y0) and interleaved Chroma (Cr1Cb1Cr0Cb0) data as 32 bits from the DMA channel. The EPPI interleaves the data to form a 4:2:2 YCrCb data stream to transmit.
- The `EPPI_CTL.SUBSPLTODD` bit is available only when `EPPI_CTL.SPLTEO` =1. In this case, if `EPPI_CTL.SUBSPLTODD` =1, the EPPI receives the Luma (Y3Y2Y1Y0) and deinterleaved Chroma (Cb3Cb2Cb1Cb0 and Cr3Cr2Cr1Cr0). The EPPI interleaves the data to form a 4:2:2 YCrCb data stream to transmit. (The EPPI does not decimate the chroma data when formatting it into 4:2:2.)
- The `EPPI_CTL.DMACFG` bit is also valid only if `EPPI_CTL.SPLTEO` =1. If `EPPI_CTL.DMACFG` =1, the EPPI uses two DMA channels and if `EPPI_CTL.DMACFG` =0, the EPPI uses only one DMA channel.

Clock Gating

In ITU-R BT.656 and GP 0/1/2 FS modes, `EPPI_FS3` becomes a clock-gating input. This functionality is valid for both internally and externally sourced `EPPI_CLK`, in both receive and transmit modes. This clock gating signal must be synchronous with `EPPI_CLK`. The external device on the rising edge of `EPPI_CLK` must drive the clock gating signal. Its function is to hold the sync and data lines in their current state until `EPPI_FS3` is driven low. There are no additional latency cycles upon coming out of clock gating mode.

If clock gating is not required, the `EPPI_FS3` pin must either be tied to ground, or configured to operate as another of its multiplexed functions.

In GP 2 FS transmit mode with internally generated frame syncs, the `EPPI_FS3` pin functions as a data enable signal.

Support for Delayed Start of EPPI Frame Syncs

The EPPI supports a delayed start of the `EPPI_FS1` and `EPPI_FS2` frame syncs. The `EPPI_FS1_DLY` and `EPPI_FS2_DLY` registers are programmable registers corresponding to `EPPI_FS1` (HSYNC) and `EPPI_FS2` (VSYNC).

The delay programmed in these registers applies to the first active edge of the internally generated frame sync. The delay starts from the first `EPPI_CLK` edge. The delay counter runs only for the first time and then shuts off until the EPPI is reenabled. (The delay counter is the period counter itself, since they do not run together.) Program the delay registers prior to the first `EPPI_CLK` edge (similar to the width and period registers). The ***EPPI Delayed Frame Sync Generation*** figure shows the functioning of `EPPI_FS1` and `EPPI_FS2`.

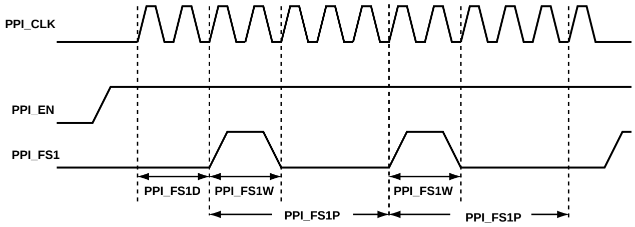


Figure 18-13: EPPI Delayed Frame Sync Generation

Ignoring Premature External Frame Syncs for Data Consistency

Once a frame has started with a VSYNC followed by an HSYNC (or both coming together), a line is tracked. When the count expires, the state machine waits at the end of line for an HSYNC to come. With the arrival of the HSYNC, the state machine starts tracking the next line, and so on.

The number of lines tracked is counted separately. Once the end of a frame is reached, the state machine waits there for the next VSYNC/HSYNC combination. The next frame starts once they are sampled. Unfortunately, every incoming FS (VSYNC or HSYNC) resets the respective counters and the tracking starts all over (even if the FS signals are premature). The result is incomplete data (or frames) to enter into memory through the PxP interface.

To correct this problem, the EPPI waits for a frame or line completion before considering any incoming FS as valid.

- Single FS mode and line tracking in dual FS mode – When a line is in progress, when HSYNC is detected prematurely, it is ignored. A line track underflow event is generated.
- Dual FS mode – If a VSYNC is received when a frame is in progress, it is ignored. A frame track underflow error (`EPPI_STAT.FTERRUNDR`) is generated.

Ignoring the FS ensures that once a frame starts, the amount of data that goes into the memory/PxP interface corresponds exactly to the programmed data size in a frame.

NOTE: Even if the premature FS is a valid FS, the state machine loses at most one frame and it recovers in the subsequent FS. The FS to number of data going into the memory relationship is always maintained as programmed.

When data underflow errors occur at the DMA interface, the EPPI does the following.

- If a premature line sync is detected, an LT underflow error is generated (`EPPI_STAT.LTERRUNDR` =1). All further line track errors are ignored until the EPPI detects the next valid line sync.
- If a premature frame sync is detected, an FT underflow error is generated (`EPPI_STAT.FTERRUNDR` =1). All further frame track and line track errors are ignored until the EPPI detects the next valid frame sync.

EPPI Event Control

The following sections describe how EPPI manages events.

EPPI Status, Error, and Interrupt Signals

The EPPI generates error interrupts (flagged in the `EPPI_STAT` register) when any one of the following error conditions occur.

- `EPPI_STAT.YFIFOERR` (YFIFO underflow or overflow)
- `EPPI_STAT.CFIFOERR` (CFIFO underflow or overflow)
- `EPPI_STAT.LTERROVR` (line track overflow error)
- `EPPI_STAT.LTERRUNDR` (line track underflow error)
- `EPPI_STAT.FTERROVR` (frame track overflow error)
- `EPPI_STAT.FTERRUNDR` (frame track underflow error)
- `EPPI_STAT.ERRNCOR` (ITU preamble error not corrected)

A W1C (write-1-to-clear) operation clears the error conditions. Each of the individual conditions which cause an EPPI error interrupt can be masked. The interrupt mask register (`EPPI_IMSK`) allows the masking of individual conditions which cause error interrupts.

There is only one interrupt line from each EPPI so all interrupts are internally OR'ed and sent as a single interrupt to the core. The `EPPI_STAT` register must then be read to discover specific errors. The following sections describe these errors in detail.

Frame and Line Track Errors

In external frame sync mode, the EPPI uses line track error (`EPPI_STAT.LTERROVR` and `EPPI_STAT.LTERRUNDR`) and frame track error (`EPPI_STAT.FTERROVR` and `EPPI_STAT.FTERRUNDR`) status bits to monitor the line and frame synchronization errors. The EPPI updates the bits when there is a mismatch detected in the HSYNC and VSYNC as compared to the programmed values in `EPPI_LINE` and `EPPI_FRAME` count registers.

Line Track Errors

The line track overflow (`EPPI_STAT.LTERROVR`) and underflow errors (`EPPI_STAT.LTERRUNDR`) generate a maskable interrupt as soon as the EPPI identifies them and not at the next frame sync.

- If the frame sync has not arrived when the `EPPI_LINE` counter expires, then the `EPPI_STAT.LTERROVR` error is generated.
- When the `EPPI_LINE` counter is running and a frame sync is detected, the `EPPI_STAT.LTERRUNDR` error is generated. A W1C operation clears boths interrupts.

Frame Track Errors

The frame track overflow (`EPPI_STAT.FTERROVR`) and underflow errors (`EPPI_STAT.FTERRUNDR`) generate a maskable interrupt as soon as the EPPI identifies them. When the `EPPI_FRAME.VALUE` counter expires, the `EPPI_STAT.FTERROVR` error is reported before the next frame sync arrives.

When the `EPPI_FRAME` counter is running and a frame sync is detected, then an `EPPI_STAT.FTERRUNDR` is reported.

Both errors generate an error interrupt. Perform a W1C operation to clear the interrupts at their respective locations in the status register.

A premature frame sync results in a frame track under run error. But, the error is logged (register bit set) only after the subsequent blanking period (if any) elapses.

Preamble Error Not Corrected Error

The EPPI supports data embedded frame syncs in ITU and SMPTE formats. In these formats, the module can receive an erroneous preamble which is not correctable. The `EPPI_STAT.ERRNCOR` error signals when this event occurs.

EPPI Programming Model

The following sections describe programming techniques, including receiving or transmitting ITU-R 656 frames; configuring transfers in GP0, GP1, GP2, and GP3 modes; and managing EPPI mode configurations.

Receiving ITU-R 656 Frames

The EPPI supports the reception of ITU-R 656 compliant frames.

1. Configure the EPPI to receive either full ITU-R 656 frame, active video, or blanking information by configuring the `EPPI_CTL.XFRTYPE` bits.
2. In both active video mode and in VBI (vertical blanking information) mode, specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register. Specify the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.
3. Configure DMA descriptors to move the data to memory.
4. Enable DMA.
5. Enable the EPPI.
6. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

Depending on the EPPI configuration, either the full ITU-R 656 frame is moved to memory or only the active video or only the blanking information.

Transmitting ITU-R 656 Frames in GP Transmit Modes

The EPPI can take active video from memory and generate the proper preambles and blanking information to produce valid ITU-R 656 video frames for transmission.

1. Provide active data frame in memory.
2. Set the `EPPI_CTL.BLANKGEN` bit so the EPPI generates blanking information.
3. Configure the `EPPI_FS1_WLHB`, `EPPI_FS1_PASPL`, `EPPI_FS2_WLVB`, `EPPI_FS2_PALPF` registers accordingly.
4. Configure the rest of the EPPI settings.
5. Configure DMA to fetch active frame data from memory buffers.
6. Enable DMA.
7. Enable the EPPI.
8. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

The EPPI takes the active data from memory, generates the blanking information, and transmits an ITU-R 656 frame

Configuring Transfers in GP 0 FS Mode

The EPPI can be configured to not use periodic frame syncs to frame the data.

1. Configure the EPPI to operate in GP 0 FS mode by setting `EPPI_CTL.XFRTYPE` = b#11 and `EPPI_CTL.FSCFG` = b#00.
2. When receiving, configure the EPPI to trigger on internally or externally by setting the `EPPI_CTL.FLDSEL` field appropriately. When transmitting, the EPPI always generates a trigger internally.
3. Configure DMA to move the data to or from memory.
4. Enable DMA.
5. Enable EPPI.
6. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

The DMA descriptions control the amount of data transferred. The frame syncs from the EPPI do not control the amount.

Configuring Transfers in GP 1 FS Mode

The GP 1 FS mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and other general-purpose devices. This mode works for both transmit and receive.

NOTE: The `EPPI_FRAME`, `EPPI_VDLY`, and `EPPI_VCNT` registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not possible in this mode.

1. Configure GP 1 FS mode by setting the `EPPI_CTL.XFRTYPE` bit =b#11 and the `EPPI_CTL.FSCFG` bit =b#01. An external device can provide the frame syncs or the EPPI can source the frame syncs.
2. Program the `EPPI_LINE` register to contain the number clock cycles expected between two assertions of the `EPPI_FS1` signal to monitor the line track errors. Program the `EPPI_LINE` register before the `EPPI_HCNT` register.
3. Program the `EPPI_HDLY` register to contain the number of clock cycles to wait after the assertion of `EPPI_FS1`. For example, the start of frame.
4. Program the `EPPI_HCNT` register to contain the number of data samples to receive or transmit for each frame.
5. Configure DMA to move the data to or from memory.
6. Enable DMA.
7. Enable the EPPI.
8. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then, switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

Data moves in or out of memory. A frame sync frames the data for every line.

Configuring Transfers in GP 2 FS Mode

GP 2 FS mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC connects to the `EPPI_FS1` signal and VSYNC connects to the `EPPI_FS2` signal.

1. Configure the EPPI to operate in GP 0 FS mode by setting the `EPPI_CTL.XFRTYPE` bit =b#11 and the `EPPI_CTL.FSCFG` bit =b#10. An external device can provide the frame syncs or the EPPI can source the frame syncs.
2. Program the `EPPI_FRAME` register to contain the number of expected lines per frame. The value can be equal to the number of `EPPI_FS1` signal assertions expected between the start of each frame sync. The EPPI uses the value to monitor frame track errors. Program the `EPPI_FRAME` register before the `EPPI_VCNT` register.
3. Program the `EPPI_LINE` register to contain the number of clock cycles expected between two assertions of the `EPPI_FS1` signal to monitor line track errors. Program the `EPPI_LINE` register before the `EPPI_HCNT` register.
4. Program the `EPPI_HDLY` register to configure the number of clock cycles to wait after the assertion of the `EPPI_FS1` signal. (For example, the start of the line).
5. Program the `EPPI_HCNT` register to contain the number of data samples to receive or transmit for each line.

6. Program the `EPPI_VDLY` register to contain the number of lines to wait after the start of frame is detected.
7. Program the `EPPI_VCNT` register to contain the number of lines to receive or transmit.
8. If setting up the EPPI for transmit, the data enable (DEN) pin behaves according to the enabling of the blanking generation and the data length setting (DLEN). See Data Enable in General-Purpose 2 Frame Sync Transmit Mode for more details.
9. Enable DMA.
10. Enable the EPPI.
11. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

Data moves in or out of memory. A frame sync frames the data for every line and frame.

Configuring Transfers in GP 3 FS Mode

GP 3 FS mode is useful for video applications that use three synchronization signals for hardware: HSYNC, VSYNC, and FIELD. The HSYNC connects to `EPPI_FS1`, VSYNC connects to `EPPI_FS2`, and FIELD connects to `EPPI_FS3`.

1. Configure the EPPI to operate in GP 3 FS mode by setting the `EPPI_CTL.XFRTYPE` bit =b#11 and the `EPPI_CTL.FSCFG` bit =b#11. An external device can provide the frame syncs or the EPPI can source the frame syncs.
2. Configure the windowing registers according to steps in GP 2 FS mode.
3. Enable DMA.
4. Enable the EPPI.
5. To program the EPPI in internal clock mode, follow the procedure above with the `EPPI_CTL.ICLKGEN` bit =0. After enabling the EPPI, add a delay of 200 SCLK1_0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the `EPPI_CTL.ICLKGEN` bit =1.

Data moves in or out of memory. A frame sync frames the data for every line and frame. Operation and result are similar to operation in GP 2 FS mode but the EPPI also uses the `EPPI_FS3` signal.

Configuring the EPPI to Use the Windowing Feature

Windowing is a useful feature for applications where the region of interest is smaller than the active video stream (for example, sensor calibration, auto-focusing, and others). It can result in significant DMA bandwidth reduction. The EPPI supports windowing for GP input modes.

1. Program the `EPPI_FRAME` register with the number of lines the frame contains.

2. Program the `EPPI_LINE` register with the number of samples per line in the frame.
3. Program the `EPPI_VDLY` register with the number of lines to wait after the start of a new frame before starting to read or transmit data.
4. Program the `EPPI_VCNT` register with the number of lines to read in or write out after `EPPI_VDLY` number of lines from the start of the frame.
5. Program the `EPPI_HDLY` register with the number of clock cycles to delay after the assertion of `EPPI_FS1` is detected for the start of a new line.
6. Program the `EPPI_HCNT` register with the number of samples to read in or write out after `EPPI_HDLY` number of cycles have expired since the assertion of `EPPI_FS1`.

EPPI Mode Configuration

This section describes EPPI mode configurations, including support for all EPPI transmit and receive modes.

Configuring 8-Bit Receive Mode

For 8-bit non-split receive mode and if `EPPI_CTL.PACKEN` =1, the EPPI packs 4 bytes of incoming data into a 32-bit word. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the `EPPI_CTL.SWAPEN` bit setting.

Table 18-13: 8-Bit Receive Mode with Packing Enabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO =X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x11						
0x22						
0x33						
0x44	0x4433 2211	0x1122 3344				
0x55						
0x66						
0x77			0x7755 3311		0x1133 5577	
0x88	0x8877 6655	0x5566 7788		0x8866 4422		0x2244 6688
0x99						
0xAA						
0xBB						

Table 18-13: 8-Bit Receive Mode with Packing Enabled (Continued)

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO =X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0xCC	0xCCBB AA99	0x99AA BBCC				
0xDD						
0xEE						
0xFF			0xFFDD BB99		0x99BB DDFF	
0x00	0x00FF EEDD	0xDDE EFF00		0x00EE CCAA		0xAACC EE00

If `EPPI_CTL.PACKEN`=0, the DMA is a 16-bit DMA and the EPPI either sign-extends or zero-fills the bytes of incoming data into a 16-bit word. The `EPPI_CTL.SWAPEN` bit has no effect if `EPPI_CTL.PACKEN`=0.

Table 18-14: 8-Bit Receive Mode with Packing Disabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=1
0x44	0x0044	0x0044	0x0044	
0x55	0x0055	0x0055		0x0055
0x66	0x0066	0x0066	0x0066	
0x77	0x0077	0x0077		0x0077
0x88	0x0088	0xFF88	0x0088	
0x99	0x0099	0xFF99		0xFF99
0xAA	0x00AA	0xFFAA	0x00AA	
0xBB	0x00BB	0xFFBB		0xFFBB

Configuring 10/12/14-Bit Receive Modes

For 10, 12, or 14-bit non-split receive modes, the EPPI first either zero-fills or sign-extends the incoming data into a 16-bit word. The action depends on the setting of the `EPPI_CTL.SIGNEXT` bit. If `EPPI_CTL.PACKEN` =1, the EPPI then packs two of these words into one 32-bit word. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the `EPPI_CTL.SWAPEN` bit setting.

Table 18-15: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=1
0x111	0			
0x222	1	0xFE22 0111	0x0111 FE22	
0x333	1			0xFF33 0111
0x044	0	0x0044 FF33	0xff33 0044	
0x155	0			
0x266	1	0xFE66 0155	0x0155 FE66	
0x377	1			0xFF77 0155
0x088	0	0x0088 FF77	0xFF77 0088	

Table 18-16: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=1
0x111	0			
0x222	1			
0x333	1		0x0011 FF33	
0x044	0	0x0044 FE22		0xFE22 0044
0x155	0			
0x266	1			
0x377	1		0x0155 FF77	
0x088	0	0x0088 FE66		0xFE66 0088

Table 18-17: 10-Bit Receive Mode, with Zero-Fill, with Packing Enabled

Pin Data (10 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=0	DMA DATA SKIP_EN=1 SKIP_EO=0 SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=0
0x111						
0x222	0x0222 0111	0x0111 0222				
0x333			0x0333 0111		0x0011 0333	
0x044	0x0044 0333	0x0333 0044		0x0044 0222		0x0222 0044
0x155						
0x266	0x0266 0155	0x0155 0266				
0x377			0x0377 0155		0x0155 0377	
0x088	0x0088 0377	0x0377 0088		0x0088 0266		0x0266 0088

The ***10-bit Receive Mode with Packing Disabled*** table shows a 10-bit receive mode example when `EPPI_CTL.PACKEN` =0:

Table 18-18: 10-bit Receive Mode with Packing Disabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x111	0	0x0111	0x0111	0x0111	
0x222	1	0xFE22	0x0222		0x0222
0x333	1	0xFF33	0x0333	0xFF33	
0x044	0	0x0044	0x0444		0x0444
0x155	0	0x0155	0x0155	0x0155	
0x266	1	0xFE66	0x0266		0x0266
0x377	1	0xFF77	0x0377	0xFF77	
0x088	0	0x0088	0x0088		0x088

Configuring 16-Bit Receive Mode

For 16-bit non-split receive mode, if `EPPI_CTL.PACKEN` =1, the EPPI packs two 16-bit incoming data into one 32-bit word. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and

`EPPI_CTL.SKIPEO` bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the `EPPI_CTL.SWAPEN` bit setting.

Table 18-19: 16-Bit Receive Mode with Packing Enabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x1111						
0x2222	0x2222 1111	0x1111 2222				
0x3333			0x3333 1111		0x1111 3333	
0x4444	0x4444 3333	0x3333 4444		0x4444 2222		0x2222 4444
0x5555						
0x6666	0x6666 5555	0x5555 6666				
0x7777			0x7777 5555		0x5555 7777	
0x8888	0x8888 7777	0x7777 8888		0x8888 6666		0x6666 8888

Table 18-20: 16-bit Receive Mode with Packing Disabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=X
0x1111	0x1111	0x1111	
0x2222	0x2222		0x2222
0x3333	0x3333	0x3333	
0x4444	0x4444		0x4444
0x5555	0x5555	0x5555	
0x6666	0x6666		0x6666
0x7777	0x7777	0x7777	
0x8888	0x8888		0x8888

Configuring 18-Bit Receive Mode

For 18-bit non-split receive mode, if `EPPI_CTL.PACKEN` =0, the EPPI zero-fills or sign-extends the incoming data into a 32-bit word. If `EPPI_CTL.PACKEN` =1, the EPPI first zero-fills or sign-extends the incoming data to 24 bits, and then packs four such 24-bit data words into three 32-bit words. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The `EPPI_CTL.SWAPEN` bit has no effect.

Table 18-21: 18-bit Receive Mode with Packing Disabled

Pin Data (18 bits)	**DMA DATA** **SKIPEN=0** **SKIPEO=X** **SWAPEN=X** **SIGNEXT=0**	**DMA DATA** **SKIPEN=1** **SKIPEO=1** **SWAPEN=X** **SIGNEXT=0**	**DMA DATA** **SKIPEN=1** **SKIPEO=0** **SWAPEN=X** **SIGNEXT=0**
0x0 6666	0x0000 6666	0x0000 6666	
0x1 7777	0x0001 7777		0x0001 7777
0x2 8888	0x0002 8888	0x0002 8888	
0x3 9999	0x0003 9999		0x0003 9999

Table 18-22: 18-bit Receive Mode with Packing Enabled

Pin Data (18 bits)	**DMA DATA** **SKIPEN=0** **SKIPEO=X** **SWAPEN=X** **SIGNEXT=0**	**DMA DATA** **SKIPEN=1** **SKIPEO=1** **SWAPEN=X** **SIGNEXT=0**	**DMA DATA** **SKIPEN=1** **SKIPEO=0** **SWAPEN=X** **SIGNEXT=0**
0x0 1122			
0x1 3344	0x4400 1122		
0x2 5566	0x5566 0133	0x6600 1122	
0x3 7788	0x0377 8802		0x8801 3344
0x0 99AA		0x99AA 0255	
0x1 BBCC	0xCC00 99AA		0xBBCC 0377
0x2 DDEE	0xDDEE 01BB	0x02DD EE00	
0x3 FF12	0x03FF 122D		0x03FF 1201

Configuring 8-Bit Split Receive Mode

For 8-bit split receive mode, the `EPPI_CTL.PACKEN` and `EPPI_CTL.SIGNEXT` bits are not valid. The EPPI always packs 4 bytes of data into one 32-bit word.

Table 18-23: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 0

Pin Data (8 bits)	**SPLTEO=1 SUBSPLTODD= 0 SWAPEN=0 SKIPEN=0 SKIPEO=X**			**SPLTEO=1 SUBSPLTODD= 1 SWAPEN=0 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$U_1V_1U_0V_0$	$U_1V_1U_0V_0$			
Y_3	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
Y_6						
U_3		$U_3V_3U_2V_2$	$U_3V_3U_2V_2$		$U_3U_2U_1U_0$	
Y_7	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$
V_4						$U_3U_2U_1U_0$

Table 18-24: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1

Pin Data (8 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$V_0U_0V_1U_1$	$V_0U_0V_1U_1$			
Y_3	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_0V_1V_2V_3$	$V_0V_1V_2V_3$
Y_6						
U_3		$V_2U_2V_3U_3$	$V_2U_2V_3U_3$		$U_0U_1U_2U_3$	
Y_7	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$
V_4						$U_0U_1U_2U_3$

When the bits settings are `EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SUBSPLTODD` =1 and `EPPI_CTL.DMACFG` =0, the EPPI packs the second Chroma component sent over the DMA bus completely before the Luma component. However, it is intentionally held until that previous word is moved out. This functionality allows the separation of Luma and Chroma values into individual buffers when using 2D-DMA. The second Chroma component is U0U1U2U3 in the ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 0*** and ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1*** tables. The Luma component is Y4Y5Y6Y7 in the ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1*** table.

Configuring 10/12/14/16-Bit Split Receive Mode with SPLTWRD=0

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. For 10, 12, or 14-bit split receive modes, the EPPI first either sign-extends or zero-fills the incoming data into a 16-bit word. The EPPI then packs two of these words into one 32-bit word to send to the DMA.

Table 18-25: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 0

Pin Data (16 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X**			**SPLTEO=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	Primary DMA Channel	**Secondary DMA Channel**	**Primary DMA Channel**	**Primary DMA Channel**	**Secondary DMA Channel**	**Primary DMA Channel**
V_0						
Y_0						
U_0		U_0V_0	U_0V_0			
Y_1	Y_1Y_0		Y_1Y_0	Y_1Y_0		Y_1Y_0
V_1					V_1V_0	V_1V_0
Y_2						
U_1		U_1V_1	U_1V_1		U_1U_0	
Y_3	Y_3Y_2		Y_3Y_2	Y_3Y_2		Y_3Y_2
V_2						U_1U_0

Table 18-26: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 1

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL
V_0						
Y_0						
U_0		V_0U_0	V_0U_0			
Y_1	Y_0Y_1		Y_0Y_1	Y_0Y_1		Y_0Y_1
V_1					V_0V_1	V_0V_1
Y_2						
U_1		V_1U_1	V_1U_1		U_0U_1	
Y_3	Y_2Y_3		Y_2Y_3	Y_2Y_3		Y_2Y_3
V_2						U_0U_1

Configuring 16-Bit Split Receive Mode with SPLTWRD=1

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. The `EPPI_CTL.SPLTWRD` bit is only valid when the `EPPI_CTL.DLEN` bit =16 bits.

Table 18-27: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0Y_0						

Table 18-27: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0 (Continued)

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
U_0Y_1						
V_1Y_2						
U_1Y_3	$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$
V_2Y_4			$U_1V_1U_0V_0$			
U_2Y_5						
V_3Y_6					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
U_3Y_7	$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	$Y_7Y_6Y_5Y_4$
V_4Y_8			$U_3V_3U_2V_2$			$U_3U_2U_1U_0$

Configuring 8-Bit Transmit Mode

For 8-bit non-split transmit mode, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory into 4 bytes to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If `EPPI_CTL.PACKEN` =0, the DMA is a 16-bit DMA and the EPPI transmits the lower 8 bits. The `EPPI_CTL.SWAPEN` bit has no effect when `EPPI_CTL.PACKEN` =0.

Table 18-28: 8-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x11223344	0x44	0x11
0x55667788	0x33	0x22
	0x22	0x33
	0x11	0x44
	0x88	0x55
	0x77	0x66
	0x66	0x77

Table 18-28: 8-bit Transmit Mode with Packing Enabled (Continued)

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
	0x55	0x88

Table 18-29: Data Sent in 8-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x34
0x2345	0x45
0x3456	0x56

Configuring 10/12/14-Bit Transmit Modes

For 10, 12, or 14-bit non-split transmit modes, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA. The EPPI unpacks the 32-bit word from memory into two 16-bit data words. The EPPI then transmits the required LSBs from each data word. The EPPI transmits either the most significant word or the least significant word as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If `EPPI_CTL.PACKEN` =0, the DMA is a 16-bit DMA and the EPPI transmits the required LSBs. The `EPPI_CTL.SWAPEN` bit has no effect when the `EPPI_CTL.PACKEN` bit =0.

Table 18-30: 10-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x222	0x111
0x3333 4444	0x111	0x222
	0x044	0x333
	0x333	0x044

Table 18-31: 10-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x234
0x2345	0x345
0x3456	0x056
0x4567	0x167

Configuring 16-Bit Transmit Mode

For 16-bit non-split transmit mode, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA. The EPPI unpacks the 32-bit word from memory into two 16-bit data words to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If the `EPPI_CTL.PACKEN` bit =0, the DMA is a 16-bit DMA and the EPPI transmits the data as is. The `EPPI_CTL.SWAPEN` has no effect when `EPPI_CTL.PACKEN` bit =0.

Table 18-32: 16-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x2222	0x1111
0x3333 4444	0x1111	0x2222
	0x4444	0x3333
	0x3333	0x4444

Table 18-33: 16-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x1234
0x2345	0x2345
0x3456	0x3456

Configuring 18-Bit Transmit Mode

For 18-bit transmit mode, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory. In this mode, when `EPPI_CTL.RGBFMTEN` is set, the least significant 2 bits of R, G, and B are dropped.

Table 18-34: 18-bit Transmit Mode with Packing Enabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x1 EF01	0x3 3EC0
0x0123 4567	0x3 89AB	0x1 98AA
	0x1 2345	0x0 0211

Table 18-35: 18-bit Transmit Mode with Packing Disabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x3 CDEF	0x2 ACFB
0x0123 4567	0x3 4567	0x0 8459

Configuring 8-Bit Split Transmit Mode

For 8-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into 4 bytes to transmit.

Table 18-36: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0	$U_1V_1U_0V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
		U_0	$U_3V_3U_2V_2$	U_0
		Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3
		V_2		V_2
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 18-37: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0	$V_3V_2V_1V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
	$V_7V_6V_5V_4$	U_0	$U_3U_2U_1U_0$	U_0
	$U_7U_6U_5U_4$	Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3
		V_2		V_2

Table 18-37: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1 and SWAPEN=0 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 18-38: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	U_1	$U_1V_1U_0V_0$	U_1
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
		V_1	$U_3V_3U_2V_2$	V_1
		Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		U_0		U_0
		Y_1		Y_1
		V_0		V_0
		Y_0		Y_0
		U_3		U_3
		Y_7		Y_7
		V_3		V_3
		Y_6		Y_6
		U_2		U_2
		Y_5		Y_5
		V_2		V_3
		Y_4		Y_4

Table 18-39: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1, and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_3	$V_3V_2V_1V_0$	V_3
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
	$V_7V_6V_5V_4$	U_3	$U_3V_3U_2V_2$	U_3
	$U_7U_6U_5U_4$	Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		V_2		V_2
		Y_1		Y_1
		U_2		U_2
		Y_0		Y_0
		V_1		V_1
		Y_7		Y_7
		U_1		U_1
		Y_6		Y_6
		V_0		V_0
		Y_5		Y_5
		U_0		U_0
		Y_4		Y_4

Configuring 10/12/14/16-Bit Transmit Mode with SPLTWRD=0

For 16-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. For 10, 12, or 14-bit split transmit modes, the EPPI first unpacks the data in the same way as for 16-bit transmit mode. But, the EPPI transmits only the required number of LSBs.

Table 18-40: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	U_0V_0	V_0	U_0V_0	V_0
Y_3Y_2	U_1V_1	Y_0	Y_1Y_0	Y_0
		U_0	U_1V_1	U_0
		Y_1	Y_3Y_2	Y_1
		V_1		V_1

Table 18-40: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 0 (Continued)

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 18-41: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_1V_0	V_0	V_1V_0	V_0
Y_3Y_2	U_1U_0	Y_0	Y_1Y_0	Y_0
	V_3V_2	U_0	U_1U_0	U_0
	U_3U_2	Y_1	Y_3Y_2	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 18-42: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_0U_0	V_0	V_0U_0	V_0
Y_3Y_2	V_1U_1	Y_1	Y_1Y_0	Y_1
		U_0	V_1U_1	U_0
		Y_0	Y_3Y_2	Y_0
		V_1		V_1
		Y_3		Y_3
		U_1		U_1
		Y_2		Y_2

Table 18-43: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_1V_0	V_1	V_1V_0	V_1
Y_3Y_2	U_1U_0	Y_1	Y_1Y_0	Y_1
	V_3V_2	U_1	U_1U_0	U_1
	U_3U_2	Y_0		Y_0
		V_0		V_0
		Y_3		Y_1
		U_0		U_0
		Y_2		Y_2

Configuring 16-Bit Split Transmit Mode with SPLTWRD=1

For 16-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. The `EPPI_CTL.SPLTWRD` bit is only valid when the `EPPI_CTL.DLEN` bit =16 bits.

Table 18-44: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0Y_0	$U_1V_1U_0V_0$	V_0Y_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
		V_1Y_2	$U_3V_3U_2V_2$	V_1Y_2
		U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

Table 18-45: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
PRIMARY DMA DATA (32 bits)	**SECONDARY DMA DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0Y_0	$V_3V_2V_1V_0$	V_0Y_0

Table 18-45: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 1, and SWAPEN = 0 (Continued)

DMACFG = 1			DMACFG = 0	
PRIMARY DMA DATA (32 bits)	**SECONDARY DMA DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
	$V_7V_6V_5V_4$	V_1Y_2	$U_3U_2U_1U_0$	V_1Y_2
	$U_7U_6U_5U_4$	U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

EPPI Programming Concepts

This section provides information on SMPTE programming.

SMPTE Modes Programming

The programming model of SMPTE modes is similar to ITU Modes. All programming modes pertaining to ITU modes like XFRTYPE, FSCFG, FLDSEL, and BLANKGEN hold true for SMPTE modes as well. The only difference is that since ITU modes use Y-Cr/Cb interleaved data and SMPTE use parallel Y-Cr/Cb data, SPLTWRD could be set while operating in SMPTE modes. The ***Programming Modes for SMPTE Formats*** table describes the programming modes for different SMPTE formats.

Table 18-46: Programming Modes for SMPTE Formats

SMPTE Format	SMPTE Channel Width	EPPI Input Bit Width	EPPI Mode	Remarks
296M	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16 bits SPLTWRD = 1	SIGNEXT not supported
	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16 bits SPLTWRD = 1	SIGNEXT not supported

ADSP-SC58x EPPI Register Descriptions

Enhanced Parallel Peripheral Interface (EPPI) contains the following registers.

Table 18-47: ADSP-SC58x EPPI Register List

Name	Description
`EPPI_CLKDIV`	Clock Divide Register
`EPPI_CTL`	Control Register
`EPPI_CTL2`	Control Register 2 Register
`EPPI_EVENCLIP`	Clipping Register for EVEN (Luma) Data Register
`EPPI_FRAME`	Lines Per Frame Register
`EPPI_FS1_DLY`	Frame Sync 1 Delay Value Register
`EPPI_FS1_PASPL`	FS1 Period Register / EPPI Active Samples Per Line Register
`EPPI_FS1_WLHB`	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
`EPPI_FS2_DLY`	Frame Sync 2 Delay Value Register
`EPPI_FS2_PALPF`	FS2 Period Register / EPPI Active Lines Per Field Register
`EPPI_FS2_WLVB`	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
`EPPI_HCNT`	Horizontal Transfer Count Register
`EPPI_HDLY`	Horizontal Delay Count Register
`EPPI_IMSK`	Interrupt Mask Register
`EPPI_LINE`	Samples Per Line Register
`EPPI_ODDCLIP`	Clipping Register for ODD (Chroma) Data Register
`EPPI_STAT`	Status Register
`EPPI_VCNT`	Vertical Transfer Count Register
`EPPI_VDLY`	Vertical Delay Count Register

Clock Divide Register

The `EPPI_CLKDIV` register provides the divisor for EPPI internal clock generation. The generated clock frequency is given by following formula:

`EPPI_CLK` = (SCLK1_0) / (`EPPI_CLKDIV` + 1)

Note that a value of 0xFFFF is invalid for the `EPPI_CLKDIV` register.

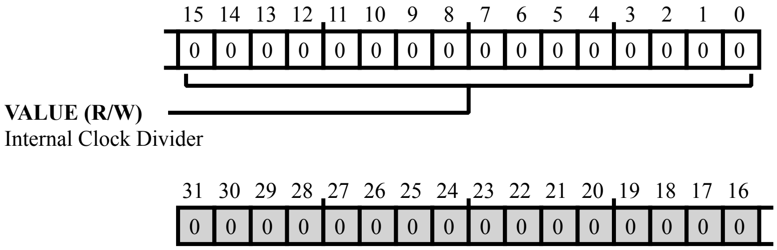

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Internal Clock Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-14: EPPI_CLKDIV Register Diagram

Table 18-48: EPPI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Internal Clock Divider.

Control Register

The `EPPI_CTL` register configures the EPPI for operating mode, control signal polarities, and data width of the port.

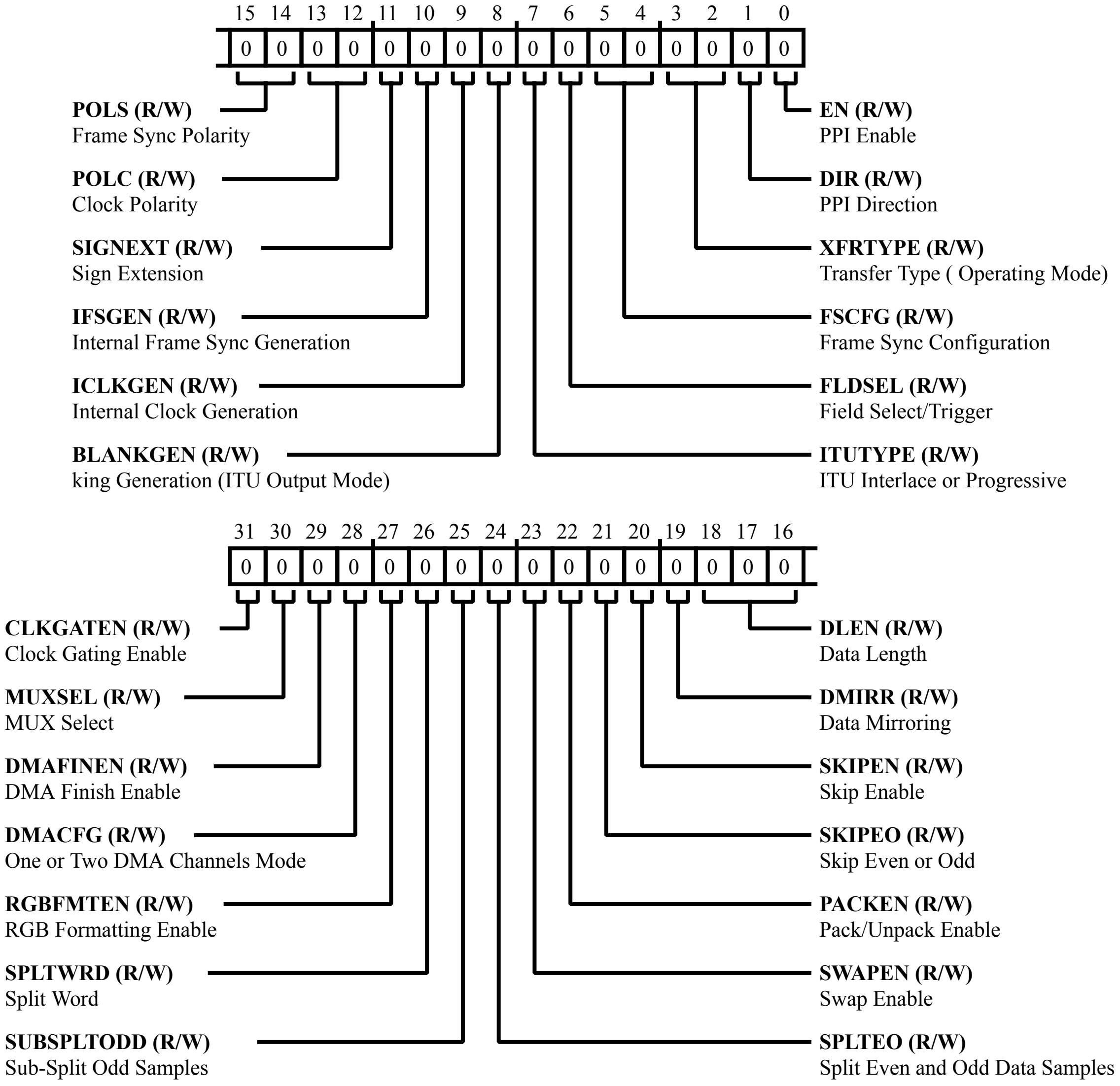

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
POLS (R/W)
Frame Sync Polarity
POLC (R/W)
Clock Polarity
SIGNEXT (R/W)
Sign Extension
IFSGEN (R/W)
Internal Frame Sync Generation
ICLKGEN (R/W)
Internal Clock Generation
BLANKGEN (R/W)
king Generation (ITU Output Mode)
EN (R/W)
PPI Enable
DIR (R/W)
PPI Direction
XFRTYPE (R/W)
Transfer Type (Operating Mode)
FSCFG (R/W)
Frame Sync Configuration
FLDSEL (R/W)
Field Select/Trigger
ITUTYPE (R/W)
ITU Interlace or Progressive
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKGATEN (R/W)
Clock Gating Enable
MUXSEL (R/W)
MUX Select
DMAFINEN (R/W)
DMA Finish Enable
DMACFG (R/W)
One or Two DMA Channels Mode
RGBFMTEN (R/W)
RGB Formatting Enable
SPLTWRD (R/W)
Split Word
SUBSPLTODD (R/W)
Sub-Split Odd Samples
DLEN (R/W)
Data Length
DMIRR (R/W)
Data Mirroring
SKIPEN (R/W)
Skip Enable
SKIPEO (R/W)
Skip Even or Odd
PACKEN (R/W)
Pack/Unpack Enable
SWAPEN (R/W)
Swap Enable
SPLTEO (R/W)
Split Even and Odd Data Samples

Figure 18-15: EPPI_CTL Register Diagram

Table 18-49: EPPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKGATEN	Clock Gating Enable. The `EPPI_CTL.CLKGATEN` bit enables using the `EPPI_FS3` pin as a clock gating pin. When `EPPI_CTL.CLKGATEN` is set, the `EPPI_FS3` pin acts as a clock gating signal, and both the internal and external clock are gated. Note that the `EPPI_FS3` pin gating signal is active low, and the `EPPI_CTL.CLKGATEN` selection is ignored if `EPPI_CTL.MUXSEL` is set or `EPPI_CTL.FSCFG` equals 0x3.	
		0	Disable
		1	Enable
30 (R/W)	MUXSEL	MUX Select. The `EPPI_CTL.MUXSEL` bit enables multiplexing of a primary and alternate camera using the EPPI main data and clock lines. For more information on this feature, see the EPPI functional description.	
		0	Normal Operation
		1	Multiplexed Operation
29 (R/W)	DMAFINEN	DMA Finish Enable. The `EPPI_CTL.DMAFINEN` bit selects whether or not the EPPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely.	
		0	No Finish Command
		1	Enable Send Finish Command
28 (R/W)	DMACFG	One or Two DMA Channels Mode. The `EPPI_CTL.DMACFG` bit is valid only if `EPPI_CTL.SPLTEO` is set. If `EPPI_CTL.DMACFG` is set, the EPPI uses two DMA channels. And, if `EPPI_CTL.DMACFG` is cleared, the EPPI uses only one DMA channel.	
		0	PPI Uses One DMA Channel
		1	PPI Uses Two DMA Channels
27 (R/W)	RGBFMTEN	RGB Formatting Enable. For 16- or 18-bit transmit modes only, the `EPPI_CTL.RGBFMTEN` bit enables conversion of RGB888 from memory into RGB666 output data (18-bit transmit) or enables conversion of RGB888 from memory into RGB565 output data (16-bit transmit). Note that `EPPI_CTL.SPLTEO` and `EPPI_CTL.RGBFMTEN` should never be set simultaneously.	
		0	Disable RGB Formatted Output
		1	Enable RGB Formatted Output

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	

26
(R/W)
	SPLTWRD	Split Word.
		The `EPPI_CTL.SPLTWRD` bit selects split word data placement when the data length (`EPPI_CTL.DLEN`) selects 16-, 20-, or 24-bit data words. For all other `EPPI_CTL.SPLTWRD` values, the set or clear selections for `EPPI_CTL.SPLTWRD` produce the same result (act as though `EPPI_CTL.SPLTWRD` is cleared). For `EPPI_CTL.SPLTWRD` set, the `EPPI_CTL.DLEN` values below result in the following combinations of split words:
		`DLEN     Cr/Cb data       Y data`
		` 16   PPI_DATA[15:8]   PPI_DATA[7:0]`
		` 20   PPI_DATA[19:10]  PPI_DATA[9:0]`
		` 24   PPI_DATA[23:12]  PPI_DATA[11:0]`	

		0	PPI_DATA has (DLEN-1) bits of Y or Cr or Cb
		1	PPI_DATA Contains 2 Elements per Word
25 (R/W)	SUBSPLTODD	Sub-Split Odd Samples. The `EPPI_CTL.SUBSPLTODD` bit is valid only if `EPPI_CTL.SPLTEO` is set. If `EPPI_CTL.SUBSPLTODD` is set, the EPPI sub-splits the odd sub-stream, and packs them separately.	
		0	Disable
		1	Enable
24 (R/W)	SPLTEO	Split Even and Odd Data Samples. If `EPPI_CTL.SPLTEO` is set, the EPPI splits the incoming data stream into two sub-streams, an even stream and an odd stream, and packs them separately.	
		0	Do Not Split Samples
		1	Split Even/Odd Samples
23 (R/W)	SWAPEN	Swap Enable. The `EPPI_CTL.SWAPEN` selects whether or not to swap the order of the first data (most-significant bits versus least-significant bits) of the DMA word. For receive modes, the EPPI puts the first data in the most significant bits (if set) or puts the first data in the least significant bits (if cleared) of the DMA word. For transmit modes, the EPPI transmits the most significant bits in the DMA word as the first data (if set) or transmits the least significant bits in the DMA word as the first data (if cleared).	
		0	Disable
		1	Enable

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	PACKEN	Pack/Unpack Enable. The `EPPI_CTL.PACKEN` select whether or not packing is enabled (for receive modes) and unpacking is enabled (for transmit modes). When this bit is set, EPPI transfer DMA is 32-bits wide. When this bit is cleared and the `EPPI_CTL.DLEN` is less than or equal to 16 bits, EPPI transfer DMA is 16-bits wide. For receive modes, if this bit is set, then the EPPI packs the incoming data into 32-bit words. If this bit is cleared, then the EPPI does not do any packing. For transmit modes, if this bit is set, then the EPPI always unpacks the 32-bit data from DMA. If this bit is not set, the EPPI does not do any unpacking.	
		0	Disable
		1	Enable
21 (R/W)	SKIPEO	Skip Even or Odd. The `EPPI_CTL.SKIPEO` bit selects whether even (if set) or odd (if cleared) samples are skipped if sample skipping is enabled (`EPPI_CTL.SKIPEN` is set). This feature only is useful for receive modes.	
		0	Skip Odd Samples
		1	Skip Even Samples
20 (R/W)	SKIPEN	Skip Enable. The `EPPI_CTL.SKIPEN` bit enables skipping alternate samples. This feature only is useful for receive modes.	
		0	No Samples Skipping
		1	Skip Alternate Samples
19 (R/W)	DMIRR	Data Mirroring. The `EPPI_CTL.DMIRR` field enables mirroring (bit reversing) of the data coming in or going out on the EPPI data pins.	

```
Pin      PPI Data      PPI Data
Data   (DAT_MRR=0)   (DAT_MRR=1)
------------------------------
 15        15            0
 14        14            1
....      ....          ....
  1         1           14
  0         0           15
```

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	No Data Mirroring
		1	Data Mirroring
18:16 (R/W)	DLEN	Data Length. The `EPPI_CTL.DLEN` bits select the data length for the EPPI. Note that the 20 bits data length selection is valid only for SMPTE modes (`EPPI_CTL.SPLTWRD` set).	

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	8 bits
		1	10 bits
		2	12 bits
		3	14 bits
		4	16 bits
		5	18 bits
		6	20 bits
		7	24 bits
15:14 (R/W)	POLS	Frame Sync Polarity. The `EPPI_CTL.POLS` selects whether the frame syncs' polarity is active low versus active high.	
		0	FS1 and FS2 are active high
		1	FS1 is active low. FS2 is active high
		2	FS1 is active high. FS2 is active low
		3	FS1 and FS2 are active low
13:12 (R/W)	POLC	Clock Polarity. The `EPPI_CTL.POLC` selects the rising versus falling edge for sampling data and sampling/driving syncs.	
		0	Clock/Sync Polarity Mode 0. For receive mode: Sample data on falling edge and sample/drive syncs on falling edge. For transmit mode: Drive data on rising edge and sample/drive syncs on rising edge.
		1	Clock/Sync Polarity Mode 1. For receive mode: Sample data on falling edge and sample/drive syncs on rising edge. For transmit mode: Drive data on rising edge and sample/drive syncs on falling edge.
		2	Clock/Sync Polarity Mode 2. For receive mode: Sample data on rising edge and sample/drive syncs on falling edge. For transmit mode: Drive data on falling edge and sample/drive syncs on rising edge.
		3	Clock/Sync Polarity Mode 3. For receive mode: Sample data on rising edge and sample/drive syncs on rising edge. For transmit mode: Drive data on falling edge and sample/drive syncs on falling edge.

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	SIGNEXT	Sign Extension. The `EPPI_CTL.SIGNEXT` select whether (for receive modes when `EPPI_CTL.DLEN` selecting 16 bit data length) the data is sign extended or zero filled. Not that `EPPI_CTL.SPLTWRD` is removed from this shared bit.	
		0	Zero Filled
		1	Sign Extended
10 (R/W)	IFSGEN	Internal Frame Sync Generation. The `EPPI_CTL.IFSGEN` bit selects whether the frame syncs are generated internally or are supplied by an external device.	
		0	External Frame Sync
		1	Internal Frame Sync
9 (R/W)	ICLKGEN	Internal Clock Generation. The `EPPI_CTL.ICLKGEN` bit selects whether the `EPPI_CLK` is generated internally or is supplied by an external device.	
		0	External Clock
		1	Internal Clock
8 (R/W)	BLANKGEN	king Generation (ITU Output Mode). The `EPPI_CTL.BLANKGEN` enables ITU output with internal blanking. In GP 8, 10 transmit bit modes (when `EPPI_CTL.SPLTWRD` is cleared) and 16-, 20-, and 24-bit transmit modes (when `EPPI_CTL.SPLTWRD` is set), `EPPI_CTL.BLANKGEN` selects whether or not the EPPI generates blanking and generates preamble and insertion with active data from memory.	
		0	Disable
		1	Enable
7 (R/W)	ITUTYPE	ITU Interlace or Progressive. The `EPPI_CTL.ITUTYPE` selects interlaced or progressive operation for ITU656 mode. This selection is valid for both TX and RX modes.	
		0	Interlaced
		1	Progressive

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FLDSEL	Field Select/Trigger. The `EPPI_CTL.FLDSEL` bits configure the EPPI field and trigger selection. These are valid for GP modes (`EPPI_CTL.XFRTYPE` =0x3) and ITU656 active video mode (`EPPI_CTL.XFRTYPE` cleared).	
		0	Field Mode 0. Read field 1 (for ITU656 active video mode). Set internal trigger (for GP RX mode). FS3 is toggled on FS2 assertion followed by FS1 assertion (when the `EPPI_CTL.FSCFG` bit selects sync mode 3 and the `EPPI_CTL.IFSGEN` bit selects internal frame sync).
		1	Field Mode 1 Read field 1 and field 2 (ITU656 active video mode). Set external trigger (GP RX mode). FS3 is toggled on FS2 assertion (when the `EPPI_CTL.FSCFG` bit selects sync mode 3 and the `EPPI_CTL.IFSGEN` bit selects internal frame sync).
5:4 (R/W)	FSCFG	Frame Sync Configuration. The `EPPI_CTL.FSCFG` bits configure the EPPI frame syncs. These are valid only for GP modes (`EPPI_CTL.XFRTYPE` =0x3). The output of the frames syncs also depends on whether the EPPI transfer direction is transmit and the EPPI is in ITU output mode (`EPPI_CTL.BLANKGEN` is set).	
		0	Sync Mode 0. FS0 driven in GP mode. FS0 not driven in ITU output mode.
		1	Sync Mode 1. FS1 driven in GP mode. HSYNC driven on FS1 in ITU output mode.
		2	Sync Mode 2. FS2 driven in GP mode. HSYNC driven on FS1 and VSYNC driven on FS1 in ITU output mode.
		3	Sync Mode 3. FS3 driven in GP mode. HSYNC driven on FS1, VSYNC driven on FS2, and FIELD driven on FS3 in ITU output mode.

Table 18-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	XFRTYPE	Transfer Type (Operating Mode). The `EPPI_CTL.XFRTYPE` bits select the EPPI operating mode. In receive mode (`EPPI_CTL.DIR` cleared), the EPPI modes include ITU656 active video only mode, ITU656 entire field mode, ITU656 vertical blanking only mode, and non-ITU656 mode (GP mode). For transmit mode (`EPPI_CTL.DIR` set), the `EPPI_CTL.XFRTYPE` bits have no effect, and the EPPI (in transmit mode) is always in GP mode.	
		0	ITU656 Active Video Only Mode
		1	ITU656 Entire Field Mode
		2	ITU656 Vertical Blanking Only Mode
		3	Non-ITU656 Mode (GP Mode)
1 (R/W)	DIR	PPI Direction. The `EPPI_CTL.DIR` bit selects whether the EPPI is in receive mode (if cleared) or in transmit mode (if set).	
		0	Receive Mode
		1	Transmit Mode
0 (R/W)	EN	PPI Enable. The `EPPI_CTL.EN` bit enables or disables the EPPI.	
		0	Disable
		1	Enable

Control Register 2 Register

The `EPPI_CTL2` register controls HSYNC finish signal generation.

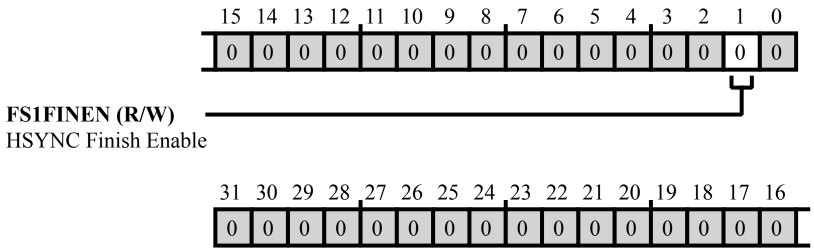

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS1FINEN (R/W)
HSYNC Finish Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-16: EPPI_CTL2 Register Diagram

Table 18-50: EPPI_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FS1FINEN	HSYNC Finish Enable. The `EPPI_CTL2.FS1FINEN` bit selects whether (if set) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a LINE is received completely or (if cleared) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely. Note that the `EPPI_CTL.DMAFINEN` bit must be set for the EPPI to generate either of the finish commands.	
		0	Finish sent after frame RX done. PPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely
		1	Finish sent after frame/line RX done. PPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely.

Clipping Register for EVEN (Luma) Data Register

The `EPPI_EVENCLIP` register selects the clipping threshold for luma data, which provides clipping of individual video components.

The high even and low even spaces in `EPPI_EVENCLIP` are 16-bits wide and (depending on the `EPPI_CTL.DLEN` bit selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits [25:16] constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with the `EPPI_CTL.SPLTWRD` bit set, the luma data gets the clipping threshold levels of the `EPPI_EVENCLIP` register, and the chroma data gets the clipping threshold levels of the `EPPI_ODDCLIP` register.

Also, note that the `EPPI_EVENCLIP` and `EPPI_ODDCLIP` registers are ignored when the `EPPI_CTL.RGBFMTEN` bit is set.

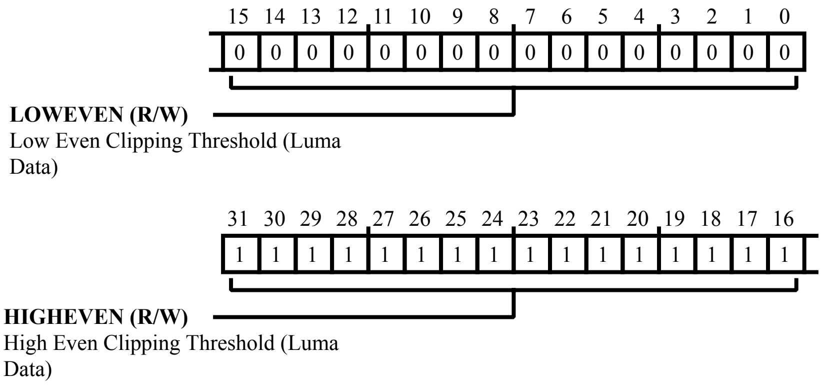

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOWEVEN (R/W)
Low Even Clipping Threshold (Luma Data)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
HIGHEVEN (R/W)
High Even Clipping Threshold (Luma Data)

Figure 18-17: EPPI_EVENCLIP Register Diagram

Table 18-51: EPPI_EVENCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHEVEN	High Even Clipping Threshold (Luma Data). The `EPPI_EVENCLIP.HIGHEVEN` bit field selects the clipping threshold for luma data. The high even spaces are 16-bits wide and (depending on the `EPPI_CTL.DLEN` selection) only the corresponding video component bits are considered for clipping.
15:0 (R/W)	LOWEVEN	Low Even Clipping Threshold (Luma Data). The `EPPI_EVENCLIP.LOWEVEN` bit field selects the clipping threshold for luma data. The low even spaces are 16-bits wide and (depending on the `EPPI_CTL.DLEN` selection) only the corresponding video component bits are considered for clipping.

Lines Per Frame Register

The `EPPI_FRAME` register tracks the frame track overflow and underflow errors. This register should be programmed with the number of lines expected per frame. Any write to the `EPPI_FRAME` register will also write the same value to the `EPPI_VCNT` register. But, any write to the `EPPI_VCNT` register does not affect the `EPPI_FRAME` register value. So the `EPPI_FRAME` register should be programmed before the `EPPI_VCNT` register.

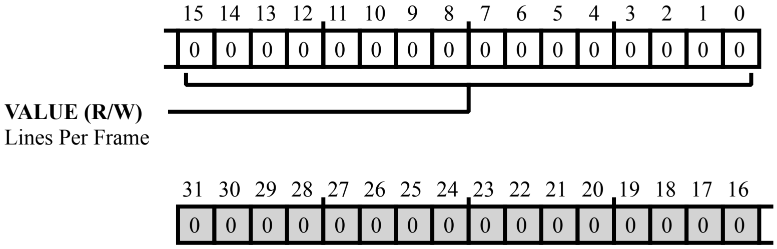

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Lines Per Frame
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-18: EPPI_FRAME Register Diagram

Table 18-52: EPPI_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lines Per Frame. The `EPPI_FRAME.VALUE` holds the number of lines expected per frame of data.

Frame Sync 1 Delay Value Register

The `EPPI_FS1_DLY` register selects the delay count (based on the period of the `EPPI_CLK` clock) between the first rising edge of `EPPI_CLK` after the EPPI is enabled and the first active edge of the associated frame sync when the internal frame sync is used.

Note that if the `EPPI_FS1_DLY` or `EPPI_FS2_DLY` registers are programmed with value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

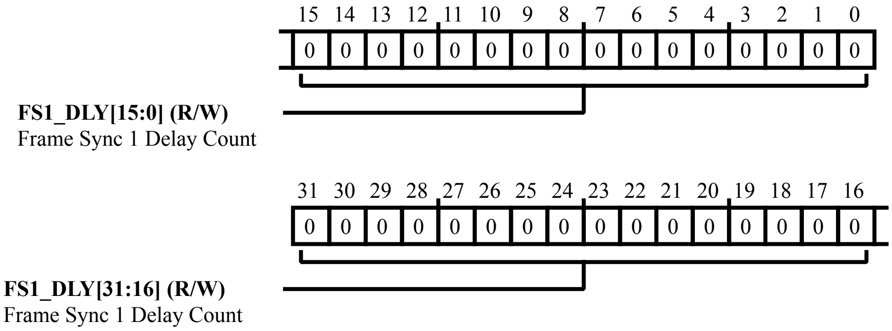

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS1_DLY[15:0] (R/W)
Frame Sync 1 Delay Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS1_DLY[31:16] (R/W)
Frame Sync 1 Delay Count

Figure 18-19: EPPI_FS1_DLY Register Diagram

Table 18-53: EPPI_FS1_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS1_DLY	Frame Sync 1 Delay Count. The `EPPI_FS1_DLY.FS1_DLY` bit field selects the delay count.

FS1 Period Register / EPPI Active Samples Per Line Register

The `EPPI_FS1_PASPL` register content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2, or 3 FS modes, the `EPPI_FS1_PASPL` register is used for the generation of frame sync 1. The register contains the period required for `EPPI_FS1` based on the `EPPI_CLK` clock.

In GP transmit mode with the `EPPI_CTL.BLANKGEN` bit set, this register contains the number of samples of active video or vertical blanking samples per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for this register is illegal. If programmed as 0, the EPPI regards the `EPPI_FS1_PASPL` register as containing 1.

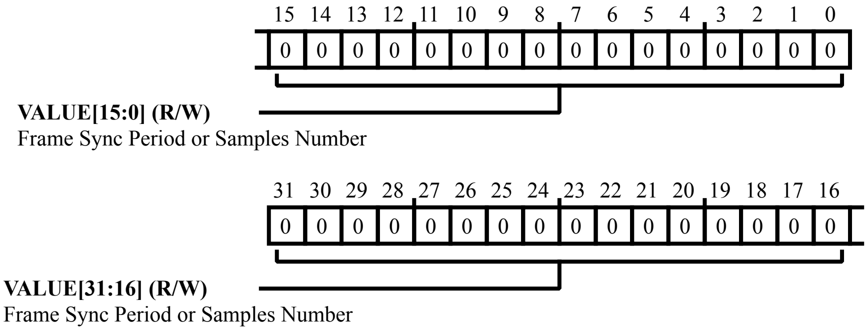

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Frame Sync Period or Samples Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Frame Sync Period or Samples Number

Figure 18-20: EPPI_FS1_PASPL Register Diagram

Table 18-54: EPPI_FS1_PASPL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Period or Samples Number. In GP 1, 2, or 3 FS modes, the `EPPI_FS1_PASPL.VALUE` bit field is used for the generation of frame sync 1 and contains the period required for `EPPI_FS1` based on the `EPPI_CLK` clock. In GP transmit mode with the `EPPI_CTL.BLANKGEN` bit set, this bit field contains the number of samples of active video or vertical blanking samples per line.

FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register

The `EPPI_FS1_WLHB` register's content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2 or 3 FS modes, `EPPI_FS1_WLHB` is used for the generation of frame sync 1. The register contains the width required for `EPPI_FS1` based on the `EPPI_CLK` clock.

In GP transmit mode with the `EPPI_CTL.BLANKGEN` bit set, this register contains the number of samples of horizontal blanking per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for the `EPPI_FS1_WLHB` register is illegal. If programmed as 0, the EPPI regards the `EPPI_FS1_WLHB` register as containing 1.

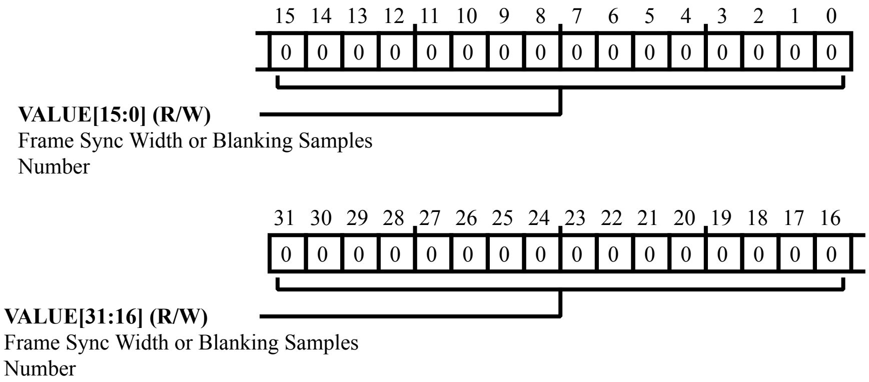

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Frame Sync Width or Blanking Samples Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Frame Sync Width or Blanking Samples Number

Figure 18-21: EPPI_FS1_WLHB Register Diagram

Table 18-55: EPPI_FS1_WLHB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Width or Blanking Samples Number. The `EPPI_FS1_WLHB.VALUE` bit field content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode. In GP 1, 2 or 3 FS modes, the `EPPI_FS1_WLHB.VALUE` bit field is used for the generation of frame sync 1. The register contains the width required for `EPPI_FS1` based on the `EPPI_CLK` clock. In GP transmit mode with `EPPI_CTL.BLANKGEN` set, this bit field contains the number of samples of horizontal blanking per line.

Frame Sync 2 Delay Value Register

The `EPPI_FS2_DLY` register selects the delay count (based on the period of the `EPPI_CLK` clock) between the first rising edge of `EPPI_CLK` after EPPI enabled and the first active edge of the associated frame sync when the internal frame sync is used.

Note that if the `EPPI_FS1_DLY` or `EPPI_FS2_DLY` registers are programmed with the value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

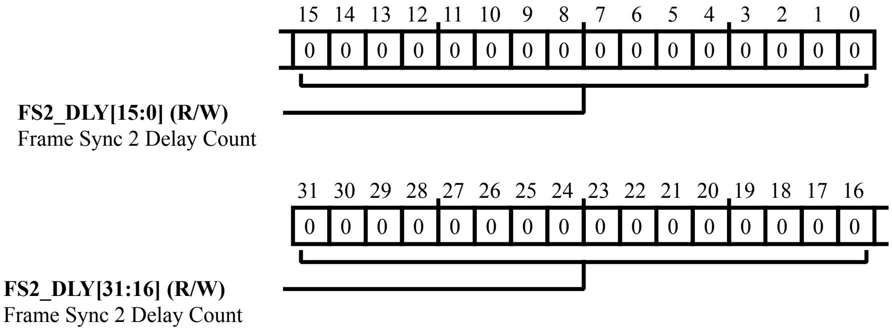

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS2_DLY[15:0] (R/W)
Frame Sync 2 Delay Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FS2_DLY[31:16] (R/W)
Frame Sync 2 Delay Count

Figure 18-22: EPPI_FS2_DLY Register Diagram

Table 18-56: EPPI_FS2_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS2_DLY	Frame Sync 2 Delay Count. The `EPPI_FS2_DLY.FS2_DLY` bit field selects the delay count.

FS2 Period Register / EPPI Active Lines Per Field Register

The `EPPI_FS2_PALPF` register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, `EPPI_FS2_PALPF` is used for the generation of frame sync 2. This register contains the period required for `EPPI_FS2` based on the `EPPI_CLK` clock.

In GP transmit mode with the `EPPI_CTL.BLANKGEN` bit set, this register contains the number of lines of active video per field.

Note that a value of 0 for the `EPPI_FS2_PALPF.F1ACT` or `EPPI_FS2_PALPF.F2ACT` bits is illegal. If either is programmed as 0, the EPPI regards the 0 value fields as containing 1.

Also, note that for progressive video, the `EPPI_FS2_PALPF.F2ACT` bit is ignored.

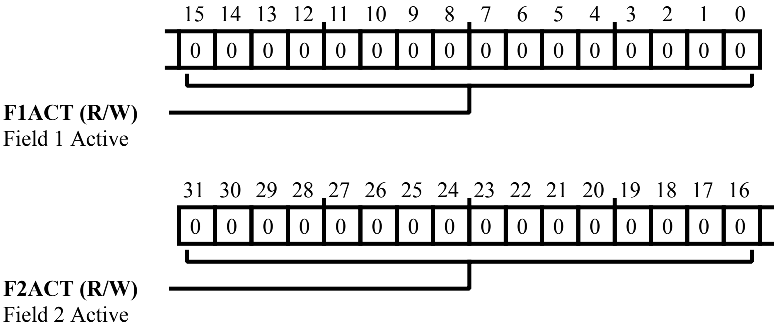

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F1ACT (R/W)
Field 1 Active
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F2ACT (R/W)
Field 2 Active

Figure 18-23: EPPI_FS2_PALPF Register Diagram

Table 18-57: EPPI_FS2_PALPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	F2ACT	Field 2 Active. The `EPPI_FS2_PALPF.F2ACT` bit field contains the number of lines of active data in field 2.
15:0 (R/W)	F1ACT	Field 1 Active. The `EPPI_FS2_PALPF.F1ACT` bit field contains the number of lines of active data in field 1.

FS2 Width Register / EPPI Lines Of Vertical Blanking Register

The `EPPI_FS2_WLVB` register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, the `EPPI_FS2_WLVB` register is used for the generation of frame sync 2. The register contains the width required for `EPPI_FS2` based on the `EPPI_CLK` clock.

In GP transmit mode with the `EPPI_CTL.BLANKGEN` bit set, this register contains the number or lines of vertical blanking.

Note that for progressive video, the `EPPI_FS2_WLVB.F2VBBD` and `EPPI_FS2_WLVB.F2VBAD` bits are ignored.

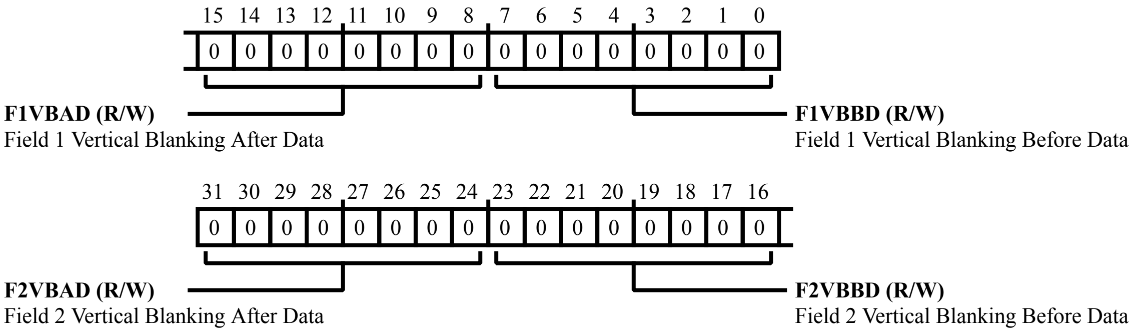

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F1VBAD (R/W)
Field 1 Vertical Blanking After Data
F1VBBD (R/W)
Field 1 Vertical Blanking Before Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F2VBAD (R/W)
Field 2 Vertical Blanking After Data
F2VBBD (R/W)
Field 2 Vertical Blanking Before Data

Figure 18-24: EPPI_FS2_WLVB Register Diagram

Table 18-58: EPPI_FS2_WLVB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	F2VBAD	Field 2 Vertical Blanking After Data. The `EPPI_FS2_WLVB.F2VBAD` bit field contains the number of lines of vertical blanking after field 2.
23:16 (R/W)	F2VBBD	Field 2 Vertical Blanking Before Data. The `EPPI_FS2_WLVB.F2VBBD` bit field contains the number of lines of vertical blanking before field 2.
15:8 (R/W)	F1VBAD	Field 1 Vertical Blanking After Data. The `EPPI_FS2_WLVB.F1VBAD` bit field contains the number of lines of vertical blanking after field 1.
7:0 (R/W)	F1VBBD	Field 1 Vertical Blanking Before Data. The `EPPI_FS2_WLVB.F1VBBD` bit field contains the number of lines of vertical blanking before field 1.

Horizontal Transfer Count Register

The `EPPI_HCNT` register holds the number of samples to read in or write out per line, after `EPPI_HDLY` number of cycles have expired since the assertion of `EPPI_FS1`. Any write to the `EPPI_LINE` register modifies the `EPPI_HCNT` register. But, any write to the `EPPI_HCNT` register does not affect the `EPPI_LINE` register value. So the `EPPI_HCNT` register should be programmed after the `EPPI_LINE` register.

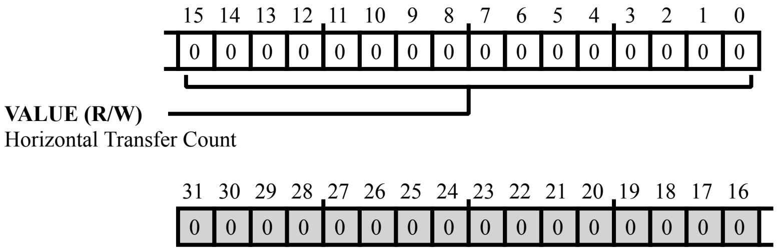

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Horizontal Transfer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-25: EPPI_HCNT Register Diagram

Table 18-59: EPPI_HCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Transfer Count. The `EPPI_HCNT.VALUE` holds the number of samples to read in or write out per line, after `EPPI_HDLY` number of cycles have expired since the last assertion of `EPPI_FS1`.

Horizontal Delay Count Register

The EPPI_HDLY register contains the number of clock cycles to delay after the assertion of EPPI_FS1 is detected before starting to read or write data.

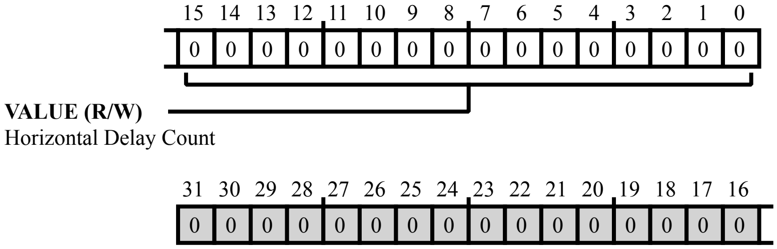

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Horizontal Delay Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-26: EPPI_HDLY Register Diagram

Table 18-60: EPPI_HDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Delay Count. The EPPI_HDLY.VALUE holds the number of EPPI_CLK cycles to delay after assertion of EPPI_FS1 before starting to read or write data.

Interrupt Mask Register

The `EPPI_IMSK` register permits the masking (if associated bit is set) of EPPI error interrupts for YFIFO underflow or overflow, CFIFO underflow or overflow, line track overflow error, line track underflow error, frame track overflow error, frame track underflow error, and ERR_NCOR (ITU preamble error not corrected. These conditions are flagged in the `EPPI_STAT` register and cleared by write-1-to-clear.

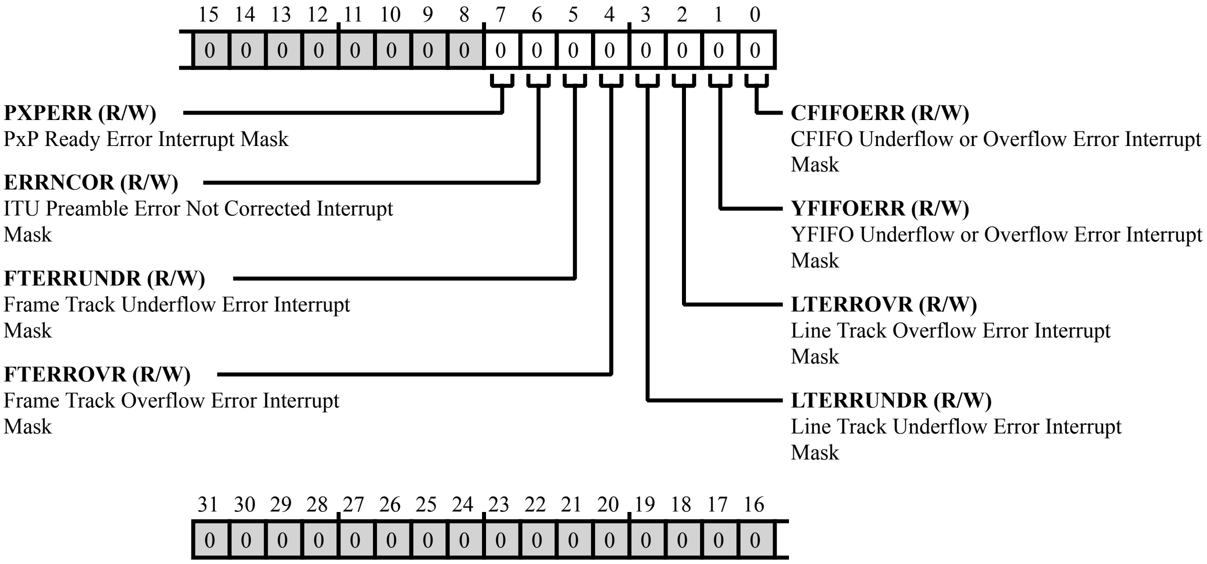

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PXPERR (R/W)
PxP Ready Error Interrupt Mask
ERRNCOR (R/W)
ITU Preamble Error Not Corrected Interrupt Mask
FTERRUNDR (R/W)
Frame Track Underflow Error Interrupt Mask
FTERROVR (R/W)
Frame Track Overflow Error Interrupt Mask
CFIFOERR (R/W)
CFIFO Underflow or Overflow Error Interrupt Mask
YFIFOERR (R/W)
YFIFO Underflow or Overflow Error Interrupt Mask
LTERROVR (R/W)
Line Track Overflow Error Interrupt Mask
LTERRUNDR (R/W)
Line Track Underflow Error Interrupt Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-27: EPPI_IMSK Register Diagram

Table 18-61: EPPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PXPERR	PxP Ready Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
6 (R/W)	ERRNCOR	ITU Preamble Error Not Corrected Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
5 (R/W)	FTERRUNDR	Frame Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
4 (R/W)	FTERROVR	Frame Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Table 18-61: EPPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	LTERRUNDR	Line Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
2 (R/W)	LTERROVR	Line Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
1 (R/W)	YFIFOERR	YFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
0 (R/W)	CFIFOERR	CFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Samples Per Line Register

The `EPPI_LINE` register tracks the line track overflow and underflow errors. This register should be programmed with the number of samples expected per line. Any write to the `EPPI_LINE` register will also write the same value to the `EPPI_HCNT` register. However, any write to the `EPPI_HCNT` register does not affect the `EPPI_LINE` register value. So the `EPPI_LINE` register should be programmed before the `EPPI_HCNT` register.

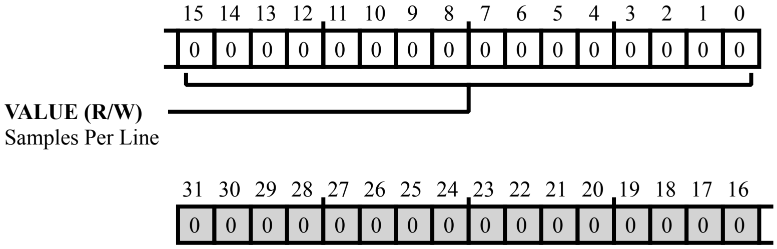

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Samples Per Line
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-28: EPPI_LINE Register Diagram

Table 18-62: EPPI_LINE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Samples Per Line. The `EPPI_LINE.VALUE` holds the number of samples expected per line.

Clipping Register for ODD (Chroma) Data Register

The `EPPI_ODDCLIP` register selects the clipping threshold for chroma data, which provides clipping of individual video components.

The high odd and low odd spaces in `EPPI_ODDCLIP` are 16-bits wide and (depending on the `EPPI_CTL.DLEN` bit selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits [25:16] constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with the `EPPI_CTL.SPLTWRD` bit set, the luma data gets the clipping threshold levels of the `EPPI_EVENCLIP` register, and the chroma data gets the clipping threshold levels of the `EPPI_ODDCLIP` register.

Also, note that the `EPPI_EVENCLIP` and `EPPI_ODDCLIP` registers are ignored when the `EPPI_CTL.RGBFMTEN` bit is set.

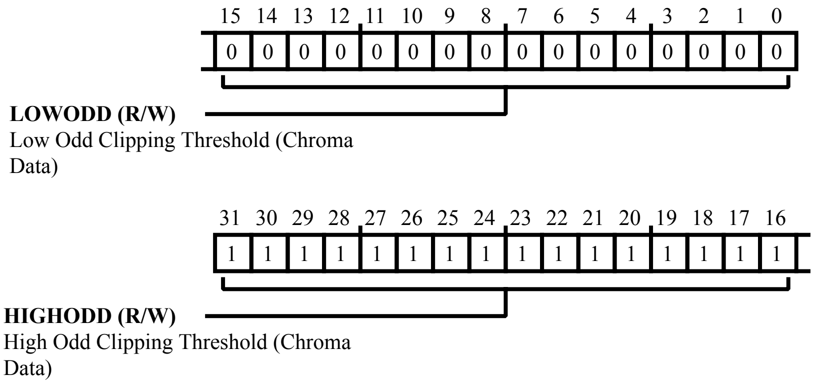

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOWODD (R/W)
Low Odd Clipping Threshold (Chroma Data)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
HIGHODD (R/W)
High Odd Clipping Threshold (Chroma Data)

Figure 18-29: EPPI_ODDCLIP Register Diagram

Table 18-63: EPPI_ODDCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHODD	High Odd Clipping Threshold (Chroma Data). The `EPPI_ODDCLIP.HIGHODD` bit field selects the clipping threshold for luma data. The high odd spaces are 16-bits wide and (depending on the `EPPI_CTL.DLEN` selection) only the corresponding video component bits are considered for clipping.
15:0 (R/W)	LOWODD	Low Odd Clipping Threshold (Chroma Data). The `EPPI_ODDCLIP.LOWODD` bit field selects the clipping threshold for luma data. The low add spaces are 16-bits wide and (depending on the `EPPI_CTL.DLEN` selection) only the corresponding video component bits are considered for clipping.

Status Register

The EPPI_STAT register contains bits that provide information about the current operating state of the EPPI.

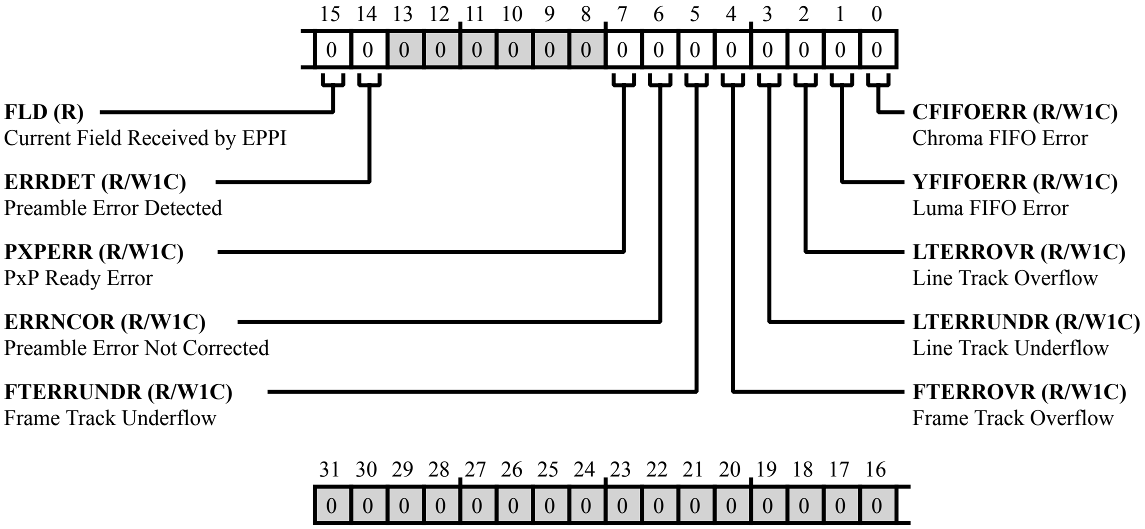

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FLD (R)
Current Field Received by EPPI
ERRDET (R/W1C)
Preamble Error Detected
PXPERR (R/W1C)
PxP Ready Error
ERRNCOR (R/W1C)
Preamble Error Not Corrected
FTERRUNDR (R/W1C)
Frame Track Underflow
CFIFOERR (R/W1C)
Chroma FIFO Error
YFIFOERR (R/W1C)
Luma FIFO Error
LTERROVR (R/W1C)
Line Track Overflow
LTERRUNDR (R/W1C)
Line Track Underflow
FTERROVR (R/W1C)
Frame Track Overflow
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-30: EPPI_STAT Register Diagram

Table 18-64: EPPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	FLD	Current Field Received by EPPI. The `EPPI_STAT.FLD` bit indicates whether the current field being received by the PPI is field 1 (if clear) or field 2 (if set).	
		0	Field 1
		1	Field 2
14 (R/W1C)	ERRDET	Preamble Error Detected. The `EPPI_STAT.ERRDET` bit is useful only in ITU receive modes and indicates if an error has been detected in the status word of EAV or SAV sequences (if set) or not (if clear).	
		0	No Preamble Error Detected
		1	Preamble Error Detected
7 (R/W1C)	PXPERR	PxP Ready Error. The `EPPI_STAT.PXPERR` bit is valid only in the RX mode. This bit indicates whether the incoming PPI data overflows the PxP interface (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	

Table 18-64: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	ERRNCOR	Preamble Error Not Corrected. The `EPPI_STAT.ERRNCOR` bit is useful only in the ITU receive modes and indicates if an error in the status word of EAV or SAV sequences can not be cleared (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No uncorrected preamble error has occurred
		1	Preamble error detected but not corrected
5 (R/W1C)	FTERRUNDR	Frame Track Underflow. The `EPPI_STAT.FTERRUNDR` bit indicates whether a frame track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
4 (R/W1C)	FTERROVR	Frame Track Overflow. The `EPPI_STAT.FTERROVR` bit indicates whether a frame track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
3 (R/W1C)	LTERRUNDR	Line Track Underflow. The `EPPI_STAT.LTERRUNDR` bit indicates whether a line track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
2 (R/W1C)	LTERROVR	Line Track Overflow. The `EPPI_STAT.LTERROVR` bit indicates whether a line track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Table 18-64: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	YFIFOERR	Luma FIFO Error. For RX modes, the `EPPI_STAT.YFIFOERR` bit indicates whether the Luma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the Luma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
0 (R/W1C)	CFIFOERR	Chroma FIFO Error. For RX modes, the `EPPI_STAT.CFIFOERR` bit indicates whether the chroma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the chroma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Vertical Transfer Count Register

The `EPPI_VCNT` register holds the number of lines to read in or write out, after `EPPI_VDLY` number of lines from the start of frame. Any write to the `EPPI_FRAME` register modifies the `EPPI_VCNT` register. However, any write to the `EPPI_VCNT` register does not affect the `EPPI_FRAME` register value. So the `EPPI_VCNT` register should be programmed after the `EPPI_FRAME` register.

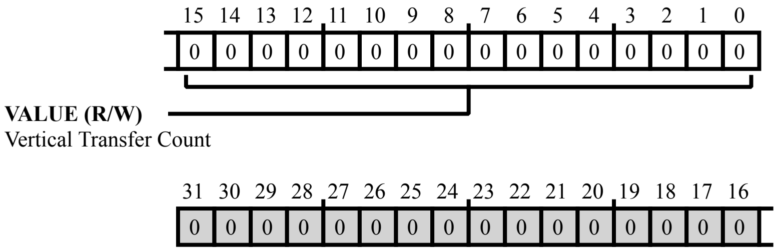

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Vertical Transfer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-31: EPPI_VCNT Register Diagram

Table 18-65: EPPI_VCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Transfer Count. The `EPPI_VCNT.VALUE` holds the number of lines to read in or write out, after `EPPI_VDLY` number of lines from the start of frame.

Vertical Delay Count Register

The `EPPI_VDLY` register contains the number of lines to wait after the start of a new frame before starting to read/transmit data.

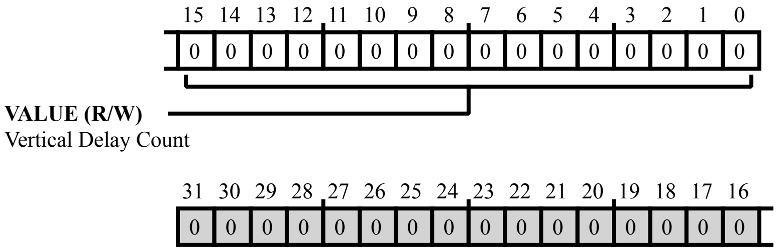

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Vertical Delay Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-32: EPPI_VDLY Register Diagram

Table 18-66: EPPI_VDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Delay Count. The `EPPI_VDLY.VALUE` holds the number of lines to wait after the start of a new frame before starting to read/transmit data.

19 Pulse-Width Modulator (PWM)

The pulse-width modulator (PWM) module is a flexible and programmable waveform generator. With minimal CPU intervention, the PWM peripheral can generate complex waveforms for:

- Motor control
- Pulse-coded modulation (PCM)
- Digital-to-analog conversion (DAC)
- Power switching
- Power conversion

The PWM module has four PWM pairs capable of three-phase PWM generation for source inverters for AC induction and DC brushless motors.

PWM Features

Each PWM generation unit features:

- 16-bit center-based PWM generation unit
- Programmable PWM pulse width
- Single or double update modes
- Programmable dead time and switching frequency
- Twos-complement implementation which permits smooth transition to full-on and full-off states
- Dedicated asynchronous PWM shutdown signal
- Debounce filter option on trip inputs that allow the system programmer to filter out short transients

Functional Description

The following sections provide details on the functionality of the PWM.

- Architectural Concepts

- Timer Units
- Channel Timing Control Unit
- Output Disable and Cross-Over Modes
- Sync Operation Modes

ADSP-SC58x PWM Register List

The Pulse-Width Modulator unit (PWM) includes multiple timers (providing period flexibility) and channels (providing mode, interrupt, and pulse shape flexibility), permitting a wide variety of PWM output options for motor control and other applications. A set of registers governs PWM operations. For more information on PWM functionality, see the PWM register descriptions.

Table 19-1: ADSP-SC58x PWM Register List

Name	Description
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register

Table 19-1: ADSP-SC58x PWM Register List (Continued)

Name	Description
`PWM_BL0_HP`	Channel B-Low Heightened-Precision Duty-0 Register
`PWM_BL1`	Channel B-Low Duty-1 Register
`PWM_BL1_HP`	Channel B-Low Heightened-Precision Duty-1 Register
`PWM_BL_DUTY0`	Channel B-Low Full Duty0 Register
`PWM_BL_DUTY1`	Channel B-Low Full Duty1 Register
`PWM_CCTL`	Channel C Control Register
`PWM_CH0`	Channel C-High Pulse Duty Register 0
`PWM_CH0_HP`	Channel C-High Pulse Heightened-Precision Duty Register 0
`PWM_CH1`	Channel C-High Pulse Duty Register 1
`PWM_CH1_HP`	Channel C-High Pulse Heightened-Precision Duty Register 1
`PWM_CHANCFG`	Channel Configuration Register
`PWM_CHA_DT`	Channel A Dead-time Register
`PWM_CHB_DT`	Channel B Dead-time Register
`PWM_CHC_DT`	Channel C Dead-time Register
`PWM_CHD_DT`	Channel D Dead-time Register
`PWM_CHOPCFG`	Chop Configuration Register
`PWM_CH_DUTY0`	Channel C-High Full Duty0 Register
`PWM_CH_DUTY1`	Channel C-High Full Duty1 Register
`PWM_CL0`	Channel C-Low Pulse Duty Register 0
`PWM_CL0_HP`	Channel C-Low Pulse Duty Register 1
`PWM_CL1`	Channel C-Low Duty-1 Register
`PWM_CL1_HP`	Channel C-Low Heightened-Precision Duty-1 Register
`PWM_CL_DUTY0`	Channel C-Low Full Duty0 Register
`PWM_CL_DUTY1`	Channel C-Low Full Duty1 Register
`PWM_CTL`	Control Register
`PWM_DCTL`	Channel D Control Register
`PWM_DH0`	Channel D-High Duty-0 Register
`PWM_DH0_HP`	Channel D-High Pulse Heightened-Precision Duty Register 0
`PWM_DH1`	Channel D-High Pulse Duty Register 1
`PWM_DH1_HP`	Channel D High Pulse Heightened-Precision Duty Register 1
`PWM_DH_DUTY0`	Channel D-High Full Duty0 Register

Table 19-1: ADSP-SC58x PWM Register List (Continued)

Name	Description
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register
PWM_DL1	Channel D-Low Pulse Duty Register 1
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register
PWM_ILAT	Interrupt Latch Register
PWM_IMSK	Interrupt Mask Register
PWM_STAT	Status Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_TRIPCFG	Trip Configuration Register

ADSP-SC58x PWM Interrupt List

Table 19-2: ADSP-SC58x PWM Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
30	PWM0_SYNC	PWM0 PWMTMR Grouped	Edge	
31	PWM0_TRIP	PWM0 Trip	Level	
32	PWM1_SYNC	PWM1 PWMTMR Grouped	Edge	
33	PWM1_TRIP	PWM1 Trip	Level	
34	PWM2_SYNC	PWM2 PWMTMR Grouped	Edge	

Table 19-2: ADSP-SC58x PWM Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
35	PWM2_TRIP	PWM2 Trip	Level	

ADSP-SC58x PWM Trigger List

Table 19-3: ADSP-SC58x PWM Trigger List Masters

Trigger ID	Name	Description	Sensitivity
21	`PWM0_SYNC`	PWM0 PWMTMR Grouped	Edge
22	`PWM1_SYNC`	PWM1 PWMTMR Grouped	Edge
23	`PWM2_SYNC`	PWM2 PWMTMR Grouped	Edge

Table 19-4: ADSP-SC58x PWM Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
8	`PWM0_TRIP_TRIG0`	PWM0 Trip Trigger Slave 0	Pulse
9	`PWM0_TRIP_TRIG1`	PWM0 Trip Trigger Slave 1	Pulse
10	`PWM0_TRIP_TRIG2`	PWM0 Trip Trigger Slave 2	Pulse
11	`PWM1_TRIP_TRIG0`	PWM1 Trip Trigger Slave 0	Pulse
12	`PWM1_TRIP_TRIG1`	PWM1 Trip Trigger Slave 1	Pulse
13	`PWM1_TRIP_TRIG2`	PWM1 Trip Trigger Slave 2	Pulse
14	`PWM2_TRIP_TRIG0`	PWM2 Trip Trigger Slave 0	Pulse
15	`PWM2_TRIP_TRIG1`	PWM2 Trip Trigger Slave 1	Pulse
16	`PWM2_TRIP_TRIG2`	PWM2 Trip Trigger Slave 2	Pulse

PWM Definitions

The following definitions are helpful when using the PWM module.

Chopping

Used to simplify the design of isolated gate drive circuits for PWM inverters. If using a transformer coupled power device gate drive amplifier, then the active PWM signal must be chopped at a high frequency.

Dead-Time

A short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits turning off a power switch (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the dc link capacitor of a typical voltage source inverter.

Duty Cycle

The proportion of on time to the regular interval or period of time (expressed in percent, 100% being fully on). A low duty cycle corresponds to low power, because the power is off for most of the time.

Switching Frequency

The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

Architectural Concepts

A clock, whose period is t_{SCLK0_0}, drives the PWM controller. The PWM generator produces four pairs (four high-side and four low-side) of PWM signals on the eight PWM output pins. Each high and low pair signal constitutes a channel. For example, the `PWM_AL` and `PWM_AH` signals make up channel A, and the `PWM_BL` and `PWM_BH` signals make up channel B, and so on.

Each pair of channel outputs references either a main timer or an independent timer. These timers operate on a switching frequency determined by the `PWM_TM0` through `PWM_TM4` registers. There are two duty registers for every PWM output. The registers enable generation of symmetrical or asymmetrical waveforms. The waveforms produce lower harmonic distortion in three-phase PWM inverters, with minimal CPU intervention.

Block Diagram

The ***PWM Block Diagram*** figure shows a block diagram that represents the main functional blocks of the PWM controller.

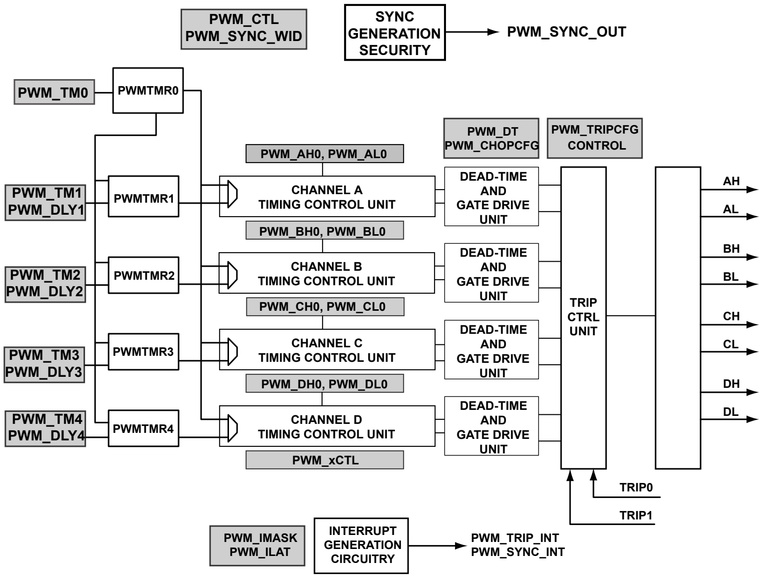

PWM_CTL
PWM_SYNC_WID
SYNC GENERATION SECURITY
PWM_SYNC_OUT
PWM_TM0
PWMTMR0
PWM_DT
PWM_CHOPCFG
PWM_TRIPCFG CONTROL
PWM_AH0, PWM_AL0
PWM_TM1
PWM_DLY1
PWMTMR1
CHANNEL A TIMING CONTROL UNIT
DEAD-TIME AND GATE DRIVE UNIT
PWM_BH0, PWM_BL0
PWM_TM2
PWM_DLY2
PWMTMR2
CHANNEL B TIMING CONTROL UNIT
DEAD-TIME AND GATE DRIVE UNIT
PWM_CH0, PWM_CL0
PWM_TM3
PWM_DLY3
PWMTMR3
CHANNEL C TIMING CONTROL UNIT
DEAD-TIME AND GATE DRIVE UNIT
PWM_DH0, PWM_DL0
PWM_TM4
PWM_DLY4
PWMTMR4
CHANNEL D TIMING CONTROL UNIT
DEAD-TIME AND GATE DRIVE UNIT
PWM_xCTL
TRIP CTRL UNIT
AH
AL
BH
BL
CH
CL
DH
DL
TRIP0
TRIP1
PWM_IMASK
PWM_ILAT
INTERRUPT GENERATION CIRCUITRY
PWM_TRIP_INT
PWM_SYNC_INT

Figure 19-1: PWM Block Diagram

The following list describes the primary blocks.

- Each pair of PWM signals references either the main timer or the independent timer.
- PWMTMR0 is the main timer and can trigger the delayed start of the other timers.
- Timing control units, one for each channel, together form the core of the PWM. The unit generates the required complex waveforms on the high-side and low-side outputs for the respective channel.
- Dead-time insertion occurs after the ideal PWM output pair is generated.
- The gate drive unit generates the high-frequency chopping signal and then mixes it with the requisite PWM output signals.
- The PWM shutdown and interrupt controller manage the various PWM shutdown modes for the timing unit and generate the requisite interrupt signals.
- The PWM sync pulse control unit generates the internal `PWM_SYNC` pulse and also controls whether the external `PWM_SYNC` input pulse is used.

Timer Units

Five timers make up the time base for the PWM module. The main timer, PWMTMR0 operates at a switching frequency determined by the period register `PWM_TM0`. The four remaining timers (PWMTMR1 through PWMTMR4) can operate at independent switching frequencies determined by their respective registers.

The respective time registers (`PWM_TM1` through `PWM_TM4`) can be programmed to work at a multiple of the main timer frequency. In this case, the `PWM_DLYA` through `PWM_DLYD` registers control the lead-lag phase of a given timer based on the main timer PWMTMR0.

NOTE: The delayed operation of a timer requires one of the following:

- The register value of the timer must be equal to the `PWM_TM0` register value.
- The `PWM_TM0` value must be an integer multiple of each register of the timer. Non-integer multiples are not allowed.

PWM Timer Period (PWM_TM) Registers

The 16-bit read/write PWM period registers (`PWM_TM0` through `PWM_TM4`) control the PWM switching frequency. The fundamental timing unit of the PWM controller is t_{SCLK0_0}. Therefore, the time increment (t_{SCLK0_0}) is 10 ns for a 100-MHz system clock (SCLK0_0) frequency, f_{SCLK0_0}. The value written to the register of a timer is effectively the number of t_{SCLK0_0} clock increments in one half of a PWM period. The following equation describes the required timer register value as a function of the desired PWM switching frequency (f_{PWM}):

$$PWM\_TM = f_{SCLK0\_0}/2 \times f_{PWM}$$

Therefore, the PWM switching period (T_s) is:

$$T_s = 2 \times PWM\_TM \times t_{SCLK0\_0}$$

For example, for an f_{SCLK0_0} of 100 MHz and a desired PWM switching frequency (f_{PWM}) of 10 kHz ($T_s = 100$ ms), the correct value to load into the timer register is:

$$PWM\_TM = 100 \times 10^6 \div 2 \times 10 \times 10^3 = 5000$$

The largest value that can be written to the 16-bit timer register is 0xFFFF = 65,535. For an f_{SCLK0_0} of 100 MHz, this value corresponds to a minimum PWM switching frequency of:

$$f_{PWM(min)} = 100 \times 10^6 \div 2 \text{ x } 65535 = 762 \text{ Hz}$$

NOTE: Timer register values of 0 and 1 are not defined. Do not use these values when the PWM outputs or PWM is enabled.

Timer Unit Operation

The PWM timers are up-down counters, and they operate on the peripheral clock with a period of t_{CK}. The period of the PWM timer is divided into two halves. In the first half, the timer roughly counts down from PWM_TMx/2 to –PWM_TMx/2. During this half, the `PWM_STAT.TMR0PHASE` through `PWM_STAT.TMR4PHASE` bits are held at 0. In the second half of the period, the timer roughly counts up from –PWM_TMx/2 to PWM_TMx/2. The `PWM_STAT.TMR0PHASE` through `PWM_STAT.TMR4PHASE` bits indicates a 1 during this half.

The actual partition of the periods varies slightly between odd and even values of the half-period, in the `PWM_TM[n]` registers.

If a timer register value is odd, for example 11, then that timer loads +5 at the beginning of the period. The timer counts down from +5 to –5 in the first half, reloads –5 at the midpoint and counts up from –5 to +5 in the second half. The reload values at the period and mid-period boundaries are the same as the previous count. The timer counts 2 × 11 half-periods = 22 total counts in the entire period as shown in the ***Operation of Timer for Odd Value of PWM_TM*** figure.

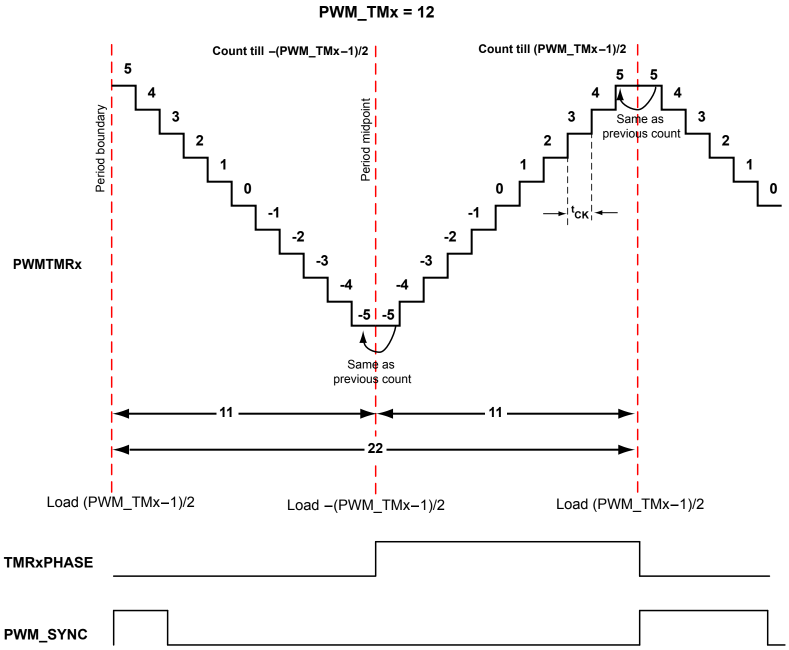

PWM_TMx = 12
Count till –(PWM_TMx–1)/2
Count till (PWM_TMx–1)/2
Period boundary
Period midpoint
PWMTMRx
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5
5
4
3
2
1
0
Same as previous count
Same as previous count
tCK
11
11
22
Load (PWM_TMx–1)/2
Load –(PWM_TMx–1)/2
Load (PWM_TMx–1)/2
TMRxPHASE
PWM_SYNC

Figure 19-2: Operation of Timer for Odd Value of PWM_TM

When the timer register value is even, for example 12, then that timer loads +5 at the beginning of the period. The timer counts from +5 to –6 in the first half, reloads –5 at the midpoint and counts up from –5 to +6 in the second half. The reload values at the period and mid-period boundaries are different from the previous count. It counts 2 × 12 half-periods = 24 total counts in the entire period as shown in the ***Operation of Timer for Even Value of PWM_TM*** figure.

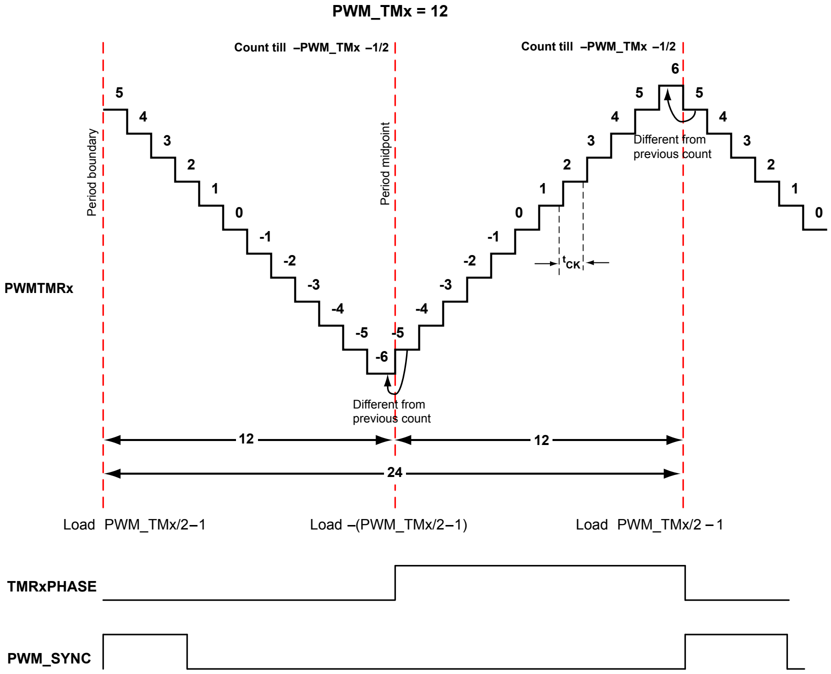

PWM_TMx = 12
Count till –PWM_TMx –1/2
Count till –PWM_TMx –1/2
Period boundary
Period midpoint
PWMTMRx
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
5
4
3
2
1
0
Different from previous count
Different from previous count
tCK
12
12
24
Load PWM_TMx/2–1
Load –(PWM_TMx/2–1)
Load PWM_TMx/2 – 1
TMRxPHASE
PWM_SYNC

Figure 19-3: Operation of Timer for Even Value of PWM_TM

NOTE: In the operation discussed in this section, double-buffering of all channel registers and the timer registers takes place at the period boundary of the respective timers.

Phase Offset Control

The PWM timers (PWMTMR1 through PWMTMR4) can operate with a programmable phase lag relative to the main timer, PWMTMR0. To implement phase offset for a channel, use the counter-registers for channel delay (`PWM_DLYA` - `PWM_DLYD`) with the PWMTMR0 and set the `PWM_CTL.DLYAEN` bit to l.

Phase offset works as follows.

1. If phase lag is used for channel A (and channel A uses PWMTMR1 to generate a duty cycle), when PWMTMR0 reaches its period boundary, it triggers the `PWM_DLYA` register. The register counts out the number SCLK0_0 cycles that are equal to the value programmed in the `PWM_DLYA` register.
2. At the end of this count, the `PWM_DLYA` register sends out a trigger to PWMTMR1. It receives a synchronization pulse in every period of PWMTMR0 at a point delayed from its period boundary by the value in the `PWM_DLYA` register.

For more information on how channels can reference different timers for their outputs, see Channel Timing Control Unit.

NOTE: Satisfy the following conditions when using this feature on timer y for channel Y relative to PWMTMRx.

- Program the `PWM_DLY[n]` register to a value less than 2 × `PWM_TM[n]`.
- `PWM_TM0` = N × `PWM_TM[n]`, where N is an integer.

The function of `PWM_TM[n]` (PWMTMR1 in the example) differs in cases where `PWM_TM0` = `PWM_TM1` (Case 1) to cases where `PWM_TM0` = N × `PWM_TM1` (Case 2). The following examples describe both cases.

Case 1: PWM_TM0 = PWM_TMy

When `PWM_TM0` = `PWM_TMy`, PWMTMRy restarts its period after receiving the synchronization pulse from the channel delay register (`PWM_DLY[n]`). If the trigger from the `PWM_DLY[n]` register is late, PWMTMRy holds its count until the trigger occurs. If the trigger is a bit early, PWMTMRy reloads without regard to whether it has completed its current period. As a result, PWMTMRy resyncs with PWMTMR0 with the phase lag programmed in the `PWM_DLYA` register in every one of its periods.

In this case, the expiration of the delay registers (`PWM_DLY[n]`) is the period boundary of PWMTMRy. Now, all the double buffered registers related to the given channel update (except the delay registers which are double buffered at the period boundary of PWMTMR0).

The ***Phase Offset Control Using DELAY*** figure shows an example where:

- `PWM_TM0`, `PWM_TM1`, and `PWM_TM2` are programmed with the same value.
- `PWM_DLYA` and `PWM_DLYB` are programmed with values DELAY1 and DELAY2 respectively, such that DELAY2 > DELAY1.

- The outputs of Channel A are referenced to PWMTMR1. The outputs of channel B are referenced to PWMTMR2.

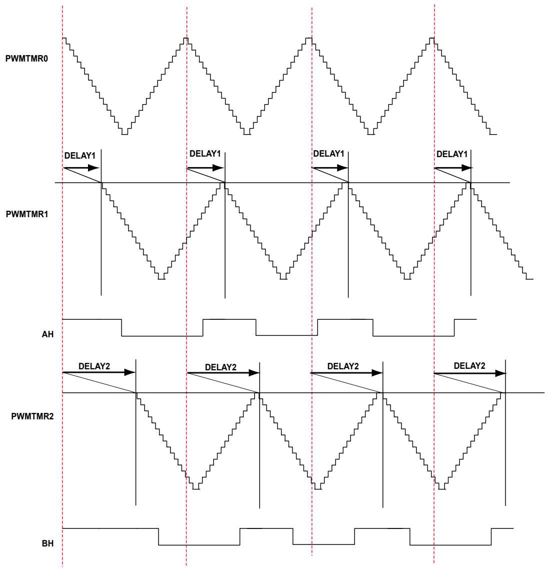


Figure 19-4: Phase Offset Control Using DELAY

The delay registers are double buffered and the new value of DELAY reloads at the period boundary of PWMTMR0. The two options exist when the new value is different from the older one. The behavior of PWMTMRy in both these cases is discussed. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows the behavior in the two cases. It is assumed that channel B references its outputs to PWMTMR0 and channel A references its outputs to PWMTMR1.

1. The new delay value is higher than the previous value. Here the corresponding PWMTMRy allows more than one time period between consecutive triggers from the channel delay (`PWM_DLYA` - `PWM_DLYD`) registers. In this case, after reaching its period boundary, PWMTMRy holds its count at the period boundary and waits for the trigger from the channel delay register. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case A functionality.
2. The next delay value programmed is smaller than the previous value. Here, the corresponding PWMTMRy allows only less than one time period between consecutive triggers from the channel delay register. Though the trigger comes earlier in this case, before PWMTMRy has counted out one full period, it reloads and starts its period again. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case B functionality.

Therefore, PWMTMR1 waits and obeys a synchronization pulse from the `PWM_DLYA` register in every one of its periods.

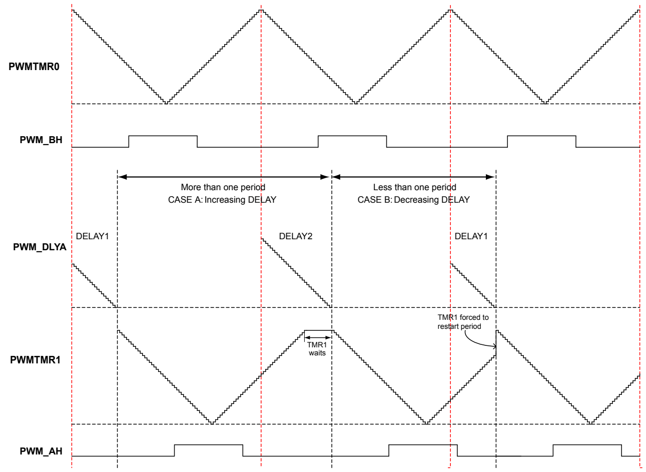

PWMTMR0
PWM_BH
More than one period
CASE A: Increasing DELAY
Less than one period
CASE B: Decreasing DELAY
DELAY1
DELAY2
DELAY1
PWM_DLYA
TMR1 forced to restart period
TMR1 waits
PWMTMR1
PWM_AH

Figure 19-5: Impact of New DELAY Value on Timer Count for Equal Timer Periods

Case 2: PWM_TM0 = N x PWM_TMy

In this case, within a single period of PWMTMR0 a program can fit multiple periods (N) of PWMTMRy. Additionally, the channel delay register is triggered only once every N periods of PWMTMRy.

The operation is as follows: Every Nth period of PWMTMRy, PWMTMRy expects a synchronization pulse from the `PWM_DLY[n]` register. When this register counts out that period and the trigger has not yet arrived, PWMTMRy waits at the end of the period for the trigger. PWMTMRy starts counting down once the trigger arrives. If the trigger comes earlier, PWMTMRy restarts immediately without waiting to complete the period count.

In the intervening periods, PWMTMRy operates independently. As the period ends, PWMTMRy reloads and starts the next period without intervention from the channel delay register.

The ***Impact of DELAY Value Change for the Multiple Timer Periods*** shows an example with N = 2. PWMTMRy syncs up with PWMTMR0 every second period, and is free running across every odd period boundary.

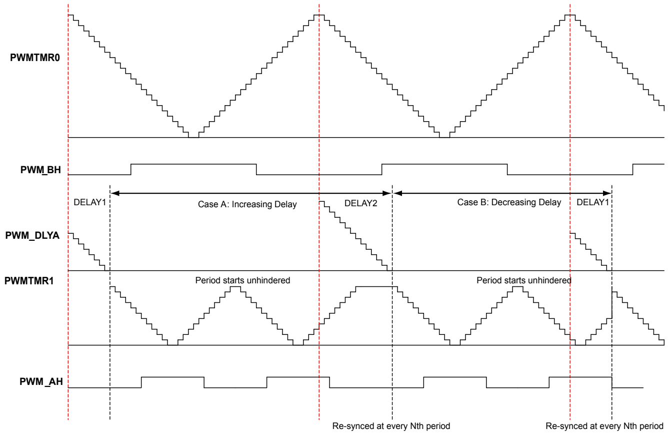

PWMTMR0
PWM_BH
DELAY1
Case A: Increasing Delay
DELAY2
Case B: Decreasing Delay
DELAY1
PWM_DLYA
PWMTMR1
Period starts unhindered
Period starts unhindered
PWM_AH
Re-synced at every Nth period
Re-synced at every Nth period

Figure 19-6: Impact of DELAY Value Change for Multiple Timer Periods

Channel Timing Control Unit

The channel timing control unit is the core of the PWM. There are four separate channels, each channel controlling a pair of output signals – the high-side output and the low-side output.

Channel Control

The `PWM_CHANCFG` register controls the static configuration of all the channels and is initialized once before the beginning of a PWM operation.

NOTE: The `PWM_CHANCFG` register is not double buffered. Do not change the contents of the register once the PWM is enabled.

Each channel works with a reference timer base. The time base can be either the main timer PWMTMR0 or the appropriate PWMTMRx. Configure the time base with the `PWM_CHANCFG.REFTMRA` bit field as follows.

- Channel A works with PWMTMR0 or PWMTMR1
- Channel B works with PWMTMR0 or PWMTMR2
- Channel C works with PWMTMR0 or PWMTMR3
- Channel D works with PWMTMR0 or PWMTMR4

The double-buffered channel control registers (`PWM_ACTL` through `PWM_DCTL`) contain bits that control the dynamic pulse behavior of the channel outputs. These registers have bits that enable or disable outputs and select the pulse position of outputs (explained in the following section).

Pulse Positioning and Duty Cycle Registers

The PWM uses the PULSEMODEHI and PULSEMODELO bit fields of the `PWM_[n]CTL` registers to define the region within the timer period where the output pulses are positioned.

- When the `PWM_CHANCFG.MODELSC` bit is 0, the PWM uses the PULSEMODEHI field to specify the pulse positioning for both the high-side and low-side outputs of the channel.
- When the bit is 1, the PWM uses `PWM_ACTL.PULSEMODELO` to define the pulse positioning for the low-side output of the channel. It uses the `PWM_ACTL.PULSEMODEHI` to define the pulse positioning for the high-side output of the channel.

Each channel output has two duty-cycle registers: `PWM_AH0` and `PWM_AH1` for the high-side output, and `PWM_AL0` and `PWM_AL1` for the low-side output. These registers determine the width of the output pulses. When the `PWM_CHANCFG.MODELSC` bit is 0, the high-side duty-cycle registers are used to determine the width of the output pulse for the low side. The duty cycle range that can be programmed into these registers is between –`PWM_TM[n]`/2 and +`PWM_TM[n]`/2 , when ignoring dead time.

When including dead time for channel A, for PULSEMODEs 00 and 01, the programmed duty cycle is modified. The range is limited between the values [–`PWM_TM[n]`/2 + `PWM_CHA_DT`] and [+`PWM_TM[n]`/2 + `PWM_CHA_DT`] considering the high-side output. For PULSEMODEs 10 and 11, the high-side duty cycle registers range is limited between values [`PWM_TM[n]`/2 + `PWM_CHA_DT`] and [–`PWM_TM[n]`/2 – `PWM_CHA_DT`].

The following section explains dead time in detail.

- Switching Dead Time (PWM_DT) Register

NOTE: Values programmed into these registers that fall outside these limits result in over or under modulation.

Duty Cycle and Pulse Positioning Control

The `PWM_ACTL.PULSEMODEHI` and `PWM_ACTL.PULSEMODELO` fields control how the duty cycle registers modify the waveform of the high and low-side outputs. (The `PWM_ACTL.PULSEMODEHI` and `PWM_ACTL.PULSEMODELO` fields are referred to as *pulse mode* in the subsequent discussion.)

- Pulse mode = 00 – Produce a symmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses only one of the duty cycle registers for an output. For example, for the AH output, PWM uses only the `PWM_AH0` register. In this mode, the values in the duty cycle registers are scaled such that a value of 0 produces a 50% duty cycle.
- Pulse mode = 01 – Produce an asymmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses both duty cycle registers. For example, for the PWM_AH output, PWM uses the `PWM_AH0` and `PWM_AH1` registers. In this mode, if the `PWM_AH1` register is programmed with the same value as the `PWM_AH0` register, the output is identical to the output when pulse mode =00.
- Pulse mode = 10 or 11 – Produce pulse waveforms either on the first half or the second half of the PWM period respectively. PWM uses both `PWM_AH0` and `PWM_AH1` registers.

  Pulse mode = 10. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition `PWM_AL0`> `PWM_AL1`.

  In pulse mode = 11. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition `PWM_AL0`< `PWM_AL1`.

The ***Pulse Positioning Modes*** figure shows the pulse positioning modes as previously described for PWM_AH. In the figure, DUTY0 is the value in the `PWM_AL0` register and DUTY1 is the value in the `PWM_AH1` register. The step signal, count, indicates the output of the timer for channel A. In the example, the signal is configured as active high and dead time is zero.

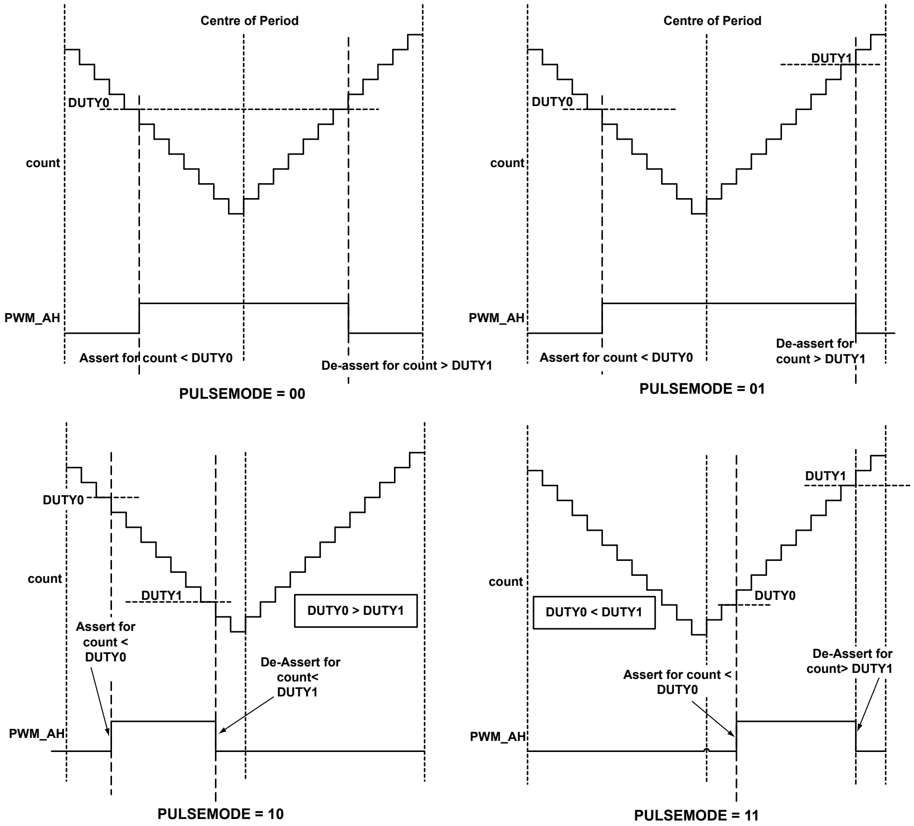

Centre of Period
DUTY0
count
PWM_AH
Assert for count < DUTY0
De-assert for count > DUTY1
PULSEMODE = 00
Centre of Period
DUTY1
DUTY0
count
PWM_AH
Assert for count < DUTY0
De-assert for count > DUTY1
PULSEMODE = 01
DUTY0
count
DUTY1
DUTY0 > DUTY1
Assert for count < DUTY0
De-Assert for count< DUTY1
PWM_AH
PULSEMODE = 10
DUTY1
count
DUTY0 < DUTY1
DUTY0
De-Assert for count> DUTY1
Assert for count < DUTY0
PWM_AH
PULSEMODE = 11

Figure 19-7: Pulse Positioning Modes

Channel Low Side Output Dependent Operation Mode and Dead Time

The low-side output waveform can be programmed to depend on the waveform of the high-side output or to be independent. The PWM uses the `PWM_CHANCFG.MODELSC` bit to control this functionality.

For example, channel A produces the high-side output PWM_AH and the low-side output PWM_AL. When the `PWM_CHANCFG.MODELSC` bit =0, the low-side output is also generated using the high-side duty-cycle registers for pulse width, the `PWM_ACTL.PULSEMODEHI` bits for pulse positioning and the `PWM_CHANCFG.POLAH` bit for polarity. If the `PWM_CH[x]_DT` register is 0, the low-side output is an inverted version of the high-side output.

When the `PWM_CH[x]_DT` register is programmed with a non-zero value, both the high-side and low-side outputs are scaled symmetrically about the points of transition in the zero dead time case. The PWM scales the output by the value programmed in the `PWM_CH[x]_DT` register.

The *Channel Outputs in Dependent Mode for Pulse Mode* figures show the high and low-side outputs for the case with zero and non-zero dead time for `PWM_ACTL.PULSEMODEHI` =00 and 01. DUTY0 is the value programmed into the `PWM_AH0` register. DUTY1 is the value programmed into the `PWM_AH1` register. The `PWM_CHANCFG.POLAH` bit =1, indicates that both signals are active high. The `PWM_CHA_DT` register holds the value DT.

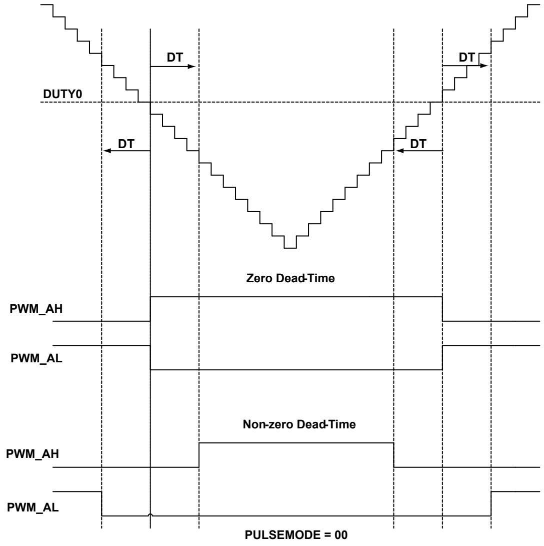

DUTY0
DT
DT
DT
DT
Zero Dead-Time
PWM_AH
PWM_AL
Non-zero Dead-Time
PWM_AH
PWM_AL
PULSEMODE = 00

Figure 19-8: Channel Outputs in Dependent Mode for Pulse Mode = 00

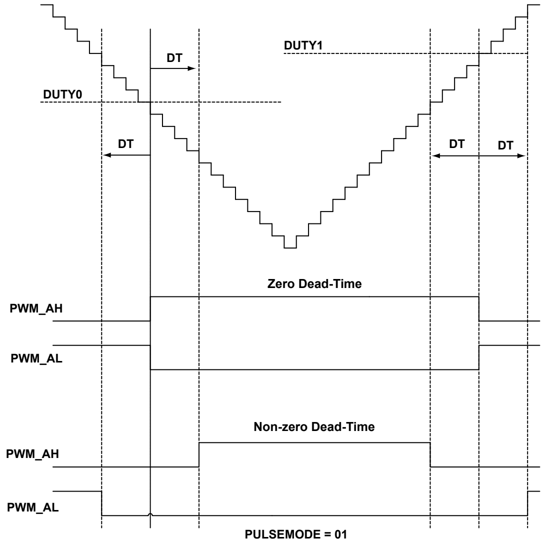

DUTY1
DT
DUTY0
DT
DT
DT
Zero Dead-Time
PWM_AH
PWM_AL
Non-zero Dead-Time
PWM_AH
PWM_AL
PULSEMODE = 01

Figure 19-9: Channel Outputs in Dependent Mode for Pulse Mode = 01

The following pair of figures shows the high and low-side outputs for the case with zero and non-zero dead-time for `PWM_ACTL.PULSEMODEHI` =10 and 11. In the figures, DUTY0 is the value programmed into `PWM_AH0` register and DUTY1 is the value programmed into the `PWM_AH1` register. `PWM_CHANCFG.POLAH` is 1 indicating that both signals are active high. The channel dead-time registers hold the value DT.

> **NOTE:** Using dead time, the guidelines for programming the duty-cycle registers in pulse modes 10 and 11 given in Duty Cycle and Pulse Positioning Control are modified as follows:
>
> Pulse mode 10: PWM_xH0 – DT > PWM_xH1 + DT
>
> Pulse mode 11: PWM_xH0 + DT < PWM_xH1 – DT

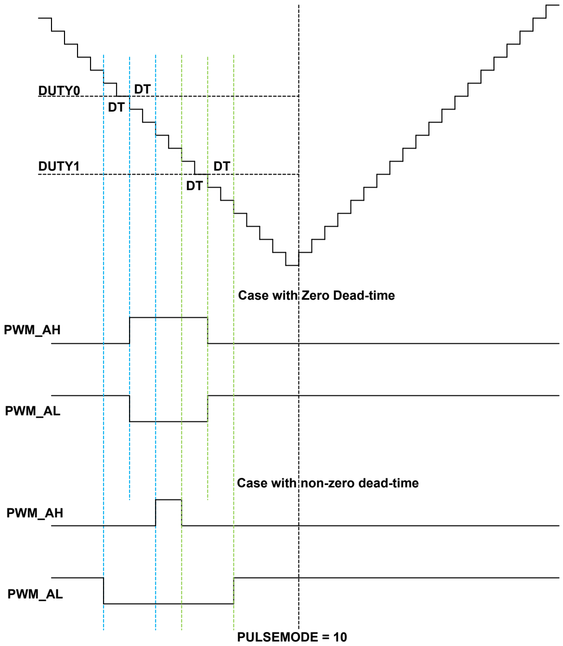

DUTY0
DT
DT
DUTY1
DT
DT
Case with Zero Dead-time
PWM_AH
PWM_AL
Case with non-zero dead-time
PWM_AH
PWM_AL
PULSEMODE = 10

Figure 19-10: Channel Outputs in Dependent Mode for Pulse Mode = 10

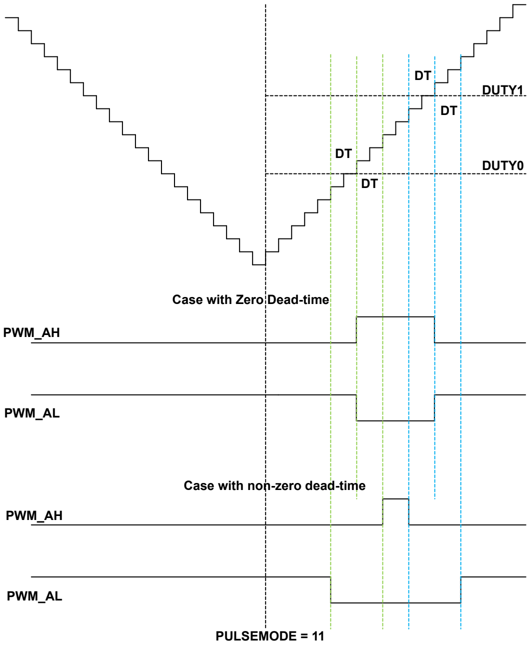

DT
DUTY1
DT
DT
DUTY0
DT
Case with Zero Dead-time
PWM_AH
PWM_AL
Case with non-zero dead-time
PWM_AH
PWM_AL
PULSEMODE = 11

Figure 19-11: Channel Outputs in Dependent Mode for Pulse Mode = 11

Channel High Side and Low Side Outputs, Independent Operation Mode

Independent control of the `PWM_AH0` and `PWM_AL0` channel outputs is possible by setting the `PWM_CHANCFG.MODELSA` bit to 1. In this case, the PWM module:

- Generates PWM_AH using the `PWM_AH0` register
- Uses the `PWM_AH1` register to configure pulse width

- Uses the `PWM_ACTL.PULSEMODEHI` bit to configure pulse position
- Uses the `PWM_CHANCFG.POLAH` bit to configure polarity
- Generates PWM_AL using `PWM_AL0`
- Uses the `PWM_AL1` register to configure pulse width
- Uses the `PWM_ACTL.PULSEMODELO` bit to configure pulse position
- Uses the `PWM_CHANCFG.POLAL` bit to configure polarity

NOTE: In independent mode, the dead-time insertion is not applicable. The hardware forces dead time to zero.

The ***PWM_AH and PWM_AL in Independent Operation Mode*** figure shows an example of the independent mode of operation where PWM_AH and PWM_AL work from different register bits.

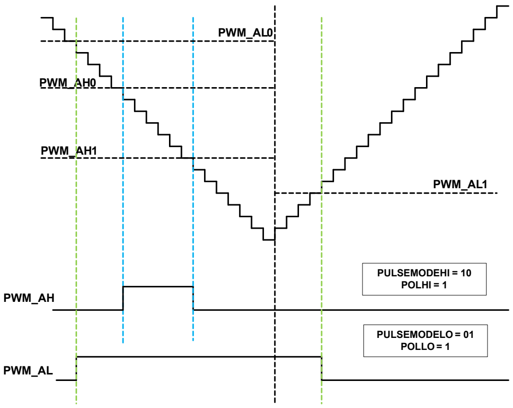


Figure 19-12: PWM_AH and PWM_AL in Independent Operation Mode

PWM_AH and PWM_AL can be positioned in the timer period with a given phase difference between them. Program the `PWM_ACTL.PULSEMODEHI` and `PWM_ACTL.PULSEMODELO` bits to different values to achieve this positioning as shown in the ***Channel Outputs Controlled Independently*** figure.

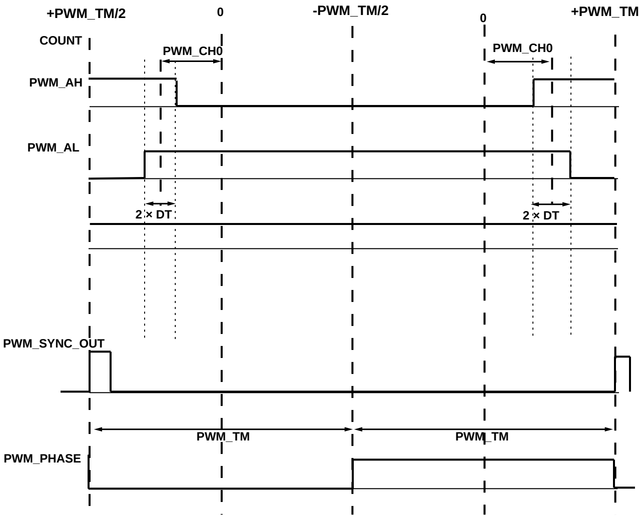


Figure 19-13: Channel Outputs Controlled Independently

Switched Reluctance Motors Application

In typical power converter configurations for switched or variable reluctance motors, motor winding is connected between the two power switches of a given inverter leg. To allow for a complete circuit in the motor winding, turn on both switches at the same time.

PWM uses switched reluctance motors in the following configurations: hard chop, alternate chop, soft chop—bottom on, and soft chop— top on.

The ***Four SR Mode Types, Active High PWM Output Signals*** figure shows the four SR mode types as active high PWM output signals.

Hard chop mode contains independently programmed rising edges of the high and low signals of a channel in the same PWM half cycle. Both signals contain independently programmed falling edges in the next PWM half cycle. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to same values.

Alternate chop mode is similar to normal PWM operation except that the PWM channel high and low signal edges are always opposite and are independently programmed. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to opposite values. The low-side invert is the only difference between hard chop mode and alternate chop mode.

Soft chop - bottom on uses a 100% duty on the low side of the channel. Soft chop - top on uses a 100% duty on the high side of the channel. Similar to hard chop mode, PWM uses the `PWM_AH0` duty register for the high channel and the `PWM_AL0` duty register for the low channel.

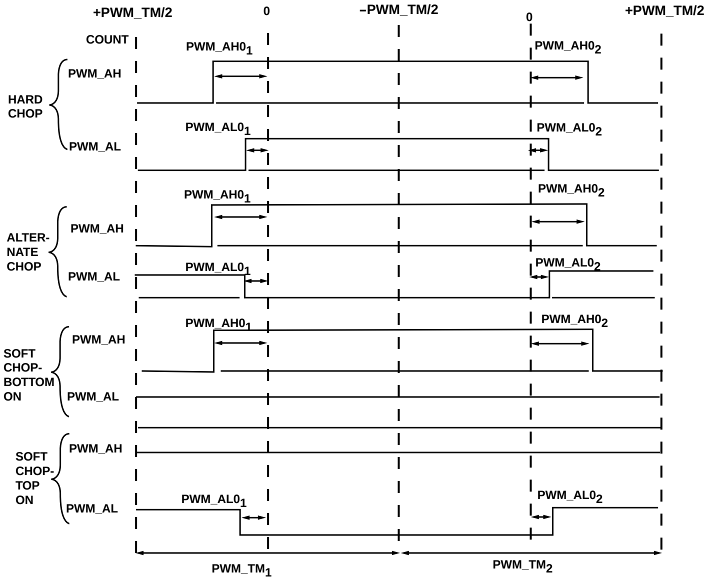

+PWM_TM/2
0
–PWM_TM/2
0
+PWM_TM/2
COUNT
PWM_AH0$_1$
PWM_AH0$_2$
HARD CHOP
PWM_AH
PWM_AL
PWM_AL0$_1$
PWM_AL0$_2$
ALTER-NATE CHOP
PWM_AH
PWM_AL
PWM_AH0$_1$
PWM_AH0$_2$
PWM_AL0$_1$
PWM_AL0$_2$
SOFT CHOP-BOTTOM ON
PWM_AH
PWM_AL
PWM_AH0$_1$
PWM_AH0$_2$
SOFT CHOP-TOP ON
PWM_AH
PWM_AL
PWM_AL0$_1$
PWM_AL0$_2$
PWM_TM$_1$
PWM_TM$_2$

Figure 19-14: Four SR Mode Types, Active High PWM Output Signals

Switching Dead Time (PWM_DT) Register

The second important parameter that must be set up in the initial configuration of the PWM controller is the switching dead time. Dead time is a short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits turning off a power switch (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the DC link capacitor of a typical voltage source inverter.

The 10-bit, read/write channel A through channel D dead-time registers (`PWM_CHA_DT` through `PWM_CHD_DT`) control the dead time for channel-x. If the value carried by any dead-time register is PWMDT, the dead time, Td, for that channel is:

$$T_d = \texttt{PWM\_CH[x]\_DT} \times 2 \times t_{SCLK0\_0}$$

Therefore, a dead-time value of 0x00A introduces a 200-ns delay (for an SCLK0_0 of 100 MHz). The delay occurs between turning off any PWM signal (for example, AH) and then turning on its complementary signal (for example, AL). The length of dead time can be programmed in increments of $2 \times t_{SCLK0\_0}$ (or 20 ns for an SCLK0_0 of 100 MHz). The channel A through channel D dead-time registers have a maximum value of 0x3FF (1023 decimal) and correspond to a maximum programmed dead time of:

$T_{d(max)} = 1023 \times 2 \times t_{SCLK0\_0} = 1023 \times 2 \times 10 \times 10^{-9} = 20.5$ µs for an f_{SCLK0_0} rate of 100 MHz.

Write 0 to the `PWM_CHA_DT` through `PWM_CHD_DT` registers to program the dead time.

Duty Cycle with Dead Time Control: Calculations for PULSEMODE 00

The duty cycle registers are scaled so that a value of 0 represents a 50% PWM duty cycle. The switching signals produced are also adjusted to incorporate the programmed dead-time value using the channel dead-time registers

(`PWM_CHA_DT` through `PWM_CHD_DT`). The unit in this case produces active low signals so that a low level corresponds to a command to turn-on the associated power device.

The ***Dead Time Between Outputs in Dependent Mode*** figure shows a typical pair of PWM outputs, PWM_AH and PWM_AL. The time values in the figure indicate the integer value in the associated register and can be converted to time by multiplying by the fundamental time increment, t_{CK}. In the example, channel A is working from `PWM_TM0`.

In the example, the pulse mode is set to 00 so that the switching patterns are perfectly symmetrical about the midpoint of the switching period. The dead time is incorporated by moving the switching instants of both PWM signals away from the instant set by the `PWM_AH0` register. Both switching edges are moved by an equal amount (DT × t_{CK}) to preserve the symmetrical output patterns. Also shown is the `PWM_SYNC` output pulse whose rising edge denotes the beginning of the switching period, and the `PWM_STAT.TMR0PHASE` bit.

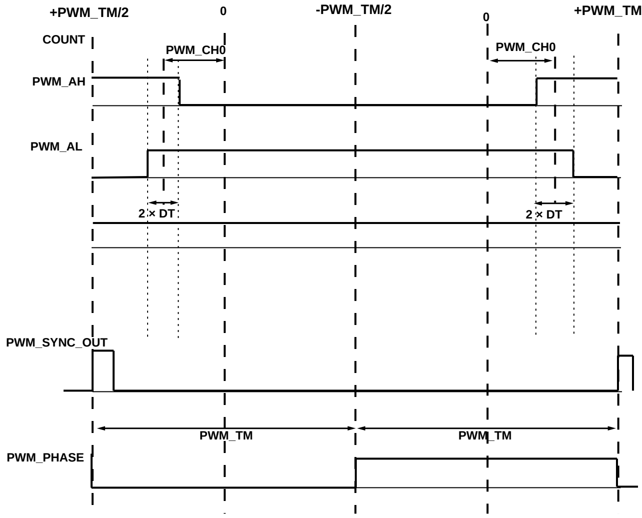


Figure 19-15: Dead Time Between Outputs in Dependent Mode

The PWM timing unit produces the resulting on-times (active low) of the PWM signals over the full PWM period (two half-periods). The figure illustrates this timing. The timing can be written per the following equation.

T_{AH} = (`PWM_TM0` + 2 × (`PWM_AH0` – `PWM_CH[x]_DT`)) × $t_{CK;}$

Range of T_{AH} is [0:2 × `PWM_TM0` × t_{CK}]

T_{AL} = (`PWM_TM0` – 2 × (`PWM_AH0` + `PWM_CH[x]_DT`)) × $t_{CK;}$

Range of T_{AL} is [0:2 × `PWM_TM0` × t_{CK}]

$$d_{AH} = \frac{T_{AH}}{Ts} = \frac{1}{2} + \frac{PWM\_AH0 - DT}{PWM\_TM}$$

$$d_{AL} = \frac{T_{AL}}{Ts} = \frac{1}{2} - \frac{PWM\_AH0 + DT}{PWM\_TM0}$$

The negative values of T_{AH} and T_{AL} are not permitted and the minimum permissible value is zero, corresponding to a 0% duty cycle. In a similar fashion, the maximum value is T_s, the PWM switching period, corresponding to a 100% duty cycle. Calculation of duty for other pulse modes can be similarly executed.

Special Consideration for PWM Operation in Over-Modulation

The PWM timing unit can produce PWM signals with variable duty cycle values at the PWM output pins. In pulse modes 00 and 01, at the extremities of the modulation process, duty cycles of 0% and 100% occur. In pulse modes 01 and 10, at the extremities of the modulation process, duty cycles of 0% and 50% occur. The modulation is called *full off* when the lower extremity of modulation is set for any PWM timer period for the corresponding channel. The modulation is called *full on* when the higher extremity of modulation is set for any PWM timer period for the corresponding channel. In between, for other duty cycle values, the operation is termed *normal modulation*.

Full On Modulation

In pulse modes 00 and 01, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full on modulation are:

- PWM_xH0 DT > PWM_TMy/2 for pulse mode 00
- PWM_xH1 DT > PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the first half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 DT > PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < PWM_TMy/2 for pulse mode 10

In pulse mode 11, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the second half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 + DT < PWM_TMy/2 for pulse mode 11
- PWM_xH1 DT > PWM_TMy/2 for pulse mode 11

Full Off Modulation

In pulse modes 00 and 01, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full off modulation are:

- PWM_xH0 DT < PWM_TMy/2 for pulse mode 00
- PWM_xH1 DT < PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the first half period of the PWM timer that the channel is referencing. In the second half-period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 DT < PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < PWM_xH0 DT for pulse mode 10

In pulse mode 11, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the second half period of the PWM timer that the channel is referencing. In the first half of the period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 + DT > PWM_TMy/2 for pulse mode 11
- PWM_xH1 DT > PWM_xH0 + DT for pulse mode 11

Normal Modulation

All other cases of modulation fall under this category.

Emergency Dead-Time Delays

Sometimes, during modulation transition, it is necessary to insert more emergency dead-time delays to prevent potential shoot through conditions in the inverter. (For example, when the PWM transitions into or out of full on or full off modulation.) Disabling and enabling usage (related to the `PWM_ACTL.DISHI` and `PWM_ACTL.DISLO` bits) also can potentially cause outputs to violate shoot-through condition criteria. Another case is when large values vary the phase delay of a PWM timer. These transitions are detected automatically. If appropriate for safety, an emergency dead-time is inserted to prevent shoot through conditions.

There is another atypical case for the insertion of the additional emergency dead time. It occurs when both PWM signals do not toggle within a dead time of each other. In this case, insert more emergency dead time into one of the PWM signals of a given pair during these transitions. The dead-time delay is inserted into the PWM signal that is toggling into the on-state. In effect, an amount (2 × DT × t_{CK}) from the rising edge of the opposite output delays the turn-on of this signal. After this delay, the PWM signal is allowed to turn-on provided the desired output is still scheduled to be in the on-state after the emergency dead-time delay.

The ***Over Modulation Transition Example*** figure illustrates two examples of such a transition. In the figure, `PWM_ACTL.PULSEMODEHI` is kept at 1. The `PWM_AH` signal has been in full on modulation for some time and, during the current period, its pulse mode is changed to 10, keeping the full on condition. At the half-period boundary, `PWM_AH` is forced to transition to a deasserted state because pulse mode is 10. An emergency dead-time is inserted on the low-side output.

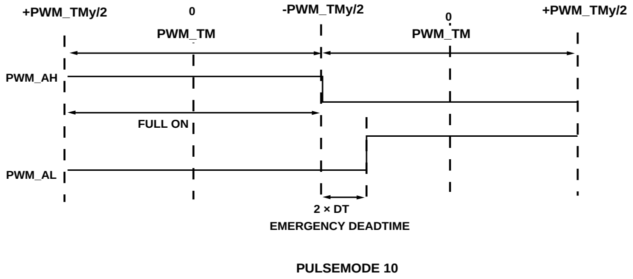


Figure 19-16: Over Modulation Transition Example

Gate Drive Unit

The gate drive unit of the PWM adds features that simplify the design of isolated gate drive circuits for PWM inverters. When using a transformer coupled power device gate drive amplifier, the active PWM signal must be chopped at a high frequency. The `PWM_CHOPCFG` register allows the programming of this chopping mode for high frequency. The chopped active PWM signals can be required for the high-side drivers only, for the low-side drivers only, or for both the high-side and low-side switches. Therefore, independent control of this mode for both high and low-side switches is included with two separate control bits in the `PWM_CHANCFG` register.

The ***High-Side and Low-Side Outputs With Gate Chop Enabled*** figure shows the typical PWM output signals with high-frequency chopping enabled on both high-side and low-side signals. Chopping of the PWM outputs is enabled by setting bits in `PWM_CHANCFG` register. The 8-bit `PWM_CHOPCFG.VALUE` value controls the high frequency chopping. The following equation gives the period of this high frequency carrier.

$T_{chop} = [4 \times (\text{CHOPDIV} + 1)] \times t_{CK}$

and the chopping frequency is therefore an integral subdivision of the peripheral clock frequency:

$f_{chop} = f_{CK}/[4 \times (\text{CHOPDIV} + 1)]$

The `PWM_CHOPCFG.VALUE` value can range from 0 to 255, corresponding to a programmable chopping frequency rate from 122 kHz to 31.25 MHz for a 125 MHz, f_{CK} rate. Program the gate drive features before enabling the PWM controller. Do not change the gate drive features during normal operation of the controller. Following a reset, clear all bits of the `PWM_CHANCFG` register so that high frequency chopping is disabled, by default.

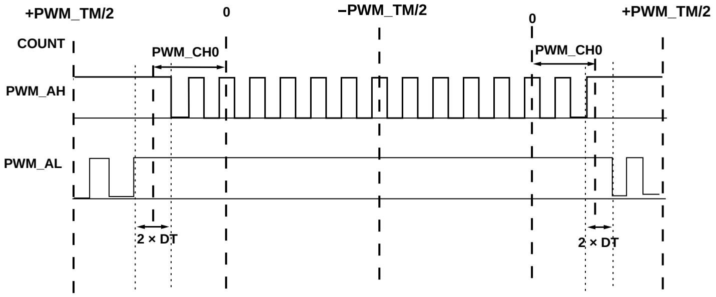


Figure 19-17: High-Side and Low-Side Outputs With Gate Chop Enabled

Output Control Feature Precedence

The order of applying output control features to the PWM signal is important and significant. Use the following order for applying the signal features to the PWM output signal.

1. Duty generation
2. Cross-over
3. High-side or low-side disable
4. Emergency dead-time insertion
5. HPPWM correction for wrong programming
6. Gate-drive chopping
7. Polarity
8. Heightened-Precision PWM (HPPWM) edge placement

NOTE: When HPPWM operation is enabled, the ***cross-over*** feature and the ***gate-drive chopping*** feature must be disabled.

Operating Modes

The PWM generator is capable of operating in the following modes:

- Sync Operation Modes
- Output Disable and Cross-Over Modes
- Heightened-Precision Edge Placement
- Emulation Mode

Sync Operation Modes

The `PWMx_SYNC` pins can operate as internally generated or externally generated. If its internally generated, PWMs can drive the signal to synchronize other PWMs and other devices. If externally generated, the PWMs can be synchronized externally.

External (Triggered) PWM Sync Generation

By setting the `PWM_CTL.EXTSYNC` bit, the PWM is set up in a mode to expect an external `PWM_SYNC` signal on the `PWM_SYNC_IN` pin through the TRU. The trigger source can be any TRU1 trigger master. Multiple PWM units can be precisely synchronized by selecting the same trigger master as the sources for each PWM sync trigger slave. Examples of useful trigger masters include:

- A `PWM_SYNC` GPIO master. This option allows synchronizing the PWMs to an off-chip timing source. (The `PWM_CTL.EXTSYNC` bit must be programmed to 1 to enable the GPIO input.) The `PWM_SYNC` GPIO pads are connected through the pin mux to configurable trigger masters using PWM master IDs TRGM_SYS_PWMn_SYNC_IN. Using the TRU, these masters can be connected to the `PWM_SYNC` trigger slaves in any desired combination.

- A TTU trigger output. This option can include a member of a TTU trigger group, which can also control the timing of other devices such as ADCs, SINC filter inputs, or GP timers. The TTU supports relative trigger delays so that timing offsets can be applied to manage system latencies with precision.
- A general-purpose timer trigger master
- A software trigger master. This option allows starting one or more PWM units simultaneously by an MMR write.

The external `PWM_SYNC` signal only determines the operation of the main timer PWMTMR0.

Synchronize the external sync by setting the `PWM_CTL.EXTSYNCSEL` bit to 0 (assumes the external `PWM_SYNC` selected is asynchronous).

The external `PWM_SYNC` period is expected to be an integer multiple of the value of the `PWM_TM0` period register. When the rising edge of the external `PWM_SYNC` is detected, the PWMTMR0 timer is restarted at the beginning of its period. If the external `PWM_SYNC` period is not exactly an integer multiple of the internal `PWM_SYNC`, the behavior of the PWM channel outputs which are referenced to PWMTMR0 are clipped.

The effect latency from `PWM_SYNC_IN` to the PWM outputs is about three clock cycles in synchronous mode, and five clock cycles in asynchronous mode.

CAUTION: Do not change the value of the `PWM_CTL.EXTSYNC` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).

Output Disable and Cross-Over Modes

Each `PWM_ACTL` channel control register contains separate enable bits for the high and low-side signals. The PWM module uses the `PWM_ACTL.DISHI` and `PWM_ACTL.DISLO` bits in the channel A control register to enable or disable the `PWM_AH` and `PWM_AL` outputs respectively. If the disable bit is set (=1), then the corresponding PWM output is disabled, irrespective of the value of the corresponding duty cycle register. This PWM output signal remains in the OFF state as long as the corresponding enable or disable bit is set.

The cross-over bit (`PWM_ACTL.XOVR`) allows programs to send the low-side output through the high-side output pin and the high-side output through the low-side output pin.

One example uses the following configuration:

- The `PWM_AH0` register =0
- The `PWM_CHANCFG.MODELSC` bit =0
- The `PWM_ACTL.DISLO` bit =1
- The `PWM_ACTL.XOVR` bit =1

The low-side output remains off, as in the case without crossover. The difference in cross-over is that the high-side output changes character and becomes like the low-side. What actually occurs is that the low-side duty cycle is sent to the high-side output pins, and the high-side duty cycle is sent to the low side pins. Because the `PWM_ACTL.DISLO` bit =1, the low-side pin remains off (see Output Control Feature Precedence).

The ***XOVR and DISHI/DISLO Functionality*** figure shows this example. In case 1, `PWM_ACTL.XOVR` =0; and in case 2, `PWM_ACTL.XOVR` =1.

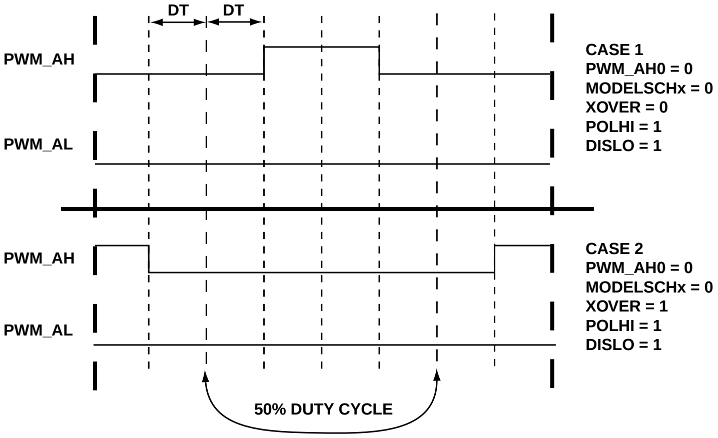


Figure 19-18: XOVR and DISHI/DISLO Functionality

Brushless DC Motor (Electronically Commutated Motor) Control

In the control of an electronically commutated motor (ECM), only two inverter legs are switched at any time. Often, the high-side device in one leg must be switched on at the same time as the low-side driver in a second leg. It is possible to turn on the high-side switch of phase A and the low-side switch of phase B at the same time by:

- Programming identical values for the duty cycles for two PWM channels (for example, `PWM_CH0` = `PWM_CH1`), and
- Setting the `PWM_BCTL.XOVR` bit to crossover the BH and BL pair of PWM signals

To control ECM, normally the third inverter leg (phase C in this example) is disabled for a number of PWM cycles. To implement this function, both the `PWM_CH` and `PWM_CL` outputs are disabled by setting the `PWM_CCTL.DISHI` and `PWM_CCTL.DISLO` bits.

In normal ECM operation, each inverter leg is disabled for certain time periods so that the PWM channel registers change based on the position of the rotor shaft (motor commutation).

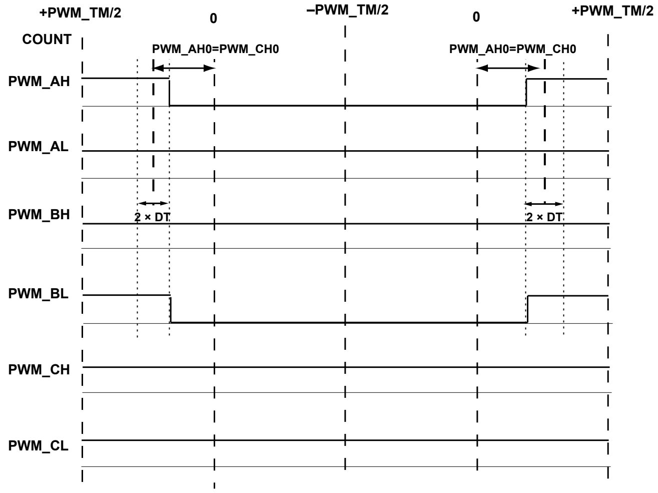

+PWM_TM/2
0
–PWM_TM/2
0
+PWM_TM/2
COUNT
PWM_AH0=PWM_CH0
PWM_AH0=PWM_CH0
PWM_AH
PWM_AL
PWM_BH
2 × DT
2 × DT
PWM_BL
PWM_CH
PWM_CL

Figure 19-19: ECM Control

Heightened-Precision Edge Placement

Heightened-precision edge placement allows a fine-grained edge placement within the system clock period. The ***Heightened-Precision Steps in a Single SCLK Period*** figure shows how the SCLK0_0 aligned edge is moved to finer resolution.

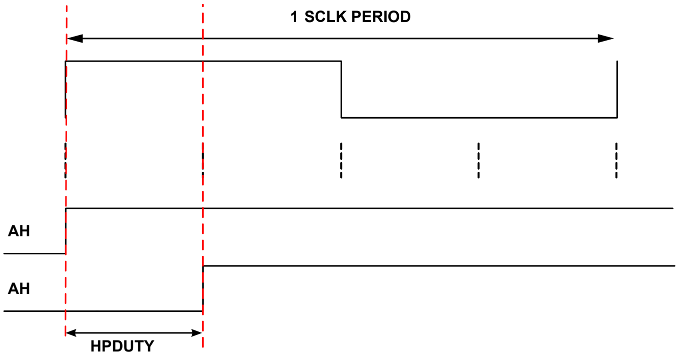

1 SCLK PERIOD
AH
AH
HPDUTY

Figure 19-20: Heightened-Precision Steps in a Single SCLK0_0 Period

The heightened-precision mode is enabled by setting bits that correspond to the high or low side output option for the channel in the `PWM_CHANCFG` register. bit. The `PWM_AH0_HP` and `PWM_AH1_HP` registers work alongside the `PWM_AH0` and `PWM_AH1` registers to provide the overall resolution. The following example explains how signed decimal programming is implied for the heightened-precision duty values.

For the `PWM_AH` output, the duty-cycle register-pair `PWM_AH0` and `PWM_AH0_HP` work together in a Q15.8 signed two's complement fixed-point format as shown in the ***Duty Cycle Notation for Heightened-Precision Edge Placement*** figure. The weight of bit position at k is 2k.

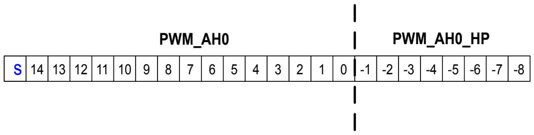


Figure 19-21: Duty Cycle Notation for Heightened-Precision Edge Placement

In the normal modes of operation (not involving heightened-precision edge placement), only the `PWM_AH0` register value is programmed. The duty value programmed is a two's complement integer value. If a value of -1 is desired, the `PWM_AH0` register is programmed with the number 0xFFFF (which is the two's complement of 1 in 16 bits).

In the heightened-precision mode, if a duty value corresponding to 0.75 is required, the equivalent two's complement value of 0.75 in the Q15.8 format is computed: 1111_1111_1111_1111.0100_0000 = 0xFFFF.40. In this case, the `PWM_AH0` register is programmed to 0xFFFF and the `PWM_AH0_HP` register is programmed to 0x40.

Heightened-Precision Edge Placement Example

The following is an example of heightened precision edge placement.

On the *positive side* of the fractional of the duty cycle, at 2 t_{CK} + 0.25 t_{CK}, the values for the `PWM_AH0` and `PWM_AH0_HP` registers are calculated as follows.

The `PWM_AH0` =0x0002 and `PWM_AH0_HP` =0x40 (bits 7:6).

The `PWM_AH_DUTY0` register contains the bit fields from both the `PWM_AH0` and `PWM_AH0_HP` registers. Bits [15:14] represent the decimal part or heightened-precision value and bits [31:16] represent the coarse duty cycle. The value for the combined registers is `PWM_AH_DUTY0` =0x00024000.

On the *negative side* of the fractional of the duty cycle, at –2 t_{CK} – 0.25 t_{CK}, the values for the `PWM_AH0` and `PWM_AH0_HP` registers are calculated as follows:

The coarse register represents the next count of the coarse value for negative values so that –2 becomes –3. These values are the two's complement of the positive offset (value = 3) `PWM_AH0` = 0xFFFD and `PWM_AH0_HP` =0xC0 (bits 7:6).

To derive the correct format for a negative duty-cycle value, for example, –2.25, use: coarse value + 1 =3 for the coarse value and 1 for 0.25. Write out the absolute value as a 32-bit number first:

0000 0000 0000 0011 (.) 0100 0000 0000 0000

Then take the two's complement of the entire 32-bit number:

1111 1111 1111 1101 (.) 1100 0000 0000 0000

NOTE: This value is also written into the full duty register (`PWM_AH_DUTY0`). The correct value for the combined registers written in the `PWM_AH_DUTY0` register is 0xFFFDC000.

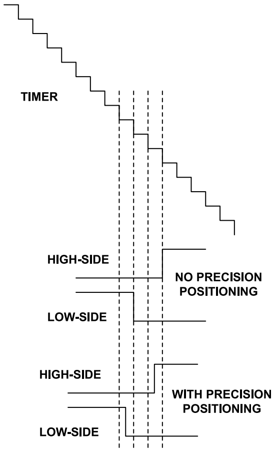
NOTE: The above examples only consider 2 bits of precision, while the ADSP-CM41xF supports up to 4 bits. That means bits [7:4] of the `PWM_AH_DUTY0` register can be used to define the higher precision. Bits

[15:12] of represent the decimal part or heightened-precision value and bits [31:16] represent the coarse duty cycle.

Sample Waveforms for High- and Low-Side with Precision Placement

When the PWM module uses heightened-precision in the dependent mode of operation, both high and low-side outputs shift in the same direction. This operation can result in pulse-expansion of the high-side and pulse-contraction of the low-side or conversely.

The ***Output Shift in The Same Direction*** figure shows an example of a case with DT =1, and pulse expansion occurs on the high-side and pulse-contraction on the low-side. It juxtaposes a case where the PWM module does not use heightened-precision and a case where PWM does use it.



NOTE: BOTH CHANNEL OUTPUTS SHIFT IN THE SAME DIRECTION

Figure 19-22: Output Shift in The Same Direction

The following ***Precision Placement: PULSEMODE*** figures illustrate the cases for pulse modes 1, 2, and 3 (pulse mode 00 is a trivial case of pulse mode 01). The pulse modes are configured in the `PWM_ACTL` and `PWM_BCTL` channel control registers. The figures show what happens to the edges as a decimal part is added to a programmed positive duty. In each case, assume that the original `PWM_AH0.DUTY` register value, which is the coarse duty value, changes to the `PWM_AH1.DUTY` value after programming the enhanced-resolution (`PWM_AH0_HP`, `PWM_AH1_HP`) registers. For example, changing 14 to 14.25 and 10 to 10.75. The figures are not drawn to these numbers. For negative duty values, the shifts are in the opposite direction.

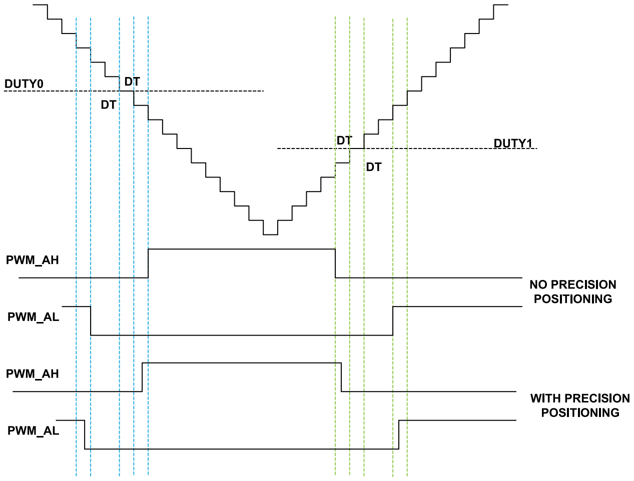

DUTY0
DT
DT
DT
DUTY1
DT
PWM_AH
PWM_AL
NO PRECISION POSITIONING
PWM_AH
PWM_AL
WITH PRECISION POSITIONING

Figure 19-23: Precision Placement: PULSEMODE=01

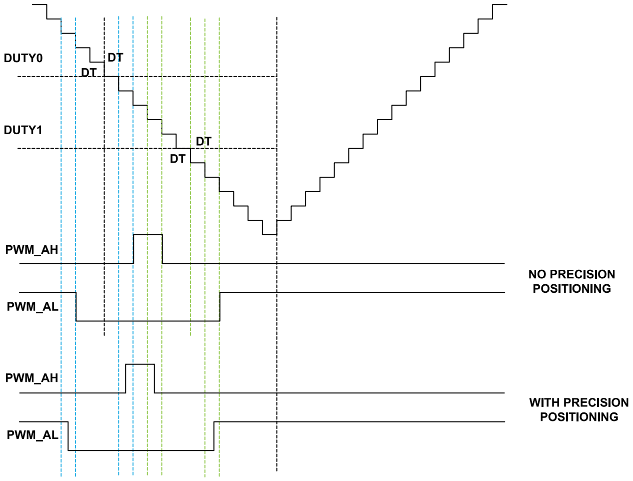

DUTY0
DT
DT
DUTY1
DT
DT
PWM_AH
PWM_AL
NO PRECISION POSITIONING
PWM_AH
PWM_AL
WITH PRECISION POSITIONING

Figure 19-24: Precision Placement: PULSEMODE=10

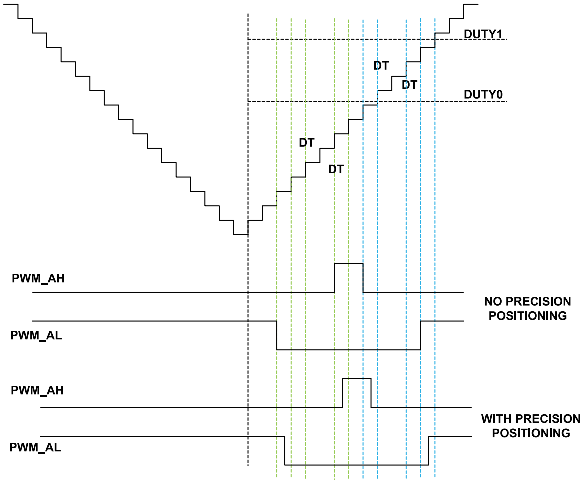

DUTY1
DT
DT
DUTY0
DT
DT
PWM_AH
PWM_AL
NO PRECISION POSITIONING
PWM_AH
PWM_AL
WITH PRECISION POSITIONING

Figure 19-25: Precision Placement: PULSEMODE=11

Emulation Mode

Depending upon system configuration, a debug halt may be presented to the PWM unit. The PWM has local control of its response to an emulator halt based on the `PWM_CTL.EMURUN` bit setting.

- `PWM_CTL.EMURUN` =1. When the processor is halted in emulation mode, the outputs continue to toggle and be driven out of the PWM block. The counters and status register bits are set or reset according to the PWM timer count or period settings.
- `PWM_CTL.EMURUN` =0. When the processor is halted in emulation mode, the outputs are shut down (enter their inactive state based on polarity), and all counters that affect status register bits are paused.
- At restart, the PWM counters resume from their paused value. The outputs are still held in their inactive state.

Event Control

The PWM uses bits in the `PWM_IMSK` and `PWM_ILAT` registers for event control. These registers allow masking and show masked interrupt request status bits, respectively. The interrupt status bits are latched and held on the interrupt event. The software must write a 1 to clear the interrupt status bit, usually during the interrupt service routine.

The timer period (TMRxPER) interrupt requests are configured using the `PWM_ILAT.TMR0PER` - `PWM_ILAT.TMR4PER` bits. The PWM uses the interrupts to execute an interrupt service routine (ISR) periodically. The ISR updates the PWM channel control and duty registers (according to a control algorithm based on expected operation and sampled existing operation). The TMRxPER interrupts also can trigger an ADC to sample data for use during the ISR.

The PWM uses the `PWM_CTL.INTSYNCREF` bit field to control the PWM_SYNC interrupt request. The PWM uses the bit field to assign the interrupt request to a system user interrupt of the core. The PWM_SYNC can be configured to be either internal or externally driven using the `PWM_CTL.EXTSYNC` bit. When configured as an external sync, the signal can be further configured as synchronous or asynchronous using the `PWM_CTL.EXTSYNCSEL` bit.

As an example, when the PWM_SYNC interrupt is serviced:

- The ADC samples data
- The data is algorithmically interpreted
- New PWM channel duties are calculated and written to the PWM

More sophisticated implementations include different startup, run time, and shutdown algorithms to determine PWM channel duties based on expected behavior and further features.

During the PWM_SYNC interrupt driven control loop, only the channel delay registers, the duty registers, and the channel C high pulse duty register values are typically updated. To see programming limitations on the PWM registers, see the "Register Descriptions".

Status information about the PWM is available in the `PWM_STAT` register, which stores all status bits, including raw interrupt status bits. In particular, the period boundary of each timer is available, as well as status bits. The PWM uses the status bits to indicate whether the operation is in the first half or the second half of the timer. Additionally, the TRIP status is also available. For more information on TRIP interrupts, see Trip Control Unit .

Trip Control Unit

The PWM output signals can be shut-off in a number of different ways. The trip inputs (`PWM_TRIP[n]`) can be mapped to provide either a temporary or permanent shutdown on any channel outputs (hi/lo/pair). This shutdown mechanism is asynchronous so that the associated PWM output disable circuitry does not go through any clocked logic. This functionality ensures correct PWM shutdown even in the event of a loss of the processor clock. In addition to the hardware shutdown features, the PWM system can be shut down in software with the `PWM_CTL.SWTRIP` bit.

The external trip signal `PWM_TRIP0`, that comes from GPIO generates the TRIP0 input whereas the trigger slaves PWM0_TRIP_TRIGn, PWM1_TRIP_TRIGn and PWM2_TRIP_TRIGn generate TRIP1 input.

During any external trip event (if not disabled), the PWM outputs are turned off. When a PWM output is turned off, it means that the output level is held at a polarity opposite that given in the `PWM_CHANCFG.POLAH` through `PWM_CHANCFG.POLDH` (channel high side polarity) bits. The PWM sync pulse continues to operate, when it is already enabled. A PWMTRIP interrupt occurs if unmasked, to notify the software of this event. In dependent mode of operation, both high and low-side outputs refer to the channel high side polarity bits.

Even if the clock to the PWM is damaged, an external trip event turns off the PWM outputs. In this case, the PWMTRIP interrupt request may not occur.

The PWM trip unit processes hardware or software fault conditions and shuts down the PWM channel outputs immediately on the occurrence of these conditions. The PWM can enable the shutdown mechanism separately for each channel. The design also allows for a self-restart mechanism to be enabled on a channel. Self-restart reenables the channel outputs following the fault condition (allowed only on hardware trips) when the PWMTMRy that the channel is using reaches its period boundary.

These sources are active low inputs where a falling edge on either of these pins indicates a fault condition.

The trip unit can shut down an output of a particular channel in response to the fault event from the `PWM_TRIP0` pin. To enable this functionality, program the `PWM_TRIPCFG.EN0A` bit corresponding to the channel.

Program the `PWM_TRIPCFG.MODE0A` bits to specify the restart mechanism for a channel that has been tripped.

1. If the `PWM_TRIPCFG.MODE0A` bit =0, once tripped, a trip condition is registered on this channel in the `PWM_STAT.FLTTRIPA` bit and the outputs of that channel are immediately shut down. This condition is called a *fault trip* condition. To resume channel output when a fault trip occurs, write a 1 to clear the `PWM_STAT.FLTTRIPA` bit. A processor write cannot clear the bit when the trip condition is still active. The raw trip status is available for both pins in the `PWM_STAT.RAWTRIP0` register bits.
2. If the `PWM_TRIPCFG.MODE0A` bit =1, once tripped, a trip condition is registered on this channel in the `PWM_STAT.SRTRIPA` bit and the outputs of that channel are immediately shut down. This condition is

called a *self-restart trip* condition. If the trip condition is not active at the next period boundary of the PWMTMRy that the channel is using, the status register bit is cleared. The outputs are restored.

The trip input pins have an external pull-down resistor on the chip pin. If the pin becomes disconnected, the processor disables the PWM.

In addition to the hardware trip conditions, a global software trip bit in the `PWM_CTL` register allows for a software-forced fault trip condition. When the global software trip bit is set to 1, irrespective of the values in the `PWM_TRIPCFG` register, it sets all the `PWM_STAT.FLTTRIPA` bits and also gates the channel outputs. To remove the trip condition from the channel, perform a W1C on the `PWM_STAT.FLTTRIPA` bit of the particular channel.

If the `PWM_TRIPCFG.EN0A` bit is set to 1 to, for any channel, then the occurrence of a fault condition on the `PWMTRIPy` bit is logged in the `PWM_STAT.FLTTRIPA` register bit. If the corresponding `PWM_IMSK.TRIP0` bit = 1, then an interrupt request is generated. Tripping a channel output does not interfere with `PWM_SYNC` generation.

The ***Operation Under Hardware Fault Conditions*** figure shows an example where PWMTRIP0 is enabled on channel A as self-restart trip. Channel A works with the `PWM_CHANCFG.POLAH` bit =1. In period 2, the `PWM_AH` signal is full on modulated, and tries to rise at the period boundary where the self-restart occurs for the channel. However, since the low-side output of the channel was only recently removed due to a trip, the rise edge on `PWM_AH` is delayed until the emergency dead-time period is over. PWMTRIP1 is enabled on channel B as a fault trip. Channel B works with the `PWM_CHANCFG.POLAH` bit =0. PWMTRIP1 stays low for an extended time period. The first processor write to reenable the channel output fails. The second processor write passes since the fault condition has gone away.

NOTE: Dead time is ensured on re-enabling the channel outputs after trip.

NOTE: Programs must not allow changes in the configuration or enable bits of `PWM_TRIPCFG` register within ± 10 clock cycles of when the external trip pulse toggles. (The configuration or enable bits of `PWM_TRIPCFG` register select between trip enable and disable). If this time frame is not followed, then unexpected behavior occurs.

Programming Model

The following sections provide general (and some application-specific) programming steps for configuring and using the PWM module.

- Programming Model for Three-Phase AC Motor Control

Programming Model for Three-Phase AC Motor Control

The ***PWM Module and Interaction with System*** figure shows how the PWM unit (green) interfaces to both software (blue) and hardware (yellow). The software configures the unit, calculates duty cycles (Duty A, Duty B, Duty C), and services the interrupts generated by the module (PWM SYNC IRQ, TRIP IRQ). The hardware applies the gate signals (AH, AL, BH, BL, CH, CL) to the inverter and provides an over-current trip signal back to the unit (TRIP0).

The typical three-phase AC motor configuration shown in the ***PWM Module and Interaction with System*** figure applies for both permanent magnet and induction motor types.

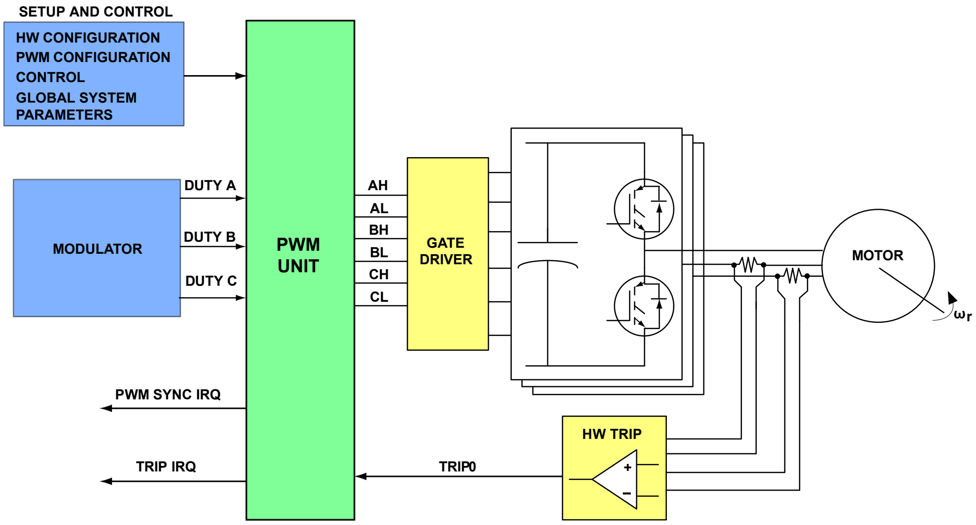

SETUP AND CONTROL
HW CONFIGURATION
PWM CONFIGURATION
CONTROL
GLOBAL SYSTEM PARAMETERS
MODULATOR
DUTY A
DUTY B
DUTY C
PWM UNIT
AH
AL
BH
BL
CH
CL
GATE DRIVER
MOTOR
ωr
PWM SYNC IRQ
TRIP IRQ
TRIP0
HW TRIP
+
−

Figure 19-26: PWM Module and Interaction with System

System Parameters

The following system parameters (characteristics) influence the module configuration for this application. This example system features:

- One three-phase AC machine
- B6 inverter
- SVPWM, including both linear- and over-modulation
- Switching frequency of 20 kHz
- Dead time of 1us
- Trip signal generated by hardware
- Active high-level gate drive
- Core frequency of 200 MHz
- Peripheral clock of 100 MHz

System State Sequencing

Managing the system state and sequence of states is critically important when programming the PWM module. The ***PWM System States*** figure provides an overview of these states.

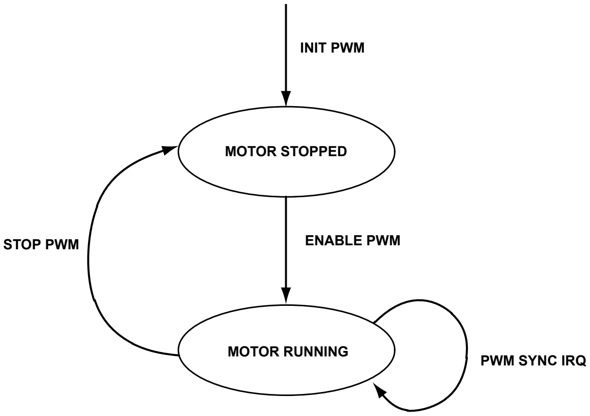


Figure 19-27: PWM System States

As shown in the state diagram, the module configuration is updated on state transitions (indicated by the arrows). The transitions are initialization, motor start, PWM sync interrupt request (on each), and motor stop. The following sections discuss the transitions in detail.

- PWM Initialization for Motor Control
- PWM Enable for Motor Control
- PWM Response to Sync Interrupt for Motor Control
- PWM Disable (and Stop the Motor) for Motor Control

PWM Initialization for Motor Control

The processor must program the PWM at power-up as follows and repeat this programming to bring the PWM and the system into a known (safe) state.

1. Place the PWM module in a safe state and set up synchronization of the module using the following bitwise operations on the `PWM_CTL` and `PWM_CHANCFG` registers:

   ```
   PWM_CTL &= 0xFFE0FF08
   PWM_CTL |= 0x20000
   PWM_CHANCFG &= 0x80808080
   PWM_CHANCFG |= 0x24242424
   ```

   These operations result in the following bit settings:

   - Disable PWM (`PWM_CTL.GLOBEN` =0)
   - Disable delay for channels A, B, C, D (`PWM_CTL.DLYAEN` through `PWM_CTL.DLYDEN` =0). All phases must run with same phase.
   - Use internal synchronization by timer TMR0 (`PWM_CTL.EXTSYNC` =0, `PWM_CTL.EXTSYNCSEL` =1)
   - The same timer, TMR0 (`PWM_CTL.INTSYNCREF` = b#000) synchronizes all phases.

- Low-side is always the inverse of high-side (`PWM_CHANCFG.POLAL` through `PWM_CHANCFG.POLDL` = 1).
- System uses active high gate driver (`PWM_CHANCFG.ENCHOPAH` through `PWM_CHANCFG.ENCHOPDH` =1).
- Disable gate chopping (`PWM_CHANCFG.ENCHOPAL` through `PWM_CHANCFG.ENCHOPDL` =0). PWM does not use the pulse transformer.

2. Set up the trip and associated interrupt requests using the following bitwise operations on the `PWM_TRIPCFG` and `PWM_ILAT` registers:

```
PWM_TRIPCFG &= 0xF0F0F0F0
PWM_TRIPCFG |= 0x1010101
PWM_ILAT &= 0xFFE0FFFC
PWM_ILAT |= 0x1
```

These operations result in the following bit settings:

- All phases must shut down simultaneously in case of fault: (`PWM_TRIPCFG.EN0A` through `PWM_TRIPCFG.EN0D` =0, `PWM_TRIPCFG.MODE0A` through `PWM_TRIPCFG.MODE0D` =0, `PWM_TRIPCFG.EN1A` through `PWM_TRIPCFG.EN1D` =0, `PWM_TRIPCFG.MODE1A` through `PWM_TRIPCFG.MODE1D` =0)
- Enable TRIP0 as fault trigger for all channels (`PWM_TRIPCFG.EN0A` through `PWM_TRIPCFG.MODE1D` =1).
- For thermal control and synchronization, SW intervention is needed at trip. Do not use automatic restart of any channels.
- Generate an interrupt at trip on TRIP0 (`PWM_ILAT.TMR0PER` = 1).

3. Configure the PWM channels using the following bitwise operations on the `PWM_TRIPCFG` and `PWM_ILAT` registers:

```

PWM_CHA_DT=0x32 PWM_CHB_DT=0x32 PWM_CHC_DT=0x32
PWM_TM0 = 0x9C4
PWM_ACTL = 0xFFFFF0000
PWM_BCTL = 0xFFFFF0000
PWM_CCTL = 0xFFFFF0000
PWM_AH0 = 0x0
PWM_BH0 = 0x0
PWM_CH0 = 0x0
```

These operations result in the following bit settings:

- Configure a dead time of 1 μs (DT =0x32 =0x32)
- Configure a PWM frequency of 20 kHz (assuming a system clock frequency of 100Mhz, so a clock divisor of 5000:1) (`PWM_TM0` =0x9C4)

- Disable all outputs (`PWM_ACTL.DISHI` through `PWM_CCTL.DISHI` =0, `PWM_ACTL.DISLO` through `PWM_CCTL.DISLO` =0)
- Use conventional PWM, disable crossover (`PWM_ACTL.XOVR` through `PWM_CCTL.XOVR` =0)
- Use symmetrical pulse position on all outputs (`PWM_ACTL.PULSEMODEHI` through `PWM_CCTL.PULSEMODEHI` =0, `PWM_ACTL.PULSEMODELO` through `PWM_CCTL.PULSEMODELO` =0)
- Set an initial duty-cycle of 50% (`PWM_AH0` through`PWM_CH0` =0x0)

PWM Enable for Motor Control

The processor must do the following programming to enable the PWM before starting the motor.

1. Start the PWM timer TMR0 using the `PWM_CTL |=0x1` bitwise operation on the `PWM_CTL` register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the `PWM_CTL.GLOBEN` bit =1.

2. Enable six PWM outputs using the following bitwise operations on the `PWM_ACTL` through `PWM_CCTL` registers.

   ```
   PWM_ACTL|=0x3
   PWM_BCTL|=0x3
   PWM_CCTL|=0x3
   ```

   ADDITIONAL INFORMATION: These operations have the same effect as enabling high and low-side channel outputs by setting the `PWM_ACTL.DISHI` through `PWM_CCTL.DISHI` bits =1 and the `PWM_ACTL.DISLO` through `PWM_CCTL.DISLO` bits =1.

3. Enable the PWM TRIP0 interrupt using the `PWM_ILAT |= 0x1` bitwise operation on the `PWM_ILAT` register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the `PWM_ILAT.TRIP0` bit =1

PWM Response to Sync Interrupt for Motor Control

When the PWM sync interrupt occurs, the processor could need to update to the PWM duty cycle with a value calculated by the motor control algorithm. This application uses symmetric pulses position and uses dependent high and low-side output. So, the PWM updates only one register for each phase.

1. Write the new duty cycle value (calculated by motor control algorithm) to the timer when the sync interrupt occurs.

   The following bitwise operations on the `PWM_AH0` through `PWM_CH0` registers accomplish this task:

```
PWM_AH0 = Duty_A_mc_algorithm_current_value
PWM_BH0 = Duty_B_mc_algorithm_current_value
PWM_CH0 = Duty_C_mc_algorithm_current_value
```

PWM Disable (and Stop the Motor) for Motor Control

The processor must program the PWM as follows to stop the motor, disable the PWM, and disable PWM interrupts. These actions place the PWM and system in a safe, passive state.

1. Disable the PWM timer using the `PWM_CTL &= 0xFFFFFFFE` bitwise operation on the `PWM_CTL` register.

   ADDITIONAL INFORMATION: This operation has the same effect as clearing the `PWM_CTL.GLOBEN` bit =0.

2. Disable all PWM outputs using the following bitwise operations on the `PWM_ACTL` through `PWM_CCTL` registers.

   ```
   PWM_ACTL &= 0xFFFFFFFFC
   PWM_BCTL &= 0xFFFFFFFFC
   PWM_CCTL &= 0xFFFFFFFFC
   ```

   These operations disable PWM outputs where the `PWM_ACTL.DISHI` through `PWM_CCTL.DISHI` bits =1 and the `PWM_ACTL.DISLO` through `PWM_CCTL.DISLO` bits =0)

3. Set the PWM duty-cycle to 50% using the following bitwise operations on the `PWM_AH0` through `PWM_CH0` registers.

   ```
   PWM_AH0 =0x0
   PWM_BH0 =0x0
   PWM_CH0 =0x0
   ```

   These operations have the same effect as clearing the `PWM_AH0.DUTY` through `PWM_CH0.DUTY` bit =0.

4. Disable the PWM TRIP0 interrupt request using the `PWM_ILAT &= 0xFFFFFFFE` bitwise operation on the `PWM_ILAT` register.

   ADDITIONAL INFORMATION: This operation has the same effect as clearing the PWM TRIP0 interrupt request `PWM_ILAT.TRIP0` =0.

ADSP-SC58x PWM Register Descriptions

Pulse-Width Modulator (PWM) contains the following registers.

Table 19-5: ADSP-SC58x PWM Register List

Name	Description
`PWM_ACTL`	Channel A Control Register

Table 19-5: ADSP-SC58x PWM Register List (Continued)

Name	Description
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register
PWM_BL0_HP	Channel B-Low Heightened-Precision Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_BL1_HP	Channel B-Low Heightened-Precision Duty-1 Register
PWM_BL_DUTY0	Channel B-Low Full Duty0 Register
PWM_BL_DUTY1	Channel B-Low Full Duty1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH0_HP	Channel C-High Pulse Heightened-Precision Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CH1_HP	Channel C-High Pulse Heightened-Precision Duty Register 1
PWM_CHANCFG	Channel Configuration Register

Table 19-5: ADSP-SC58x PWM Register List (Continued)

Name	Description
PWM_CHA_DT	Channel A Dead-time Register
PWM_CHB_DT	Channel B Dead-time Register
PWM_CHC_DT	Channel C Dead-time Register
PWM_CHD_DT	Channel D Dead-time Register
PWM_CHOPCFG	Chop Configuration Register
PWM_CH_DUTY0	Channel C-High Full Duty0 Register
PWM_CH_DUTY1	Channel C-High Full Duty1 Register
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL0_HP	Channel C-Low Pulse Duty Register 1
PWM_CL1	Channel C-Low Duty-1 Register
PWM_CL1_HP	Channel C-Low Heightened-Precision Duty-1 Register
PWM_CL_DUTY0	Channel C-Low Full Duty0 Register
PWM_CL_DUTY1	Channel C-Low Full Duty1 Register
PWM_CTL	Control Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH0_HP	Channel D-High Pulse Heightened-Precision Duty Register 0
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DH1_HP	Channel D High Pulse Heightened-Precision Duty Register 1
PWM_DH_DUTY0	Channel D-High Full Duty0 Register
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register
PWM_DL1	Channel D-Low Pulse Duty Register 1
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register

Table 19-5: ADSP-SC58x PWM Register List (Continued)

Name	Description
`PWM_ILAT`	Interrupt Latch Register
`PWM_IMSK`	Interrupt Mask Register
`PWM_STAT`	Status Register
`PWM_SYNC_WID`	Sync Pulse Width Register
`PWM_TM0`	Timer 0 Period Register
`PWM_TM1`	Timer 1 Period Register
`PWM_TM2`	Timer 2 Period Register
`PWM_TM3`	Timer 3 Period Register
`PWM_TM4`	Timer 4 Period Register
`PWM_TRIPCFG`	Trip Configuration Register

Channel A Control Register

The PWM_ACTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

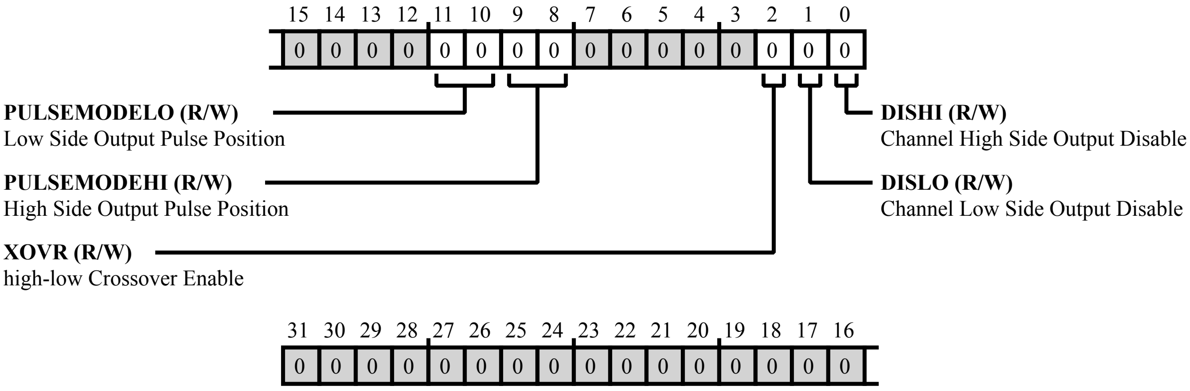

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PULSEMODELO (R/W)
Low Side Output Pulse Position
PULSEMODEHI (R/W)
High Side Output Pulse Position
XOVR (R/W)
high-low Crossover Enable
DISHI (R/W)
Channel High Side Output Disable
DISLO (R/W)
Channel Low Side Output Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-28: PWM_ACTL Register Diagram

Table 19-6: PWM_ACTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_ACTL.PULSEMODELO bits select the pulse position for Channel A low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 19-6: PWM_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The `PWM_ACTL.PULSEMODEHI` bits select the pulse position for Channel A high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the `PWM_AH0` register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (`PWM_AH0` and `PWM_AH1`). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (`PWM_AH0` and `PWM_AH1`).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The `PWM_ACTL.XOVR` bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The `PWM_ACTL.DISLO` bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The `PWM_ACTL.DISHI` bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel A-High Duty-0 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_ACTL.PULSEMODEHI` bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AH0 register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel A high pulse output for a count less than PWM_AH0 and deasserts this output for a count greater than PWM_AH1.

The value range for the PWM_AH0 and PWM_AH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHA_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AH0 and PWM_AH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHA_DT (two's complement) and +PWM_TM0/2 + PWM_CHA_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHA_DT (two's complement) and -PWM_TM0/2 - PWM_CHA_DT .

Note that using values in the PWM_AH0 or PWM_AH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

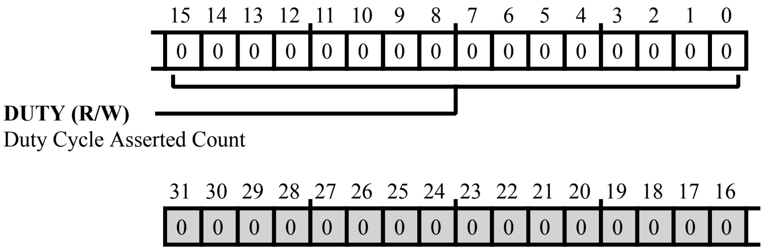


Figure 19-29: PWM_AH0 Register Diagram

Table 19-7: PWM_AH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_AH0.DUTY` bits select the duty cycle asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-0 Register

The `PWM_AH0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_AH0` register, allows programs to specify fractional duty cycles.The `PWM_AH0_HP` register and the `PWM_AH0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_AH0_HP` and the `PWM_AH0` registers are also present in the single full duty register (if available).

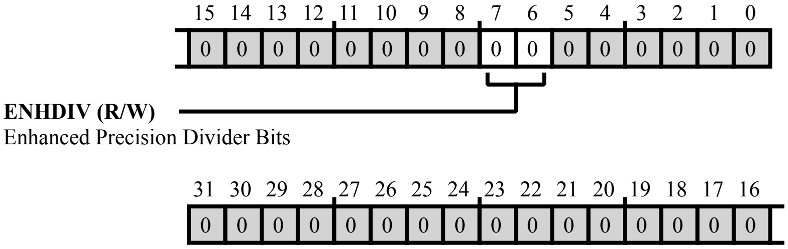

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-30: PWM_AH0_HP Register Diagram

Table 19-8: PWM_AH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits.

Channel A-High Duty-1 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. For more information, see the PWM_AH0 register description.

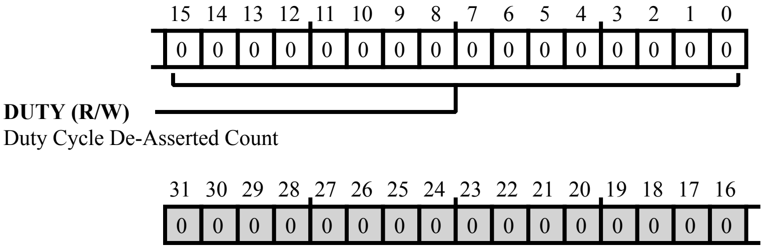

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-31: PWM_AH1 Register Diagram

Table 19-9: PWM_AH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_AH1.DUTY` bits select the duty cycle de-asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-1 Register

The `PWM_AH1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_AH0` register, allows programs to specify fractional duty cycles.The `PWM_AH1_HP` register and the `PWM_AH1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_AH1_HP` and the `PWM_AH1` registers are also present in the single full duty register (if available).

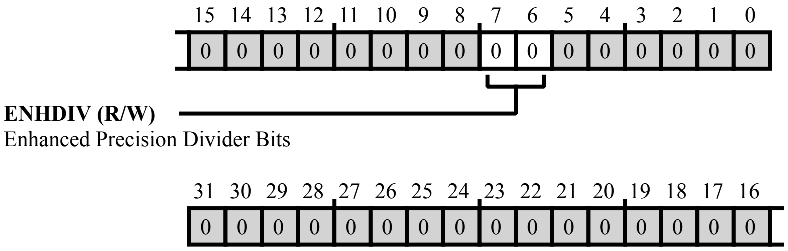

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-32: PWM_AH1_HP Register Diagram

Table 19-10: PWM_AH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AH1_HP.ENHDIV` bits provide fractional duty cycles for Channel A high side output.

Channel A-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_AH_DUTY0` register contains the `PWM_AH_DUTY0.DUTY` bit field from the `PWM_AH0` register and the `PWM_AH_DUTY0.ENHDIV` bit field from the `PWM_AH0_HP` register.

Note that the `PWM_AH_DUTY0` register reads the `PWM_AH0` and the `PWM_AH0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

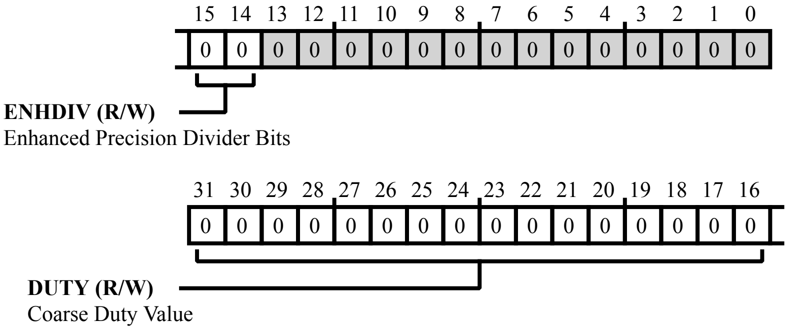

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-33: PWM_AH_DUTY0 Register Diagram

Table 19-11: PWM_AH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_AH_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_AH_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AH_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_AH_DUTY1` register contains the `PWM_AH_DUTY1.DUTY` bit field from the `PWM_AH1` register and the `PWM_AH_DUTY1.ENHDIV` bit field from the `PWM_AH1_HP` register.

Note that the `PWM_AH_DUTY1` register reads the `PWM_AH1` and the `PWM_AH1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

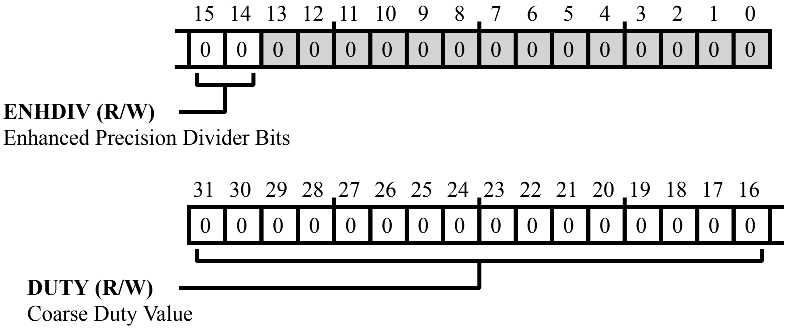

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-34: PWM_AH_DUTY1 Register Diagram

Table 19-12: PWM_AH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_AH_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_AH_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AH_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Duty-0 Register

The `PWM_AL0` and `PWM_AL1` registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_ACTL.PULSEMODELO` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_AL0` register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel A low pulse output for count less than `PWM_AL0` and deasserts this output for count greater than `PWM_AL1`.

The value range for the `PWM_AL0` and `PWM_AL1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHA_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_AL0` and `PWM_AL1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHA_DT` (two's complement) and +`PWM_TM0`/2 + `PWM_CHA_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHA_DT` (two's complement) and -`PWM_TM0`/2 - `PWM_CHA_DT`.

Note that using values in the `PWM_AL0` or `PWM_AL1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

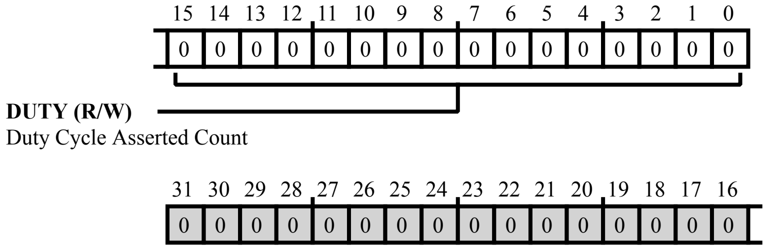


Figure 19-35: PWM_AL0 Register Diagram

Table 19-13: PWM_AL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_AL0.DUTY` bits select the duty cycle asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-0 Register

The `PWM_AL0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_AL0` register, allows programs to specify fractional duty cycles.The `PWM_AL0_HP` register and the `PWM_AL0` register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the `PWM_AL0_HP` and the `PWM_AL0` registers are also present in the single full duty register (if available).

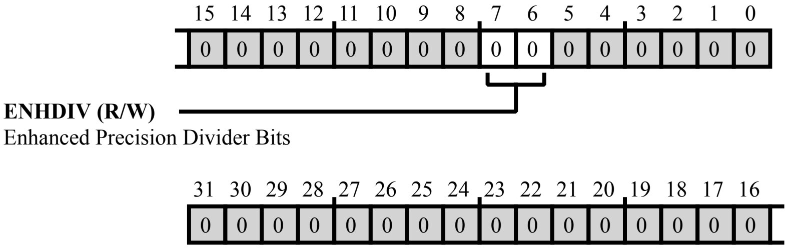

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-36: PWM_AL0_HP Register Diagram

Table 19-14: PWM_AL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AL0_HP.ENHDIV` bits provide fractional duty cycles for Channel A low side output.

Channel A-Low Duty-1 Register

The `PWM_AL0` and `PWM_AL1` registers determine the width for the low side output pulses. For more information, see the `PWM_AL0` register description.

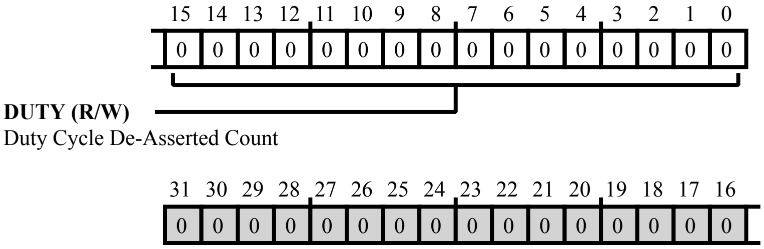

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-37: PWM_AL1 Register Diagram

Table 19-15: PWM_AL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_AL1.DUTY` bits select the duty cycle de-asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-1 Register

The `PWM_AL1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_AL1` register, allows programs to specify fractional duty cycles.The `PWM_AL1_HP` register and the `PWM_AL1` register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the `PWM_AL1_HP` and the `PWM_AL1` registers are also present in the single full duty register (if available).

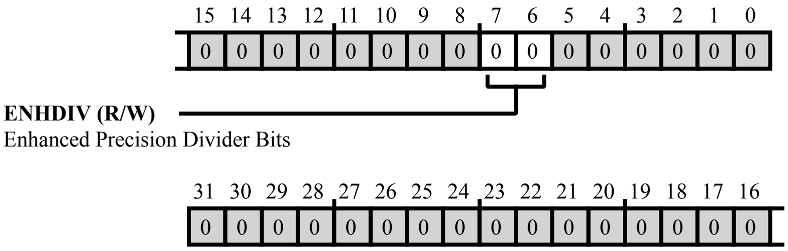

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-38: PWM_AL1_HP Register Diagram

Table 19-16: PWM_AL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AL1_HP.ENHDIV` bits provide fractional duty cycles for Channel A low side output.

Channel A-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_AL_DUTY0` register contains the `PWM_AL_DUTY0.DUTY` bit field from the `PWM_AL0` register and the `PWM_AL_DUTY0.ENHDIV` bit field from the `PWM_AL0_HP` register.

Note that the `PWM_AL_DUTY0` register reads the `PWM_AL0` and the `PWM_AL0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

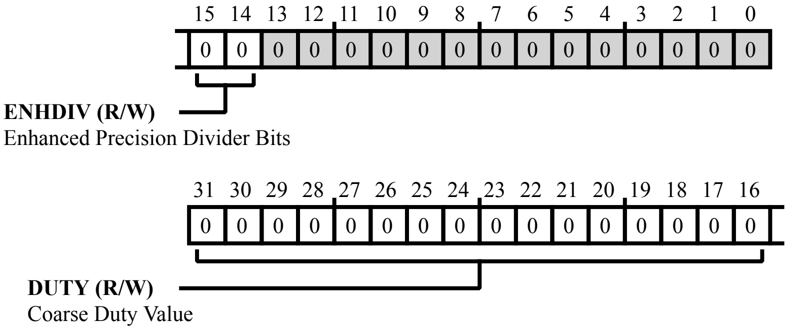

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-39: PWM_AL_DUTY0 Register Diagram

Table 19-17: PWM_AL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_AL_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_AL_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AL_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_AL_DUTY1` register contains the `PWM_AL_DUTY1.DUTY` bit field from the `PWM_AL1` register and the `PWM_AL_DUTY1.ENHDIV` bit field from the `PWM_AH0_HP` register.

Note that the `PWM_AL_DUTY1` register reads the `PWM_AL1` and the `PWM_AL1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

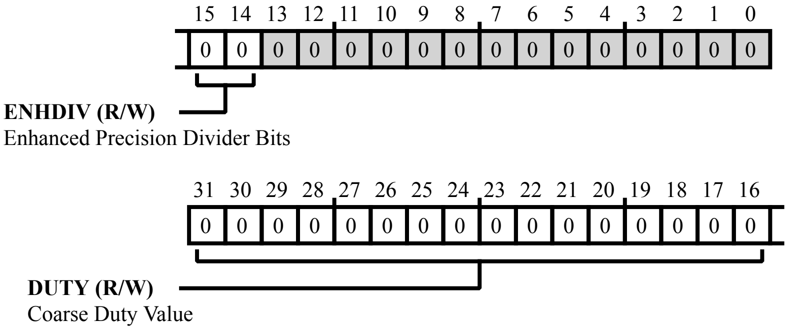

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-40: PWM_AL_DUTY1 Register Diagram

Table 19-18: PWM_AL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_AL_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_AL_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_AL_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B Control Register

The `PWM_BCTL` register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

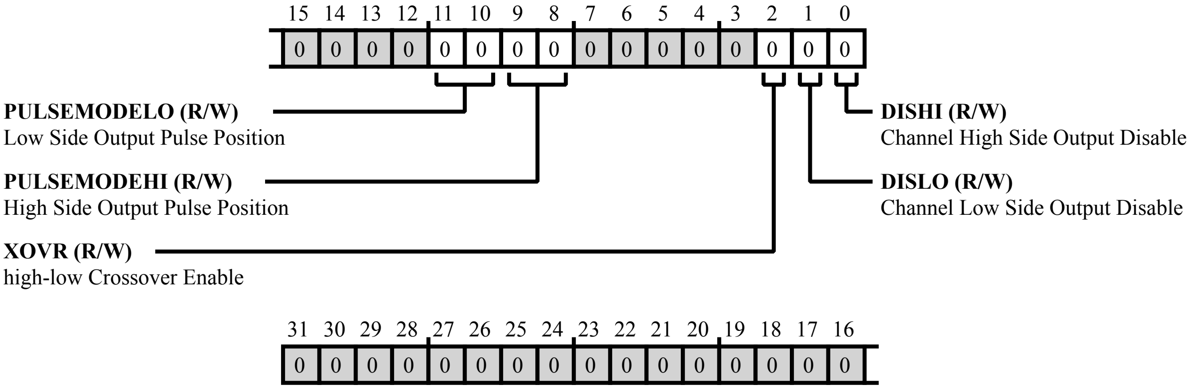

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PULSEMODELO (R/W)
Low Side Output Pulse Position
PULSEMODEHI (R/W)
High Side Output Pulse Position
XOVR (R/W)
high-low Crossover Enable
DISHI (R/W)
Channel High Side Output Disable
DISLO (R/W)
Channel Low Side Output Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-41: PWM_BCTL Register Diagram

Table 19-19: PWM_BCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The `PWM_BCTL.PULSEMODELO` bits select the pulse position for Channel B low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the `PWM_BL0` register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (`PWM_BL0` and `PWM_BL1`). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (`PWM_BL0` and `PWM_BL1`).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 19-19: PWM_BCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The `PWM_BCTL.PULSEMODEHI` bits select the pulse position for Channel B high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the `PWM_BH0` register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (`PWM_BH0` and `PWM_BH1`). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (`PWM_BH0` and `PWM_BH1`). 0 Symmetrical 1 Asymmetrical 2 Left Half 3 Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The `PWM_BCTL.XOVR` bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin. 0 Disable Crossover 1 Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The `PWM_BCTL.DISLO` bit enables the channels low side output. 0 Enable Low Side Output 1 Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The `PWM_BCTL.DISHI` bit enables the channels high side output. 0 Enable High Side Output 1 Disable High Side Output

Channel B-High Duty-0 Register

The `PWM_BH0` and `PWM_BH1` registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_BCTL.PULSEMODEHI` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_BH0` register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel B high pulse output for count less than `PWM_BH0` and deasserts this output for count greater than `PWM_BH1`.

The value range for the `PWM_BH0` and `PWM_BH1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHB_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_BH0` and `PWM_BH1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHB_DT` (two's complement) and +`PWM_TM0`/2 + `PWM_CHB_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHB_DT` (two's complement) and -`PWM_TM0`/2 - `PWM_CHB_DT`.

Note that using values in the `PWM_BH0` or `PWM_BH1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

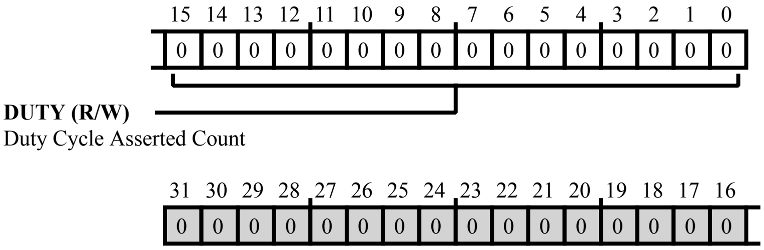


Figure 19-42: PWM_BH0 Register Diagram

Table 19-20: PWM_BH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count.

Channel B-High Heightened-Precision Duty-0 Register

The `PWM_BH0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_BH0` register, allows programs to specify fractional duty cycles. The `PWM_BH0_HP` register and the `PWM_BH0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_BH0_HP` and the `PWM_BH0` registers are also present in the single full duty register (if available).

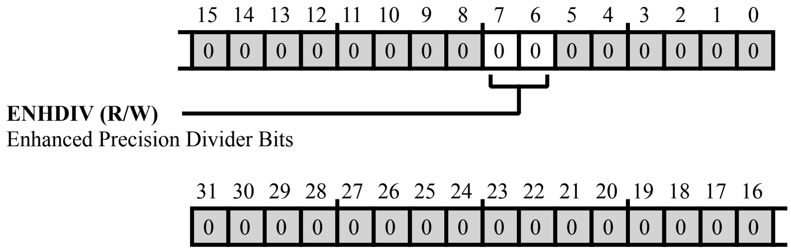

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-43: PWM_BH0_HP Register Diagram

Table 19-21: PWM_BH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BH0_HP.ENHDIV` bits provide fractional duty cycles for Channel B high side output.

Channel B-High Duty-1 Register

The `PWM_BH0` and `PWM_BH1` registers determine the width for the high side output pulses. For more information, see the `PWM_BH0` register description.

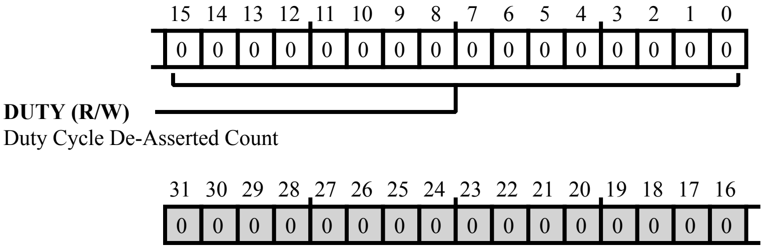

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-44: PWM_BH1 Register Diagram

Table 19-22: PWM_BH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel B-High Heightened-Precision Duty-1 Register

The `PWM_BH1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_BH1` register, allows programs to specify fractional duty cycles. The `PWM_BH1_HP` register and the `PWM_BH1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_BH1_HP` and the `PWM_BH1` registers are also present in the single full duty register (if available).

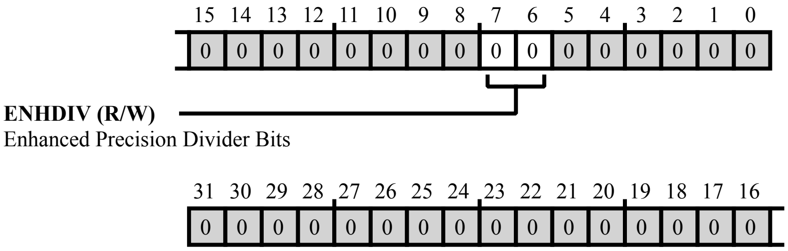

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-45: PWM_BH1_HP Register Diagram

Table 19-23: PWM_BH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BH1_HP.ENHDIV` bits provide fractional duty cycles for Channel B high side output.

Channel B-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_BH_DUTY0` register contains the `PWM_BH_DUTY0.DUTY` bit field from the `PWM_BH0` register and the `PWM_BH_DUTY0.ENHDIV` bit field from the `PWM_BH0_HP` register.

Note that the `PWM_BH_DUTY0` register reads the `PWM_BH0` and the `PWM_BH0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

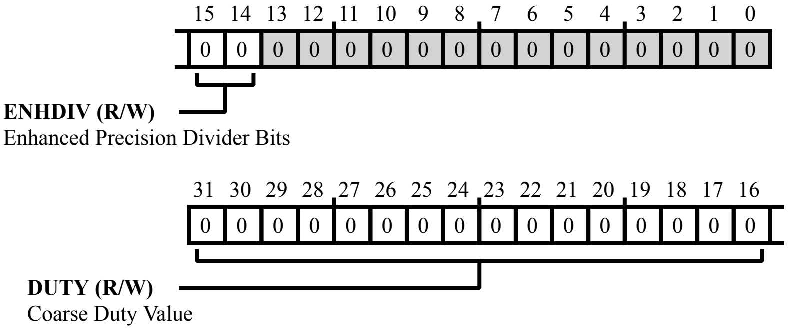

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-46: PWM_BH_DUTY0 Register Diagram

Table 19-24: PWM_BH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_BH_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_BH_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BH_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_BH_DUTY1` register contains the `PWM_BH_DUTY1.DUTY` bit field from the `PWM_BH1` register and the `PWM_BH_DUTY1.ENHDIV` bit field from the `PWM_BH1_HP` register.

Note that the `PWM_BH_DUTY1` register reads the `PWM_BH1` and the `PWM_BH1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

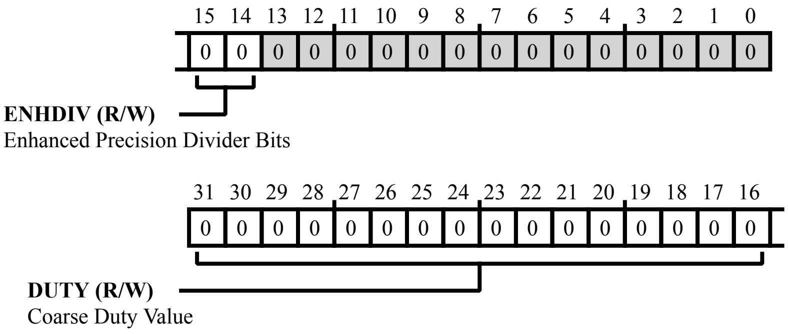

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-47: PWM_BH_DUTY1 Register Diagram

Table 19-25: PWM_BH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_BH_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_BH_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BH_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Duty-0 Register

The `PWM_BL0` and `PWM_BL1` registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_BCTL.PULSEMODELO` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_BL0` register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel B low pulse output for count less than `PWM_BL0` and deasserts this output for count greater than `PWM_BL1`.

The value range for the `PWM_BL0` and `PWM_BL1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHB_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_BL0` and `PWM_BL1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHB_DT` (two's complement) and +`PWM_TM0`/2 + `PWM_CHB_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHB_DT` (two's complement) and -`PWM_TM0`/2 - `PWM_CHB_DT`.

Note that using values in the `PWM_BL0` or `PWM_BL1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

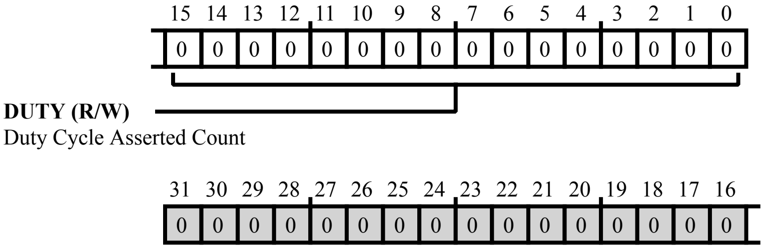


Figure 19-48: PWM_BL0 Register Diagram

Table 19-26: PWM_BL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_BL0.DUTY` bits select the duty cycle asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-0 Register

The `PWM_BL0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_BL0` register, allows programs to specify fractional duty cycles. The `PWM_BL0_HP` register and the `PWM_BL0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_BL0_HP` and the `PWM_BL0` registers are also present in the single full duty register (`PWM_BL_DUTY0`).

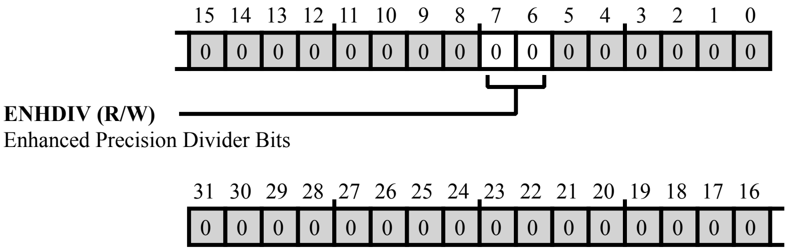

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-49: PWM_BL0_HP Register Diagram

Table 19-27: PWM_BL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BL0_HP.ENHDIV` bits provide fractional duty cycles for Channel B low side output.

Channel B-Low Duty-1 Register

The `PWM_BL0` and `PWM_BL1` registers determine the width for the low side output pulses. For more information, see the `PWM_BL0` register description.

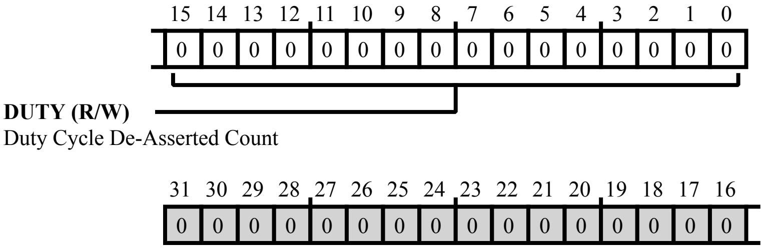

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-50: PWM_BL1 Register Diagram

Table 19-28: PWM_BL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_BL1.DUTY` bits select the duty cycle de-asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-1 Register

The `PWM_BL1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_BL1` register, allows programs to specify fractional duty cycles. The `PWM_BL1_HP` register and the `PWM_BL1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_BL1_HP` and the `PWM_BL1` registers are also present in the single full duty register (if available).

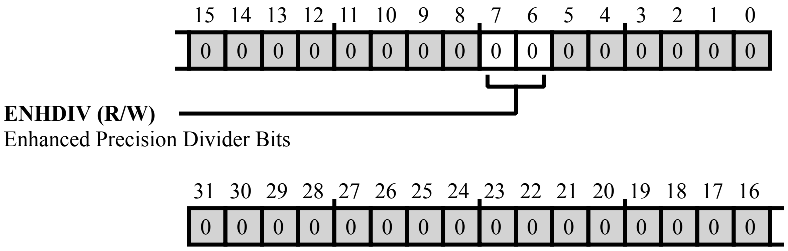

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-51: PWM_BL1_HP Register Diagram

Table 19-29: PWM_BL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BL1_HP.ENHDIV` bits provide fractional duty cycles for Channel B low side output.

Channel B-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_BL_DUTY0` register contains the `PWM_BL_DUTY0.DUTY` bit field from the `PWM_BL0` register and the `PWM_BL_DUTY0.ENHDIV` bit field from the `PWM_BL0_HP` register.

Note that the `PWM_BL_DUTY0` register reads the `PWM_BL0` and the `PWM_BL0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

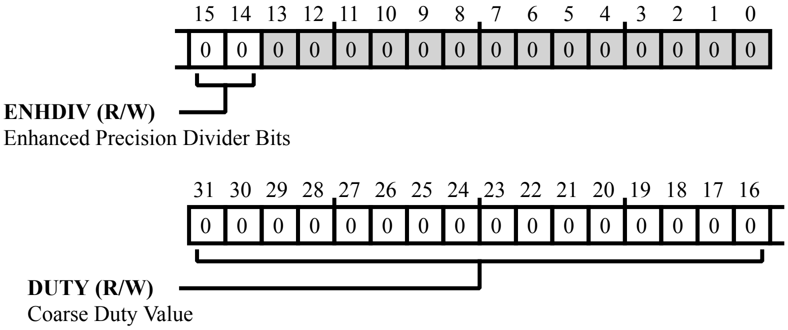

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-52: PWM_BL_DUTY0 Register Diagram

Table 19-30: PWM_BL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_BL_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_BL_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BL_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_BL_DUTY1` register contains the `PWM_BL_DUTY1.DUTY` bit field from the `PWM_BL1` register and the `PWM_BL_DUTY1.ENHDIV` bit field from the `PWM_BL1_HP` register.

Note that the `PWM_BL_DUTY1` register reads the `PWM_BL1` and the `PWM_BL1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

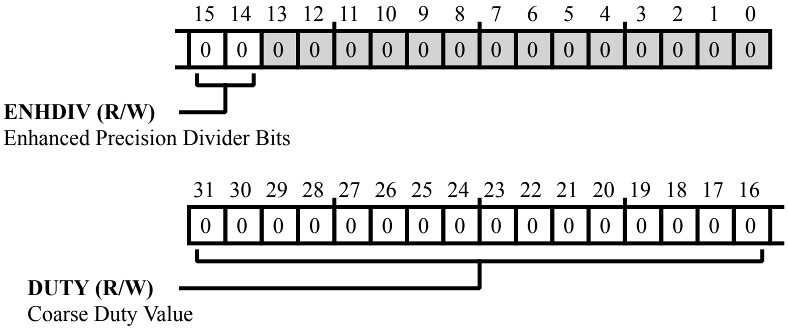

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-53: PWM_BL_DUTY1 Register Diagram

Table 19-31: PWM_BL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_BL_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_BL_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_BL_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C Control Register

The PWM_CCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

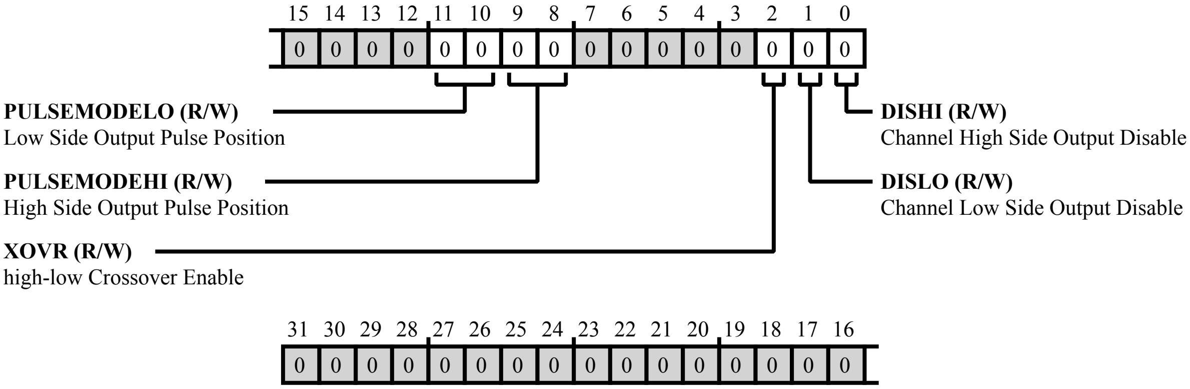

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PULSEMODELO (R/W)
Low Side Output Pulse Position
PULSEMODEHI (R/W)
High Side Output Pulse Position
XOVR (R/W)
high-low Crossover Enable
DISHI (R/W)
Channel High Side Output Disable
DISLO (R/W)
Channel Low Side Output Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-54: PWM_CCTL Register Diagram

Table 19-32: PWM_CCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_CCTL.PULSEMODELO bits select the pulse position for Channel C low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 19-32: PWM_CCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The `PWM_CCTL.PULSEMODEHI` bits select the pulse position for Channel C high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the `PWM_CH0` register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (`PWM_CH0` and `PWM_CH1`). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (`PWM_CH0` and `PWM_CH1`).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The `PWM_CCTL.XOVR` bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The `PWM_CCTL.DISLO` bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The `PWM_CCTL.DISHI` bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel C-High Pulse Duty Register 0

The `PWM_CH0` and `PWM_CH1` registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_CCTL.PULSEMODEHI` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_CH0` register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel C high pulse output for count less than `PWM_CH0` and deasserts this output for count greater than `PWM_CH1`.

The value range for the `PWM_CH0` and `PWM_CH1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHC_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_CH0` and `PWM_CH1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHC_DT` (two's complement) and +`PWM_TM0`/2 + `PWM_CHC_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHC_DT` (two's complement) and -`PWM_TM0`/2 - `PWM_CHC_DT`.

Note that using values in the `PWM_CH0` or `PWM_CH1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

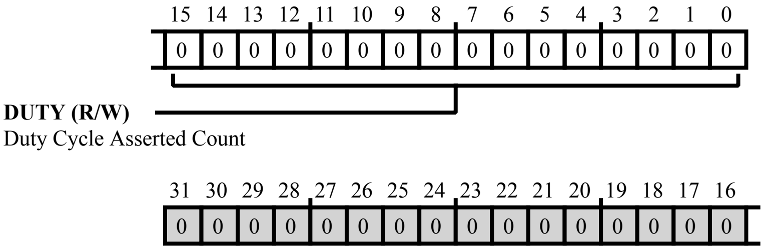


Figure 19-55: PWM_CH0 Register Diagram

Table 19-33: PWM_CH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_CH0.DUTY` bits select the duty cycle asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 0

The `PWM_CH0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_CH0` register, allows programs to specify fractional duty cycles. The `PWM_CH0_HP` register and the `PWM_CH0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_CH0_HP` and the `PWM_CH0` registers are also present in the single full duty register (if available).

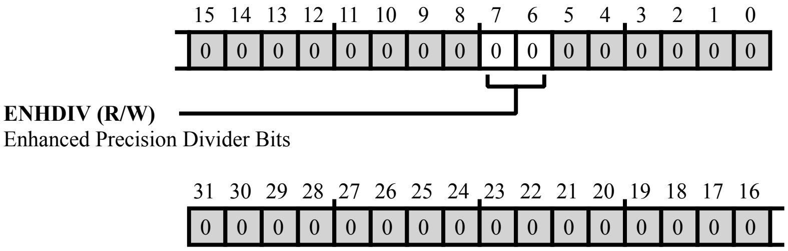

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-56: PWM_CH0_HP Register Diagram

Table 19-34: PWM_CH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CH0_HP.ENHDIV` bits provide fractional duty cycles for Channel C high side output.

Channel C-High Pulse Duty Register 1

The `PWM_CH0` and `PWM_CH1` registers determine the width for the high side output pulses. For more information, see the `PWM_CH0` register description.

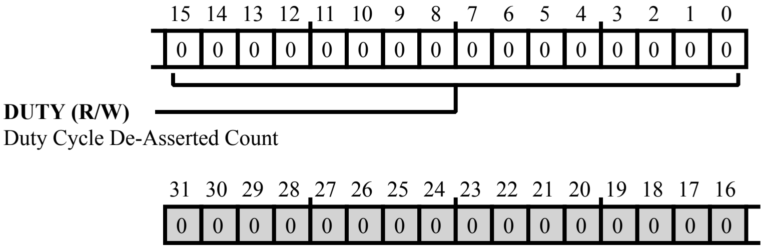

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-57: PWM_CH1 Register Diagram

Table 19-35: PWM_CH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_CH1.DUTY` bits select the duty cycle de-asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 1

The `PWM_CH1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_CH1` register, allows programs to specify fractional duty cycles. The `PWM_CH1_HP` register and the `PWM_CH1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_CH1_HP` and the `PWM_CH1` registers are also present in the single full duty register (if available).

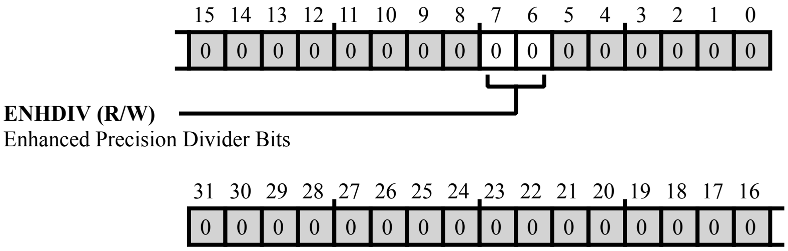

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-58: PWM_CH1_HP Register Diagram

Table 19-36: PWM_CH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CH1_HP.ENHDIV` bits provide fractional duty cycles for Channel C high side output.

Channel Configuration Register

The `PWM_CHANCFG` register configures Channel A, B, C, and D reference timer selection, high and low side output features, and enables high frequency chopping operation. Do not change the value of any bits in the PWM register while the PWM is enabled (`PWM_CTL.GLOBEN` =1).

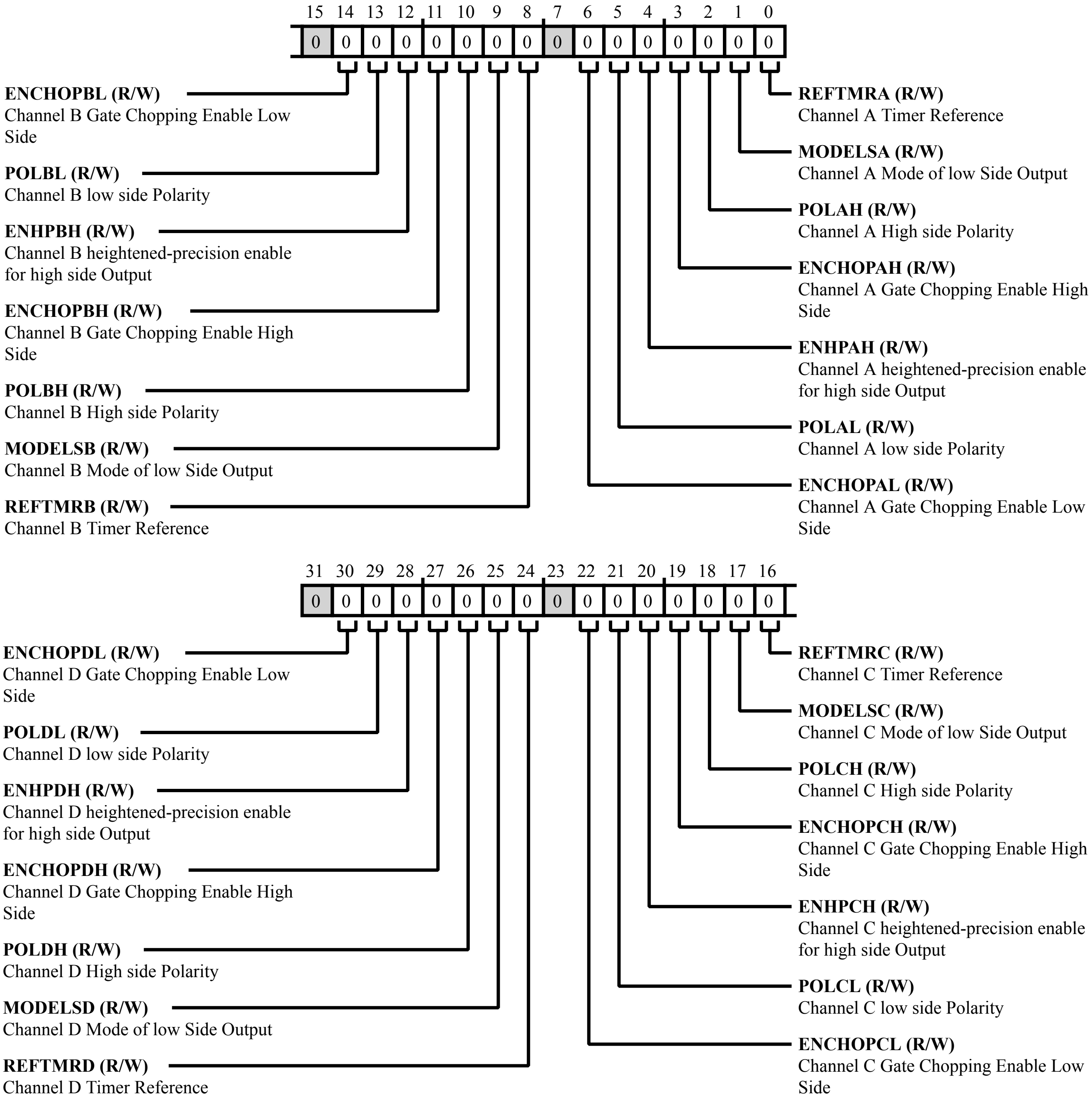

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENCHOPBL (R/W)
Channel B Gate Chopping Enable Low Side
POLBL (R/W)
Channel B low side Polarity
ENHPBH (R/W)
Channel B heightened-precision enable for high side Output
ENCHOPBH (R/W)
Channel B Gate Chopping Enable High Side
POLBH (R/W)
Channel B High side Polarity
MODELSB (R/W)
Channel B Mode of low Side Output
REFTMRB (R/W)
Channel B Timer Reference
REFTMRA (R/W)
Channel A Timer Reference
MODELSA (R/W)
Channel A Mode of low Side Output
POLAH (R/W)
Channel A High side Polarity
ENCHOPAH (R/W)
Channel A Gate Chopping Enable High Side
ENHPAH (R/W)
Channel A heightened-precision enable for high side Output
POLAL (R/W)
Channel A low side Polarity
ENCHOPAL (R/W)
Channel A Gate Chopping Enable Low Side
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENCHOPDL (R/W)
Channel D Gate Chopping Enable Low Side
POLDL (R/W)
Channel D low side Polarity
ENHPDH (R/W)
Channel D heightened-precision enable for high side Output
ENCHOPDH (R/W)
Channel D Gate Chopping Enable High Side
POLDH (R/W)
Channel D High side Polarity
MODELSD (R/W)
Channel D Mode of low Side Output
REFTMRD (R/W)
Channel D Timer Reference
REFTMRC (R/W)
Channel C Timer Reference
MODELSC (R/W)
Channel C Mode of low Side Output
POLCH (R/W)
Channel C High side Polarity
ENCHOPCH (R/W)
Channel C Gate Chopping Enable High Side
ENHPCH (R/W)
Channel C heightened-precision enable for high side Output
POLCL (R/W)
Channel C low side Polarity
ENCHOPCL (R/W)
Channel C Gate Chopping Enable Low Side

Figure 19-59: PWM_CHANCFG Register Diagram

Table 19-37: PWM_CHANCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	ENCHOPDL	Channel D Gate Chopping Enable Low Side. The `PWM_CHANCFG.ENCHOPDL` bit enables mixing of the Channel D low side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel D Low Side
		1	Enable Chopping Channel D Low Side
29 (R/W)	POLDL	Channel D low side Polarity. The `PWM_CHANCFG.POLDL` bit selects the Channel D low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
28 (R/W)	ENHPDH	Channel D heightened-precision enable for high side Output. The `PWM_CHANCFG.ENHPDH` bit enables heightened-precision Channel D high side output.	
		0	Disable HP Output Channel D High
		1	Enable HP Output Channel D High
27 (R/W)	ENCHOPDH	Channel D Gate Chopping Enable High Side. The `PWM_CHANCFG.ENCHOPDH` bit enables mixing of the Channel D high side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel D High Side
		1	Enable Chopping Channel D High Side
26 (R/W)	POLDH	Channel D High side Polarity. The `PWM_CHANCFG.POLDH` bit selects the Channel D high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 19-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODELSD	Channel D Mode of low Side Output. The `PWM_CHANCFG.MODELSD` bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When `PWM_CHANCFG.MODELSD` =0, the low side output is an inverted form of the high side output, which is generated using the `PWM_BH0` and `PWM_BH1` registers for pulse width, using the `PWM_BCTL.PULSEMODEHI` bits for pulse positioning, and `PWM_CHANCFG.POLBH` bits for polarity.	
		0	Invert of high output
		1	Independent control
24 (R/W)	REFTMRD	Channel D Timer Reference. The `PWM_CHANCFG.REFTMRD` bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel D operation.	
		0	PWMTMR0 is Channel D reference
		1	PWMTMR1 is Channel D reference
22 (R/W)	ENCHOPCL	Channel C Gate Chopping Enable Low Side. The `PWM_CHANCFG.ENCHOPCL` bit enables mixing of the Channel C low side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel C Low Side
		1	Enable Chopping Channel C Low Side
21 (R/W)	POLCL	Channel C low side Polarity. The `PWM_CHANCFG.POLCL` bit selects the Channel C low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
20 (R/W)	ENHPCH	Channel C heightened-precision enable for high side Output. The `PWM_CHANCFG.ENHPCH` bit enables heightened-precision Channel C high side output.	
		0	Disable HP Output Channel C High
		1	Enable HP Output Channel C High

Table 19-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	ENCHOPCH	Channel C Gate Chopping Enable High Side. The `PWM_CHANCFG.ENCHOPCH` bit enables mixing of the Channel C high side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel C High Side
		1	Enable Chopping Channel C High Side
18 (R/W)	POLCH	Channel C High side Polarity. The `PWM_CHANCFG.POLCH` bit selects the Channel C high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
17 (R/W)	MODELSC	Channel C Mode of low Side Output. The `PWM_CHANCFG.MODELSC` bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When `PWM_CHANCFG.MODELSC` =0, the low side output is an inverted form of the high side output, which is generated using the `PWM_BH0` and `PWM_BH1` registers for pulse width, using the `PWM_BCTL.PULSEMODEHI` bits for pulse positioning, and `PWM_CHANCFG.POLBH` bits for polarity.	
		0	Invert of high output
		1	Independent control
16 (R/W)	REFTMRC	Channel C Timer Reference. The `PWM_CHANCFG.REFTMRC` bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel C operation.	
		0	PWMTMR0 is Channel C reference
		1	PWMTMR1 is Channel C reference
14 (R/W)	ENCHOPBL	Channel B Gate Chopping Enable Low Side. The `PWM_CHANCFG.ENCHOPBL` bit enables mixing of the Channel B low side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel B Low Side
		1	Enable Chopping Channel B Low Side

Table 19-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	POLBL	Channel B low side Polarity. The `PWM_CHANCFG.POLBL` bit selects the Channel B low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
12 (R/W)	ENHPBH	Channel B heightened-precision enable for high side Output. The `PWM_CHANCFG.ENHPBH` bit enables heightened-precision Channel B high side output.	
		0	Disable HP Output Channel B High
		1	Enable HP Output Channel B High
11 (R/W)	ENCHOPBH	Channel B Gate Chopping Enable High Side. The `PWM_CHANCFG.ENCHOPBH` bit enables mixing of the Channel B high side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel B High Side
		1	Enable Chopping Channel B High Side
10 (R/W)	POLBH	Channel B High side Polarity. The `PWM_CHANCFG.POLBH` bit selects the Channel B high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
9 (R/W)	MODELSB	Channel B Mode of low Side Output. The `PWM_CHANCFG.MODELSB` bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When `PWM_CHANCFG.MODELSB` =0, the low side output is an inverted form of the high side output, which is generated using the `PWM_BH0` and `PWM_BH1` registers for pulse width, using the `PWM_BCTL.PULSEMODEHI` bits for pulse positioning, and `PWM_CHANCFG.POLBH` bits for polarity.	
		0	Invert of high output
		1	Independent control
8 (R/W)	REFTMRB	Channel B Timer Reference. The `PWM_CHANCFG.REFTMRB` bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel B operation.	
		0	PWMTMR0 is Channel B reference
		1	PWMTMR1 is Channel B reference

Table 19-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	ENCHOPAL	Channel A Gate Chopping Enable Low Side. The `PWM_CHANCFG.ENCHOPAL` bit enables mixing of the Channel A low side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel A Low Side
		1	Enable Chopping Channel A Low Side
5 (R/W)	POLAL	Channel A low side Polarity. The `PWM_CHANCFG.POLAL` bit selects the Channel A low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
4 (R/W)	ENHPAH	Channel A heightened-precision enable for high side Output. The `PWM_CHANCFG.ENHPAH` bit enables heightened-precision Channel A high side output.	
		0	Disable HP Output Channel A High
		1	Enable HP Output Channel A High
3 (R/W)	ENCHOPAH	Channel A Gate Chopping Enable High Side. The `PWM_CHANCFG.ENCHOPAH` bit enables mixing of the Channel A high side output signals with a high-frequency chopping signal, which is configured with the `PWM_CHOPCFG` register.	
		0	Disable Chopping Channel A High Side
		1	Enable Chopping Channel A High Side
2 (R/W)	POLAH	Channel A High side Polarity. The `PWM_CHANCFG.POLAH` bit selects the Channel A high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 19-37: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODELSA	Channel A Mode of low Side Output. The `PWM_CHANCFG.MODELSA` bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When `PWM_CHANCFG.MODELSA` =0, the low side output is an inverted form of the high side output, which is generated using the `PWM_AH0` and `PWM_AH1` registers for pulse width, using the `PWM_ACTL.PULSEMODEHI` bits for pulse positioning, and `PWM_CHANCFG.POLAH` bits for polarity.	
		0	Invert of high output
		1	Independent control
0 (R/W)	REFTMRA	Channel A Timer Reference. The `PWM_CHANCFG.REFTMRA` bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel A operation.	
		0	PWMTMR0 is Channel A reference
		1	PWMTMR1 is Channel A reference

Channel A Dead-time Register

The `PWM_CHA_DT` register controls the value of dead-time for channel A independently.

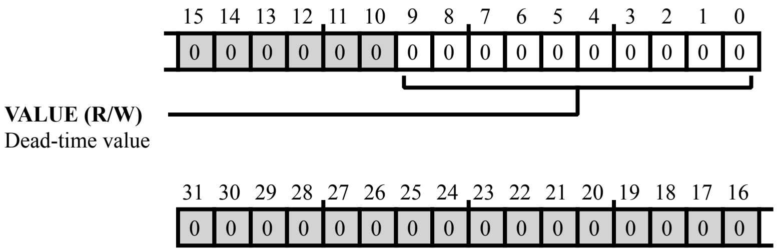

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Dead-time value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-60: PWM_CHA_DT Register Diagram

Table 19-38: PWM_CHA_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value.

Channel B Dead-time Register

The `PWM_CHB_DT` register controls the value of dead-time for channel B independently.

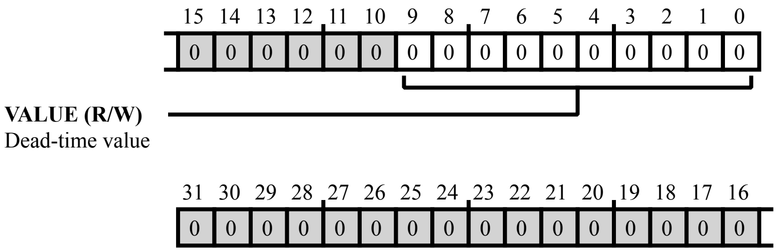

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Dead-time value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-61: PWM_CHB_DT Register Diagram

Table 19-39: PWM_CHB_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The `PWM_CHB_DT.VALUE` bit field contains the dead-time value.

Channel C Dead-time Register

The `PWM_CHC_DT` register controls the value of dead-time for channel C independently.

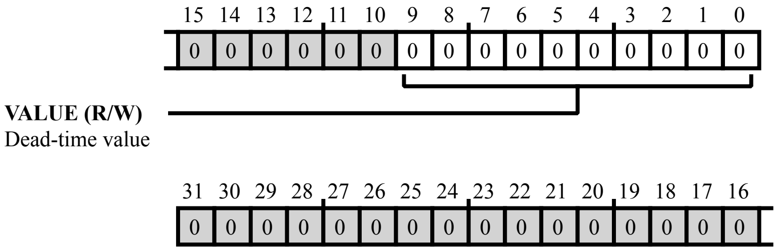

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Dead-time value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-62: PWM_CHC_DT Register Diagram

Table 19-40: PWM_CHC_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The `PWM_CHC_DT.VALUE` bit field contains the dead-time value.

Channel D Dead-time Register

The `PWM_CHD_DT` register controls the value of dead-time for channel D independently.

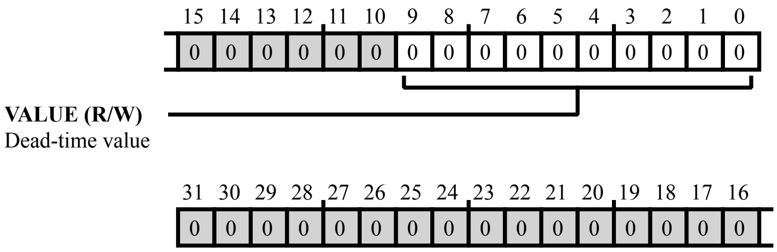

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Dead-time value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-63: PWM_CHD_DT Register Diagram

Table 19-41: PWM_CHD_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The `PWM_CHD_DT.VALUE` bit field contains the dead-time value.

Chop Configuration Register

The `PWM_CHOPCFG` register holds a divisor value that controls the chopping frequency. The PWM permits a mixing of the output signals with a high-frequency chopping signal to aid with interfacing to pulse transformers. Also note that high-frequency chopping may be independently enabled for each channel's high-side and the low-side outputs using channel control bits. (For example, control chopping for Channel A with the `PWM_CHANCFG.ENCHOPAH` and `PWM_CHANCFG.ENCHOPAH` bits.)

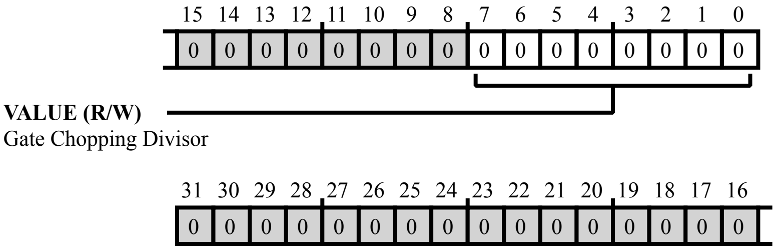

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Gate Chopping Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-64: PWM_CHOPCFG Register Diagram

Table 19-42: PWM_CHOPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Gate Chopping Divisor. The `PWM_CHOPCFG.VALUE` bits provide the high frequency chopping divisor. When the divisor value is changed, the new period takes effect from the next edge of the chopping signal. The `PWM_CHOPCFG.VALUE` value may be calculated using either of the following formulas: CHOPDIV = $[(T_{CHOP}/T_{CK}) / 4] - 1$ CHOPDIV = $[(f_{CK} / f_{CHOP}) / 4] - 1$

Channel C-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_CH_DUTY0` register contains the `PWM_CH_DUTY0.DUTY` bit field from the `PWM_CH0` register and the `PWM_CH_DUTY0.ENHDIV` bit field from the `PWM_CH0_HP` register.

Note that the `PWM_CH_DUTY0` register reads the `PWM_CH0` and the `PWM_CH0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

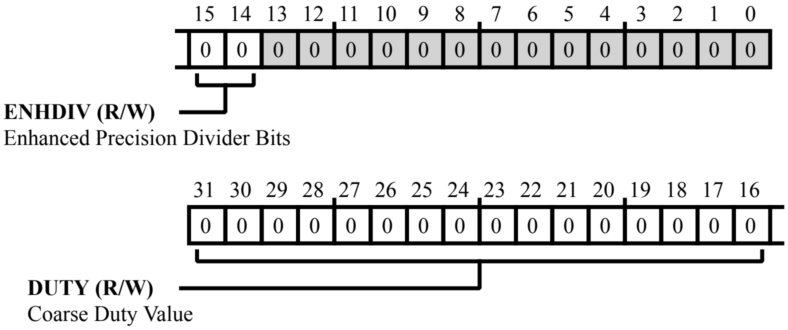

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-65: PWM_CH_DUTY0 Register Diagram

Table 19-43: PWM_CH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_CH_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_CH_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CH_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CH_DUTY1 register contains the PWM_CH_DUTY1.DUTY bit field from the PWM_CH1 register and the PWM_CH_DUTY1.ENHDIV bit field from the PWM_CH1_HP register.

Note that the PWM_CH_DUTY1 register reads the PWM_CH1 and the PWM_CH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

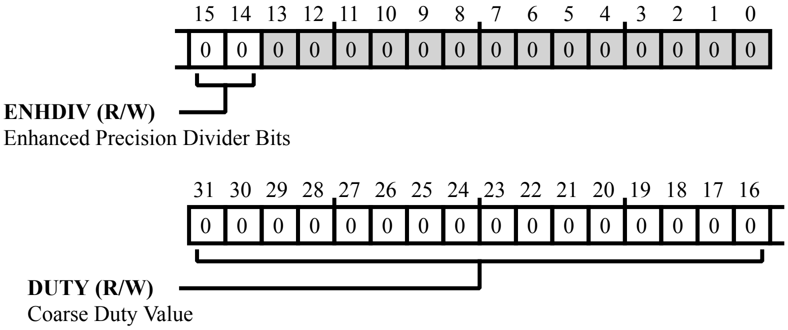

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-66: PWM_CH_DUTY1 Register Diagram

Table 19-44: PWM_CH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Pulse Duty Register 0

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CL0 register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel C low pulse output for count less than PWM_CL0 and deasserts this output for count greater than PWM_CL1.

The value range for the PWM_CL0 and PWM_CL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHC_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CL0 and PWM_CL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHC_DT (two's complement) and +PWM_TM0/2 + PWM_CHC_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHC_DT (two's complement) to -PWM_TM0/2 - PWM_CHC_DT.

Note that using values in the PWM_CL0 or PWM_CL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

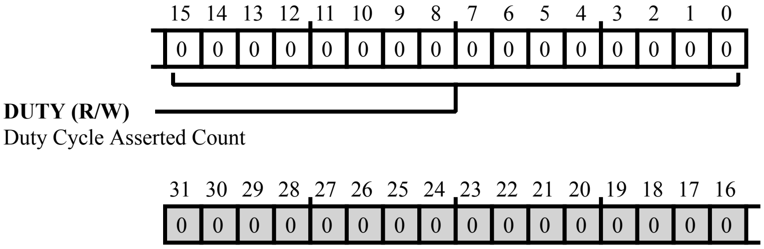


Figure 19-67: PWM_CL0 Register Diagram

Table 19-45: PWM_CL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CL0.DUTY bits select the duty cycle asserted count for Channel C low side output.

Channel C-Low Pulse Duty Register 1

The `PWM_CL0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_CL0` register, allows programs to specify fractional duty cycles. The `PWM_CL0_HP` register and the `PWM_CL0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_CL0_HP` and the `PWM_CL0` registers are also present in the single full duty register (if available).

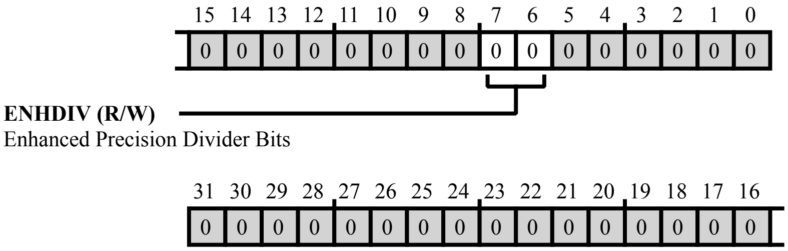

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-68: PWM_CL0_HP Register Diagram

Table 19-46: PWM_CL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CL0_HP.ENHDIV` bits provide fractional duty cycles for Channel C low side output.

Channel C-Low Duty-1 Register

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. For more information, see the PWM_CL0 register description.

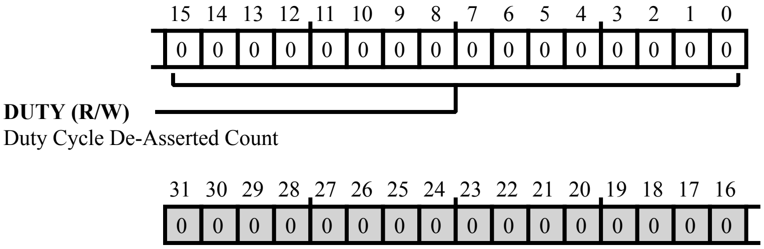

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-69: PWM_CL1 Register Diagram

Table 19-47: PWM_CL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel C-Low Heightened-Precision Duty-1 Register

The `PWM_CL1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_CL1` register, allows programs to specify fractional duty cycles. The `PWM_CL1_HP` register and the `PWM_CL1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_CL1_HP` and the `PWM_CL1` registers are also present in the single full duty register (if available).

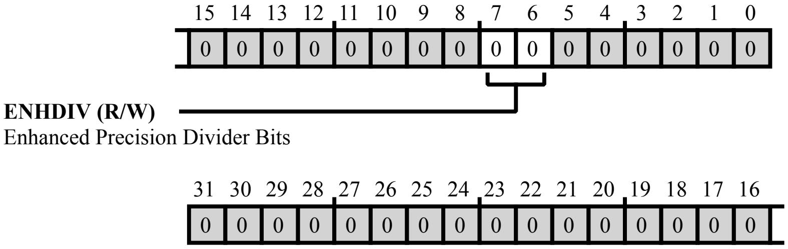

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-70: PWM_CL1_HP Register Diagram

Table 19-48: PWM_CL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CL1_HP.ENHDIV` bits provide fractional duty cycles for Channel C low side output.

Channel C-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_CL_DUTY0` register contains the `PWM_CL_DUTY0.DUTY` bit field from the `PWM_CL0` register and the `PWM_CL_DUTY0.ENHDIV` bit field from the `PWM_CL0_HP` register.

Note that the `PWM_CL_DUTY0` register reads the `PWM_CL0` and the `PWM_CL0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

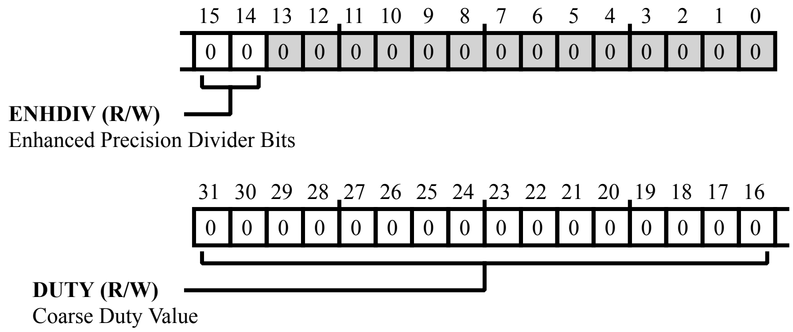

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-71: PWM_CL_DUTY0 Register Diagram

Table 19-49: PWM_CL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_CL_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_CL_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CL_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_CL_DUTY1` register contains the `PWM_CL_DUTY1.DUTY` bit field from the `PWM_CL1` register and the `PWM_CL_DUTY1.ENHDIV` bit field from the `PWM_CL1_HP` register.

Note that the `PWM_CL_DUTY1` register reads the `PWM_CL1` and the `PWM_CL1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

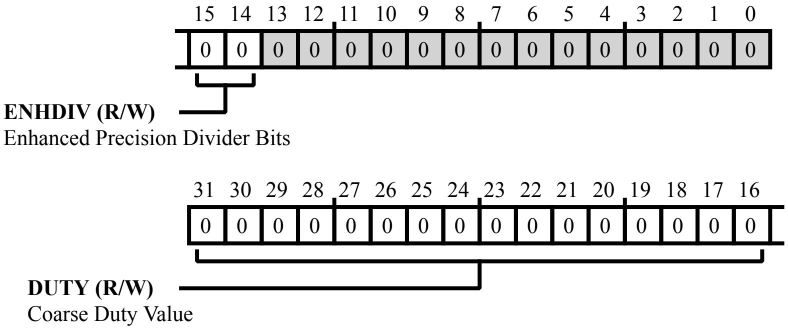

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-72: PWM_CL_DUTY1 Register Diagram

Table 19-50: PWM_CL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_CL_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_CL_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_CL_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Control Register

The `PWM_CTL` register enables the PWM, enables delay counters for the channels, and configures sync features. This register also provides support for tripping a PWM fault condition through software.

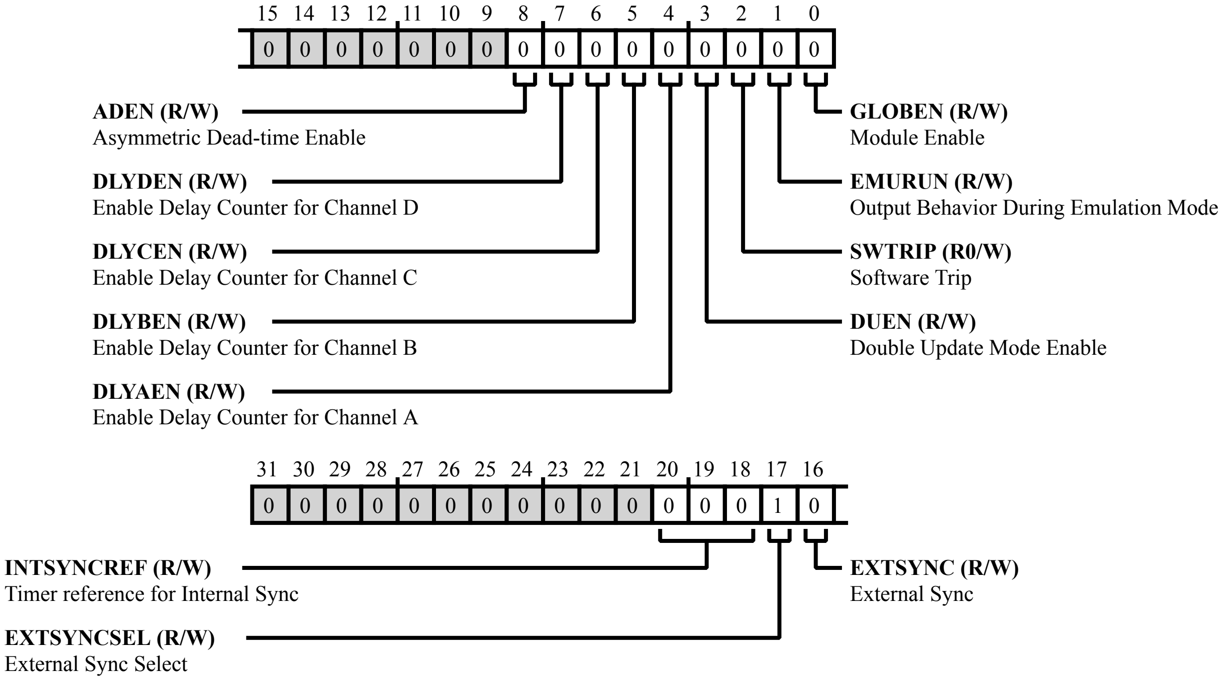

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADEN (R/W)
Asymmetric Dead-time Enable
DLYDEN (R/W)
Enable Delay Counter for Channel D
DLYCEN (R/W)
Enable Delay Counter for Channel C
DLYBEN (R/W)
Enable Delay Counter for Channel B
DLYAEN (R/W)
Enable Delay Counter for Channel A
GLOBEN (R/W)
Module Enable
EMURUN (R/W)
Output Behavior During Emulation Mode
SWTRIP (R0/W)
Software Trip
DUEN (R/W)
Double Update Mode Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
INTSYNCREF (R/W)
Timer reference for Internal Sync
EXTSYNCSEL (R/W)
External Sync Select
EXTSYNC (R/W)
External Sync

Figure 19-73: PWM_CTL Register Diagram

Table 19-51: PWM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:18 (R/W)	INTSYNCREF	Timer reference for Internal Sync. The `PWM_CTL.INTSYNCREF` bits select the timer reference for the internal sync. Note that all other combinations reserved.	
		0	PWMTMR0 provides sync reference
		1	PWMTMR1 provides sync reference
		2	PWMTMR2 provides sync reference
		3	PWMTMR3 provides sync reference
		4	PWMTMR4 provides sync reference

Table 19-51: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	EXTSYNCSEL	External Sync Select. The `PWM_CTL.EXTSYNCSEL` bit selects whether the external sync signal is synchronous or asynchronous. Note that latency in PWM sync response differs between asynchronous and synchronous external sync modes. For more information, see the PWM functional description.	
		0	Asynchronous External Sync
		1	Synchronous External Sync
16 (R/W)	EXTSYNC	External Sync. The `PWM_CTL.EXTSYNC` bit selects whether the PWM uses an external or internal sync signal for the main timer (PWMTMR0). Do not change the value of the `PWM_CTL.EXTSYNC` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).	
		0	Internal sync used
		1	External sync used
8 (R/W)	ADEN	Asymmetric Dead-time Enable. When symmetric dead-time is enabled (`PWM_CTL.ADEN` = 0), in the dependent mode, both the high-side and low-side outputs are reduced by DT cycles on both the assertion and de-assertion edges. When symmetric dead-time is disabled (`PWM_CTL.ADEN` = 1), the falling edges of both high and low-side outputs occur at the programmed duty value, but their rise-times are delayed by 2 x DT.	
		0	Dead-time is symmetric
		1	Dead-time is asymmetric
7 (R/W)	DLYDEN	Enable Delay Counter for Channel D. The `PWM_CTL.DLYDEN` bit enables the Channel D delay counter, supporting phase-offset control. Do not change the value of the `PWM_CTL.DLYDEN` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).	
		0	Disable
		1	Enable
6 (R/W)	DLYCEN	Enable Delay Counter for Channel C. The `PWM_CTL.DLYCEN` bit enables the Channel C delay counter, supporting phase-offset control. Do not change the value of the `PWM_CTL.DLYCEN` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).	
		0	Disable
		1	Enable

Table 19-51: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DLYBEN	Enable Delay Counter for Channel B. The `PWM_CTL.DLYBEN` bit enables the Channel B delay counter, supporting phase-offset control. Do not change the value of the `PWM_CTL.DLYBEN` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).	
		0	Disable
		1	Enable
4 (R/W)	DLYAEN	Enable Delay Counter for Channel A. The `PWM_CTL.DLYAEN` bit enables the Channel A delay counter, supporting phase-offset control. Do not change the value of the `PWM_CTL.DLYAEN` bit while the PWM is enabled (`PWM_CTL.GLOBEN` =1).	
		0	Disable
		1	Enable
3 (R/W)	DUEN	Double Update Mode Enable. In Single Update Mode, double buffering of all registers happens at the period boundary of the PWM timer. In Double Update Mode, double buffering of all registers (except delay counter registers) happens at the middle of the period as well as the beginning of the period.	
		0	Single Update Mode
		1	Double Update Mode
2 (R0/W)	SWTRIP	Software Trip. The `PWM_CTL.SWTRIP` bit permits tripping a fault condition through software, shutting down PWM output. This bit always read as 0. If the `PWM_CTL.SWTRIP` bit and `PWM_CTL.GLOBEN` bit are set in the same write, the write does not trip the fault condition.	
		0	Don't force fault
		1	Force a Fault Trip Condition
1 (R/W)	EMURUN	Output Behavior During Emulation Mode. The `PWM_CTL.EMURUN` bit selects PWM output behavior during emulation mode.	
		0	Disable Outputs
		1	Enable Outputs

Table 19-51: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	GLOBEN	Module Enable. The `PWM_CTL.GLOBEN` bit enables the PWM, enabling all timers and outputs. While this bit is enabled, processor code should not change the value of the `PWM_CTL.DLYAEN` bit, `PWM_CTL.DLYBEN` bit, `PWM_CTL.DLYCEN` bit, `PWM_CTL.DLYDEN` bit, `PWM_CTL.EXTSYNCSEL` bit, or any bits in the `PWM_CHANCFG` register. Note that there is a latency between PWM disable and the cessation of output waveforms. There is also a latency between PWM enable and start of output waveforms. For the latency description, see the PWM functional description.	
		0	Disable
		1	Enable

Channel D Control Register

The PWM_DCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

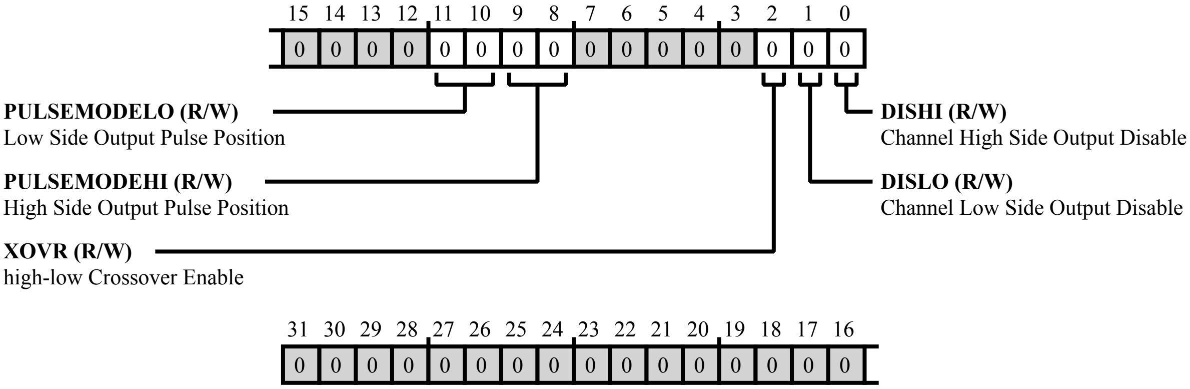

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PULSEMODELO (R/W)
Low Side Output Pulse Position
PULSEMODEHI (R/W)
High Side Output Pulse Position
XOVR (R/W)
high-low Crossover Enable
DISHI (R/W)
Channel High Side Output Disable
DISLO (R/W)
Channel Low Side Output Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-74: PWM_DCTL Register Diagram

Table 19-52: PWM_DCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_DCTL.PULSEMODELO bits select the pulse position for Channel D low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 19-52: PWM_DCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The `PWM_DCTL.PULSEMODEHI` bits select the pulse position for Channel D high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the `PWM_DH0` register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (`PWM_DH0` and `PWM_DH1`). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (`PWM_DH0` and `PWM_DH1`).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The `PWM_DCTL.XOVR` bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The `PWM_DCTL.DISLO` bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The `PWM_DCTL.DISHI` bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel D-High Duty-0 Register

The `PWM_DH0` and `PWM_DH1` registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_DCTL.PULSEMODEHI` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_DH0` register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel D high pulse output for count less than `PWM_DH0` and deasserts this output for count greater than `PWM_DH1`.

The value range for the `PWM_DH0` and `PWM_DH1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHD_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_DH0` and `PWM_DH1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHD_DT` (two's complement) to +`PWM_TM0`/2 + `PWM_CHD_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHD_DT` (two's complement) to -`PWM_TM0`/2 - `PWM_CHD_DT`.

Note that using values in the `PWM_DH0` or `PWM_DH1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

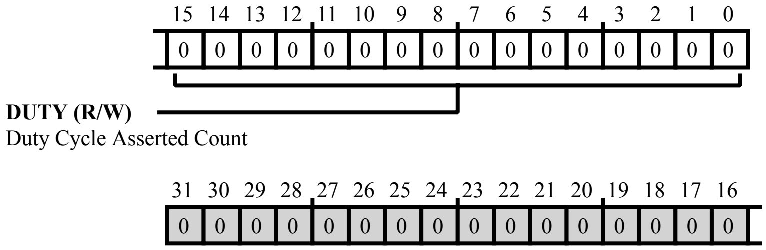


Figure 19-75: PWM_DH0 Register Diagram

Table 19-53: PWM_DH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_DH0.DUTY` bits select the duty cycle asserted count for Channel D high side output.

Channel D-High Pulse Heightened-Precision Duty Register 0

The `PWM_DH0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_DH0` register, allows programs to specify fractional duty cycles. The `PWM_DH0_HP` register and the `PWM_DH0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_DH0_HP` and the `PWM_DH0` registers are also present in the single full duty register (if available).

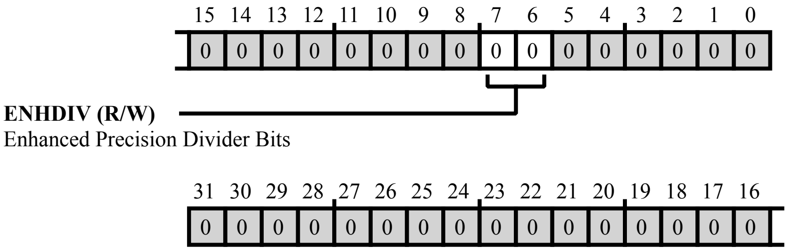

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-76: PWM_DH0_HP Register Diagram

Table 19-54: PWM_DH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DH0_HP.ENHDIV` bits provide fractional duty cycles for Channel D high side output.

Channel D-High Pulse Duty Register 1

The `PWM_DH0` and `PWM_DH1` registers determine the width for the high side output pulses. For more information, see the `PWM_DH0` register description.

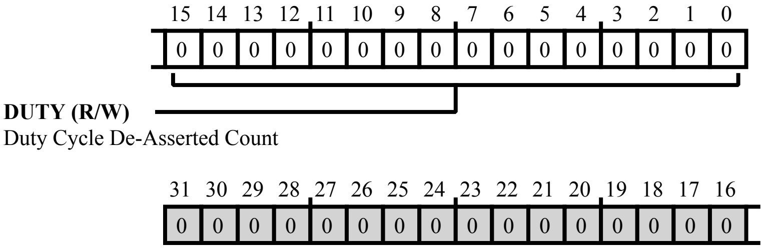

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-77: PWM_DH1 Register Diagram

Table 19-55: PWM_DH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_DH1.DUTY` bits select the duty cycle de-asserted count for Channel D high side output.

Channel D High Pulse Heightened-Precision Duty Register 1

The `PWM_DH1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_DH1` register, allows programs to specify fractional duty cycles. The `PWM_DH1_HP` register and the `PWM_DH1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_DH1_HP` and the `PWM_DH1` registers are also present in the single full duty register (if available).

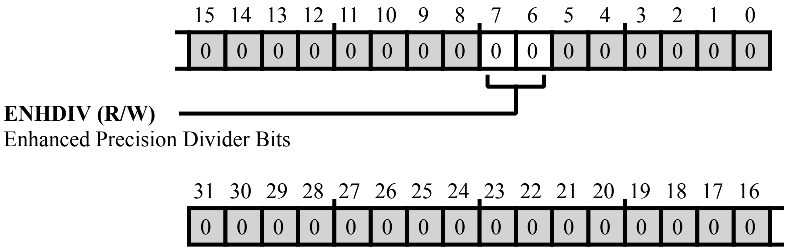

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-78: PWM_DH1_HP Register Diagram

Table 19-56: PWM_DH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DH1_HP.ENHDIV` bits provide fractional duty cycles for Channel D high side output.

Channel D-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_DH_DUTY0` register contains the `PWM_DH_DUTY0.DUTY` bit field from the `PWM_DH0` register and the `PWM_DH_DUTY0.ENHDIV` bit field from the `PWM_DH0_HP` register.

Note that the `PWM_DH_DUTY0` register reads the `PWM_DH0` and the `PWM_DH0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

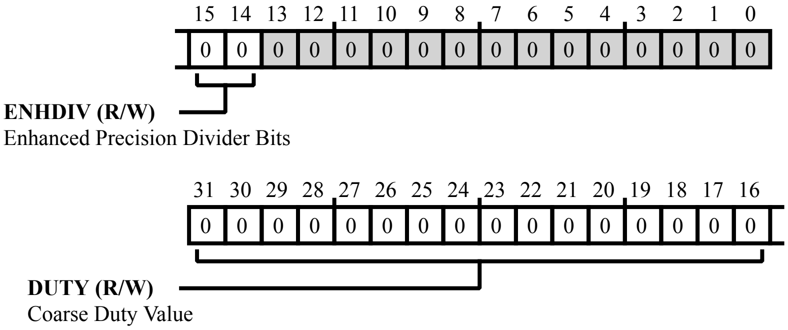

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-79: PWM_DH_DUTY0 Register Diagram

Table 19-57: PWM_DH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_DH_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_DH_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DH_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_DH_DUTY1` register contains the `PWM_DH_DUTY1.DUTY` bit field from the `PWM_DH1` register and the `PWM_DH_DUTY1.ENHDIV` bit field from the `PWM_DH1_HP` register.

Note that the `PWM_DH_DUTY1` register reads the `PWM_DH1` and the `PWM_DH1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

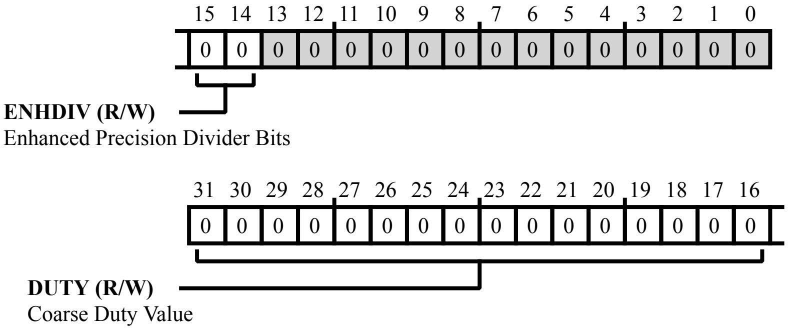

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-80: PWM_DH_DUTY1 Register Diagram

Table 19-58: PWM_DH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_DH_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_DH_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DH_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Pulse Duty Register 0

The `PWM_DL0` and `PWM_DL1` registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the `PWM_DCTL.PULSEMODELO` bits. When the pulse mode is symmetrical, the PWM uses the value in the `PWM_DL0` register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel D low pulse output for count less than `PWM_DL0` and deasserts this output for count greater than `PWM_DL1`.

The value range for the `PWM_DL0` and `PWM_DL1` registers depends on the period of the timer being used by the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 (two's complement) and +`PWM_TM0`/2, when dead time (`PWM_CHD_DT`) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for `PWM_DL0` and `PWM_DL1` depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if `PWM_TM0` is used, the duty cycle values may be between -`PWM_TM0`/2 + `PWM_CHD_DT` (two's complement) and +`PWM_TM0`/2 + `PWM_CHD_DT`.

When dead time is considered for left half or right half pulse modes, if `PWM_TM0` is used, the duty cycle values may be between `PWM_TM0`/2 + `PWM_CHD_DT` (two's complement) and -`PWM_TM0`/2 - `PWM_CHD_DT`.

Note that using values in the `PWM_DL0` or `PWM_DL1` registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

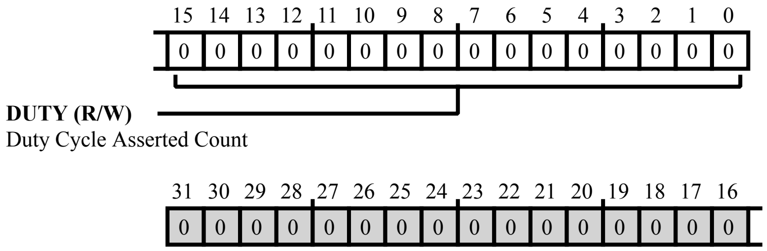


Figure 19-81: PWM_DL0 Register Diagram

Table 19-59: PWM_DL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The `PWM_DL0.DUTY` bits select the duty cycle asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-0 Register

The `PWM_DL0_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_DL0` register, allows programs to specify fractional duty cycles. The `PWM_DL0_HP` register and the `PWM_DL0` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_DL0_HP` and the `PWM_DL0` registers are also present in the single full duty register (if available).

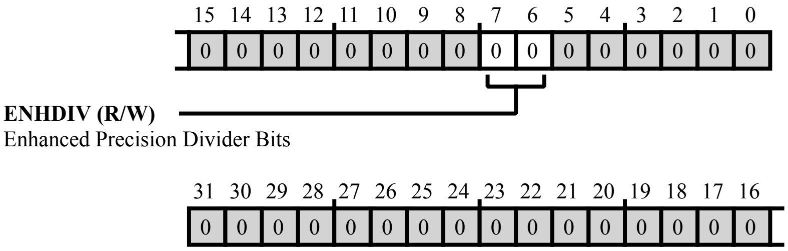

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-82: PWM_DL0_HP Register Diagram

Table 19-60: PWM_DL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DL0_HP.ENHDIV` bits provide fractional duty cycles for Channel D low side output.

Channel D-Low Pulse Duty Register 1

The `PWM_DL0` and `PWM_DL1` registers determine the width for the low side output pulses. For more information, see the `PWM_DL0` register description.

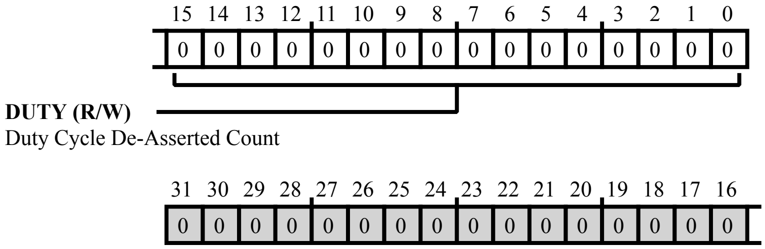

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Duty Cycle De-Asserted Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-83: PWM_DL1 Register Diagram

Table 19-61: PWM_DL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The `PWM_DL1.DUTY` bits select the duty cycle de-asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-1 Register

The `PWM_DL1_HP` register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the `PWM_DL1` register, allows programs to specify fractional duty cycles. The `PWM_DL1_HP` register and the `PWM_DL1` register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the `PWM_DL1_HP` and the `PWM_DL1` registers are also present in the single full duty register (if available).

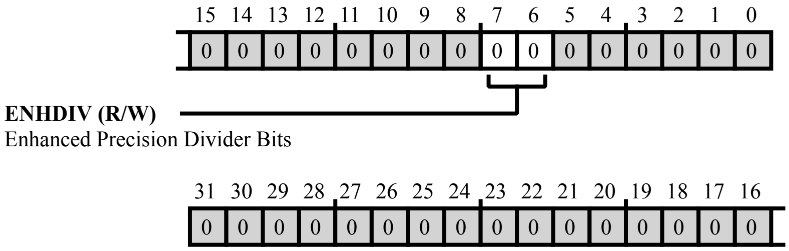

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-84: PWM_DL1_HP Register Diagram

Table 19-62: PWM_DL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DL1_HP.ENHDIV` bits provide fractional duty cycles for Channel D low side output.

Channel A Delay Register

The `PWM_DLYA` register controls a delay for the Channel A timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (`PWM_CTL.DLYAEN` =1). For more information about applying the delay, see the PWM Functional Description section. Note that the `PWM_DLYA` delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, `PWM_DLYA` must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

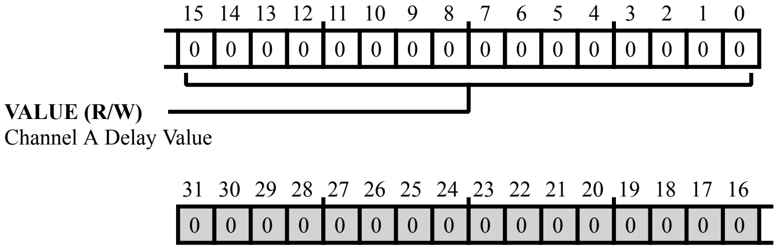

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Channel A Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-85: PWM_DLYA Register Diagram

Table 19-63: PWM_DLYA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel A Delay Value. The `PWM_DLYA.VALUE` bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel A.

Channel B Delay Register

The `PWM_DLYB` register controls a delay for the Channel B timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (`PWM_CTL.DLYBEN` =1). For more information about applying the delay, see the PWM Functional Description section. Note that the `PWM_DLYB` delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, `PWM_DLYB` must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

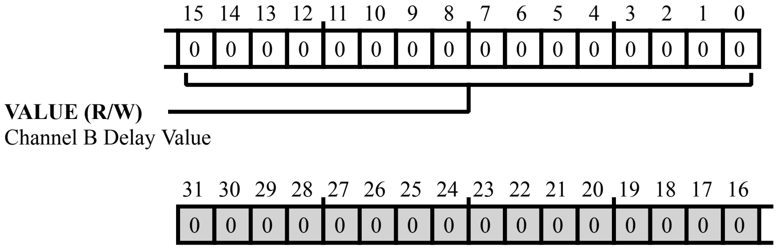

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Channel B Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-86: PWM_DLYB Register Diagram

Table 19-64: PWM_DLYB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel B Delay Value. The `PWM_DLYB.VALUE` bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel B.

Channel C Delay Register

The `PWM_DLYC` register controls a delay for the Channel C timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (`PWM_CTL.DLYCEN` =1). For more information about applying the delay, see the PWM Functional Description section. Note that the `PWM_DLYC` delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, `PWM_DLYC` must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

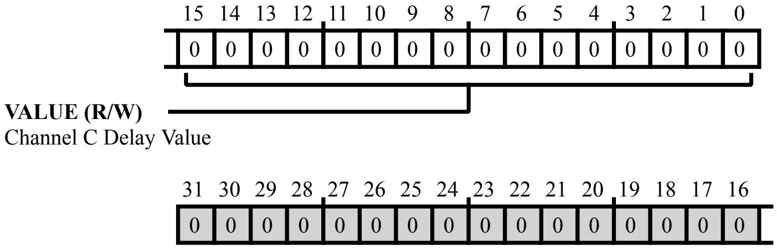

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Channel C Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-87: PWM_DLYC Register Diagram

Table 19-65: PWM_DLYC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel C Delay Value. The `PWM_DLYC.VALUE` bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel C.

Channel D Delay Register

The `PWM_DLYD` register controls a delay for the Channel D timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (`PWM_CTL.DLYDEN` =1). For more information about applying the delay, see the PWM Functional Description section. Note that the `PWM_DLYD` delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, `PWM_DLYD` must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

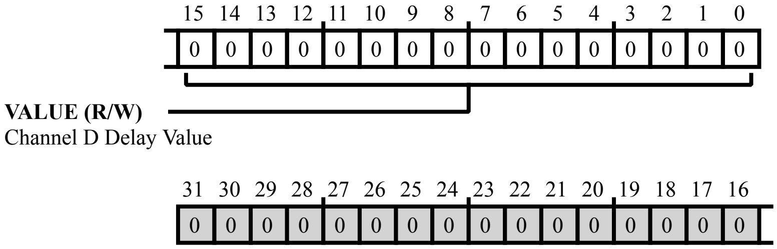

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Channel D Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-88: PWM_DLYD Register Diagram

Table 19-66: PWM_DLYD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel D Delay Value. The `PWM_DLYD.VALUE` bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel D.

Channel D-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_DL_DUTY0` register contains the `PWM_DL_DUTY0.DUTY` bit field from the `PWM_DL0` register and the `PWM_DL_DUTY0.ENHDIV` bit field from the `PWM_DL0_HP` register.

Note that the `PWM_DL_DUTY0` register reads the `PWM_DL0` and the `PWM_DL0_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

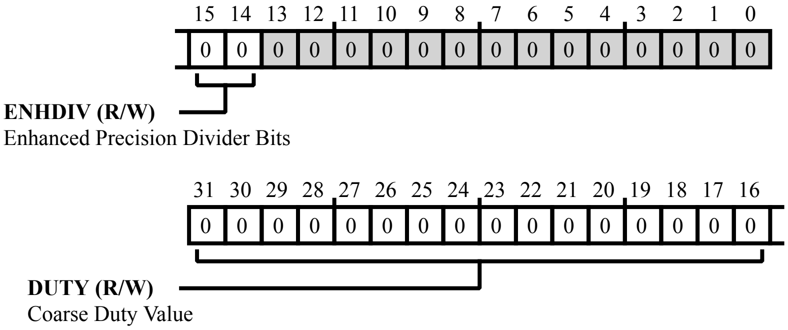

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-89: PWM_DL_DUTY0 Register Diagram

Table 19-67: PWM_DL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_DL_DUTY0.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_DL_DUTY0.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DL_DUTY0.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The `PWM_DL_DUTY1` register contains the `PWM_DL_DUTY1.DUTY` bit field from the `PWM_DL1` register and the `PWM_DL_DUTY1.ENHDIV` bit field from the `PWM_DL1_HP` register.

Note that the `PWM_DL_DUTY1` register reads the `PWM_DL1` and the `PWM_DL1_HP` register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

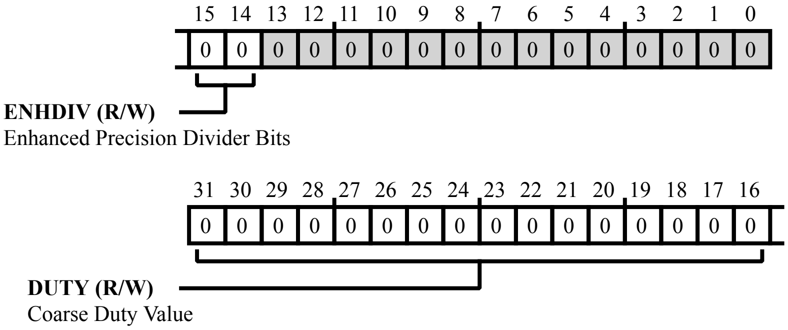

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENHDIV (R/W)
Enhanced Precision Divider Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DUTY (R/W)
Coarse Duty Value

Figure 19-90: PWM_DL_DUTY1 Register Diagram

Table 19-68: PWM_DL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The `PWM_DL_DUTY1.DUTY` bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the `PWM_DL_DUTY1.DUTY` bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The `PWM_DL_DUTY1.ENHDIV` bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Interrupt Latch Register

The `PWM_ILAT` register latches the occurrence of unmasked (enabled) PWM interrupt requests. These interrupt requests are unmasked or masked with the `PWM_IMSK` register.

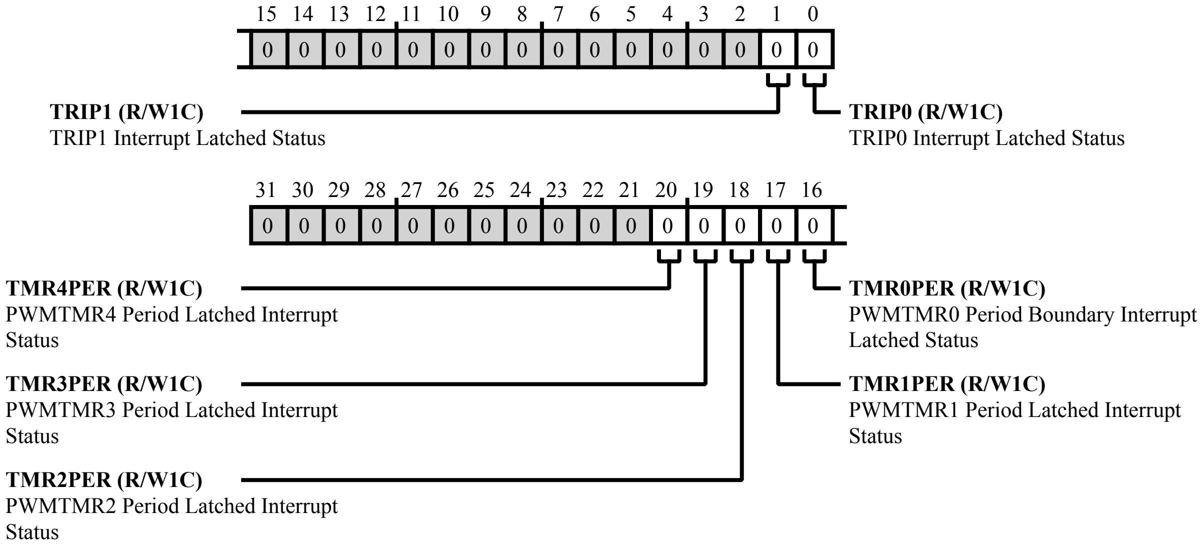

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRIP1 (R/W1C)
TRIP1 Interrupt Latched Status
TRIP0 (R/W1C)
TRIP0 Interrupt Latched Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMR4PER (R/W1C)
PWMTMR4 Period Latched Interrupt Status
TMR0PER (R/W1C)
PWMTMR0 Period Boundary Interrupt Latched Status
TMR3PER (R/W1C)
PWMTMR3 Period Latched Interrupt Status
TMR1PER (R/W1C)
PWMTMR1 Period Latched Interrupt Status
TMR2PER (R/W1C)
PWMTMR2 Period Latched Interrupt Status

Figure 19-91: PWM_ILAT Register Diagram

Table 19-69: PWM_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Latched Interrupt Status. The `PWM_ILAT.TMR4PER` bit indicates the latched status of the PWMTMR4 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
19 (R/W1C)	TMR3PER	PWMTMR3 Period Latched Interrupt Status. The `PWM_ILAT.TMR3PER` bit indicates the latched status of the PWMTMR3 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
18 (R/W1C)	TMR2PER	PWMTMR2 Period Latched Interrupt Status. The `PWM_ILAT.TMR2PER` bit indicates the latched status of the PWMTMR2 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched

Table 19-69: PWM_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	TMR1PER	PWMTMR1 Period Latched Interrupt Status. The `PWM_ILAT.TMR1PER` bit indicates the latched status of the PWMTMR1 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Interrupt Latched Status. The `PWM_ILAT.TMR0PER` bit indicates the latched status of the PWMTMR0 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
1 (R/W1C)	TRIP1	TRIP1 Interrupt Latched Status. The `PWM_ILAT.TRIP1` bit indicates the latched status of the TRIP1 interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
0 (R/W1C)	TRIP0	TRIP0 Interrupt Latched Status. The `PWM_ILAT.TRIP0` bit indicates the latched status of the TRIP0 interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched

Interrupt Mask Register

The `PWM_IMSK` register masks (disables) or unmasks (enables) PWM interrupt requests. When an unmasked interrupt occurs, the PWM latches the interrupt status in the `PWM_ILAT` register.

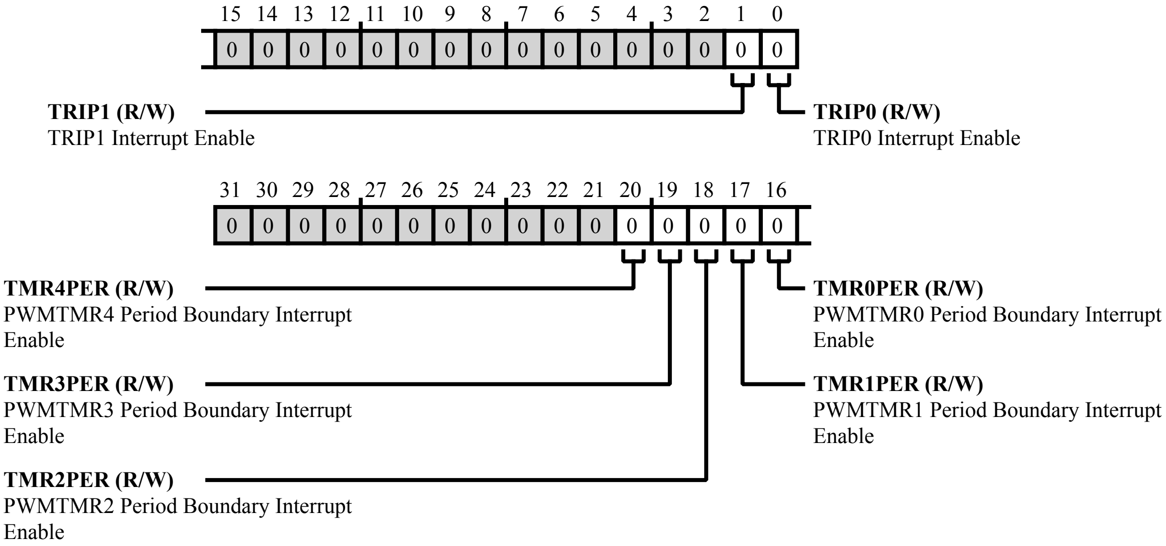

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRIP1 (R/W)
TRIP1 Interrupt Enable
TRIP0 (R/W)
TRIP0 Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMR4PER (R/W)
PWMTMR4 Period Boundary Interrupt Enable
TMR0PER (R/W)
PWMTMR0 Period Boundary Interrupt Enable
TMR3PER (R/W)
PWMTMR3 Period Boundary Interrupt Enable
TMR1PER (R/W)
PWMTMR1 Period Boundary Interrupt Enable
TMR2PER (R/W)
PWMTMR2 Period Boundary Interrupt Enable

Figure 19-92: PWM_IMSK Register Diagram

Table 19-70: PWM_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	TMR4PER	PWMTMR4 Period Boundary Interrupt Enable. The `PWM_IMSK.TMR4PER` bit enables (unmasks) the PWMTMR4 period boundary interrupt request. This condition occurs when the timers period boundary is reached (`PWM_STAT.TMR4PER` =1).	
		0	Mask PWMTMR4 Period Interrupt
		1	Unmask PWMTMR4 Period Interrupt
19 (R/W)	TMR3PER	PWMTMR3 Period Boundary Interrupt Enable. The `PWM_IMSK.TMR3PER` bit enables (unmasks) the PWMTMR3 period boundary interrupt request. This condition occurs when the timers period boundary is reached (`PWM_STAT.TMR3PER` =1).	
		0	Mask PWMTMR3 Period Interrupt
		1	Unmask PWMTMR3 Period Interrupt

Table 19-70: PWM_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	TMR2PER	PWMTMR2 Period Boundary Interrupt Enable. The `PWM_IMSK.TMR2PER` bit enables (unmasks) the PWMTMR2 period boundary interrupt request. This condition occurs when the timers period boundary is reached (`PWM_STAT.TMR2PER` =1).	
		0	Mask PWMTMR2 Period Interrupt
		1	Unmask PWMTMR2 Period Interrupt
17 (R/W)	TMR1PER	PWMTMR1 Period Boundary Interrupt Enable. The `PWM_IMSK.TMR1PER` bit enables (unmasks) the PWMTMR1 period boundary interrupt request. This condition occurs when the timers period boundary is reached (`PWM_STAT.TMR1PER` =1).	
		0	Mask PWMTMR1 Period Interrupt
		1	Unmask PWMTMR1 Period Interrupt
16 (R/W)	TMR0PER	PWMTMR0 Period Boundary Interrupt Enable. The `PWM_IMSK.TMR0PER` bit enables (unmasks) the PWMTMR0 period boundary interrupt request. This condition occurs when the timers period boundary is reached (`PWM_STAT.TMR0PER` =1).	
		0	Mask PWMTMR0 Period Interrupt
		1	Unmask PWMTMR0 Period Interrupt
1 (R/W)	TRIP1	TRIP1 Interrupt Enable. The `PWM_IMSK.TRIP1` bit enables (unmasks) the TRIP1 interrupt request. This condition occurs when fault input is tripped (`PWM_STAT.TRIP1` =1).	
		0	Mask TRIP1 Interrupt
		1	Unmask TRIP1 Interrupt
0 (R/W)	TRIP0	TRIP0 Interrupt Enable. The `PWM_IMSK.TRIP0` bit enables (unmasks) the TRIP0 interrupt request. This condition occurs when fault input is tripped (`PWM_STAT.TRIP0` =1).	
		0	Mask TRIP0 Interrupt
		1	Unmask TRIP0 Interrupt

Status Register

The `PWM_STAT` register indicates the PWM PWMTRIP1-0 fault and input level status, indicates the Channel A-D fault and self-restart status, and indicates the PWMTMR4-0 phase.

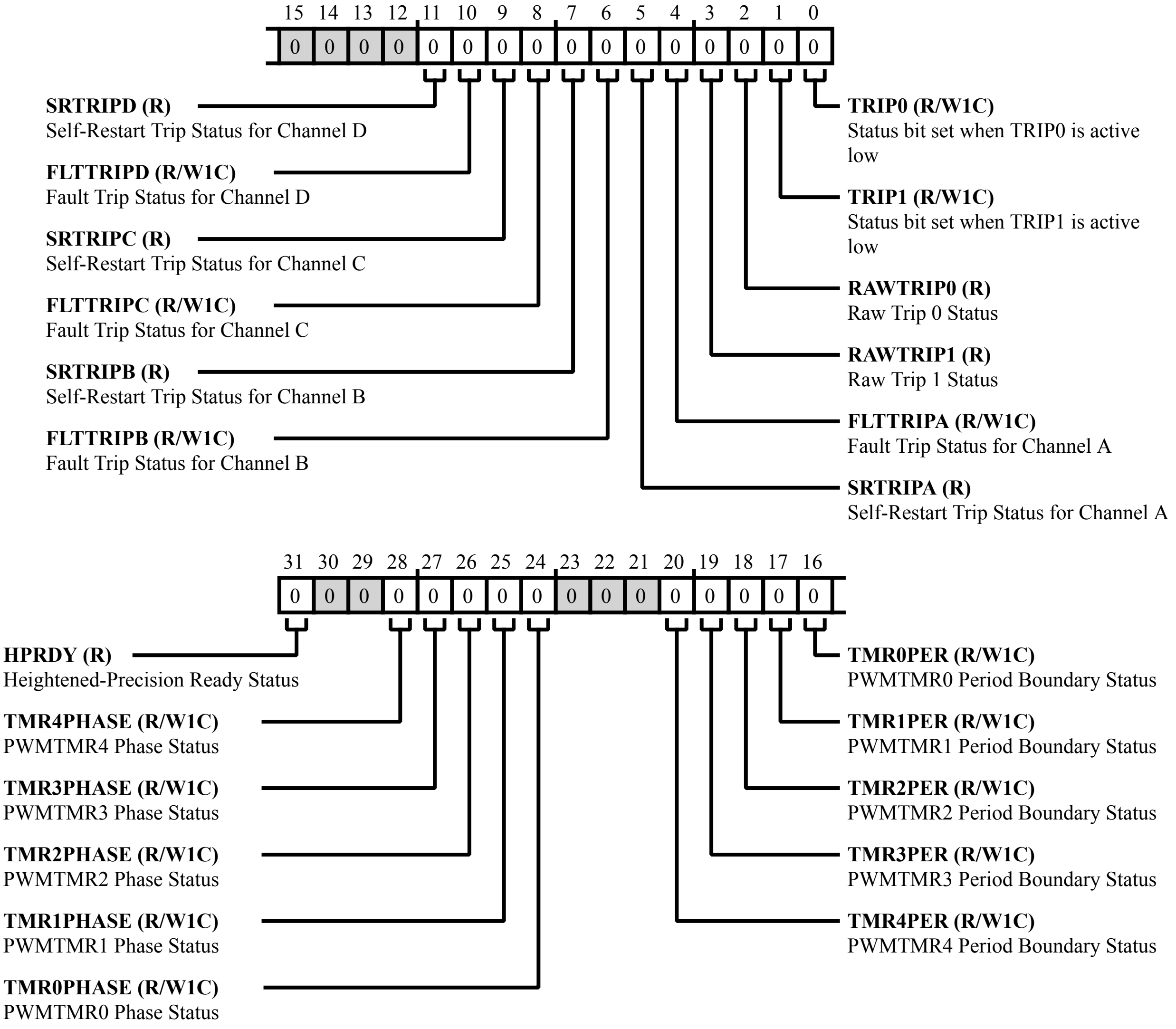

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SRTRIPD (R)
Self-Restart Trip Status for Channel D
FLTTRIPD (R/W1C)
Fault Trip Status for Channel D
SRTRIPC (R)
Self-Restart Trip Status for Channel C
FLTTRIPC (R/W1C)
Fault Trip Status for Channel C
SRTRIPB (R)
Self-Restart Trip Status for Channel B
FLTTRIPB (R/W1C)
Fault Trip Status for Channel B
TRIP0 (R/W1C)
Status bit set when TRIP0 is active low
TRIP1 (R/W1C)
Status bit set when TRIP1 is active low
RAWTRIP0 (R)
Raw Trip 0 Status
RAWTRIP1 (R)
Raw Trip 1 Status
FLTTRIPA (R/W1C)
Fault Trip Status for Channel A
SRTRIPA (R)
Self-Restart Trip Status for Channel A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HPRDY (R)
Heightened-Precision Ready Status
TMR4PHASE (R/W1C)
PWMTMR4 Phase Status
TMR3PHASE (R/W1C)
PWMTMR3 Phase Status
TMR2PHASE (R/W1C)
PWMTMR2 Phase Status
TMR1PHASE (R/W1C)
PWMTMR1 Phase Status
TMR0PHASE (R/W1C)
PWMTMR0 Phase Status
TMR0PER (R/W1C)
PWMTMR0 Period Boundary Status
TMR1PER (R/W1C)
PWMTMR1 Period Boundary Status
TMR2PER (R/W1C)
PWMTMR2 Period Boundary Status
TMR3PER (R/W1C)
PWMTMR3 Period Boundary Status
TMR4PER (R/W1C)
PWMTMR4 Period Boundary Status

Figure 19-93: PWM_STAT Register Diagram

Table 19-71: PWM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	HPRDY	Heightened-Precision Ready Status. The `PWM_STAT.HPRDY` bit indicates whether or not the PWM is ready for heightened-precision operation.	
		0	HPPWM Not Ready For Operation
		1	HPPWM Ready For Operation
28 (R/W1C)	TMR4PHASE	PWMTMR4 Phase Status. The `PWM_STAT.TMR4PHASE` bit indicates the current phase for the PWMTMR4 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
27 (R/W1C)	TMR3PHASE	PWMTMR3 Phase Status. The `PWM_STAT.TMR3PHASE` bit indicates the current phase for the PWMTMR3 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
26 (R/W1C)	TMR2PHASE	PWMTMR2 Phase Status. The `PWM_STAT.TMR2PHASE` bit indicates the current phase for the PWMTMR2 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
25 (R/W1C)	TMR1PHASE	PWMTMR1 Phase Status. The `PWM_STAT.TMR1PHASE` bit indicates the current phase for the PWMTMR1 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
24 (R/W1C)	TMR0PHASE	PWMTMR0 Phase Status. The `PWM_STAT.TMR0PHASE` bit indicates the current phase for the PWMTMR0 waveform.	
		0	1st Half Phase
		1	2nd Half Phase

Table 19-71: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Boundary Status. The `PWM_STAT.TMR4PER` bit indicates whether or not the PWMTMR4 period boundary has been reached.	
		0	PWMTMR4 period boundary not reached
		1	PWMTMR4 period boundary reached
19 (R/W1C)	TMR3PER	PWMTMR3 Period Boundary Status. The `PWM_STAT.TMR3PER` bit indicates whether or not the PWMTMR3 period boundary has been reached.	
		0	PWMTMR3 period boundary not reached
		1	PWMTMR3 period boundary reached
18 (R/W1C)	TMR2PER	PWMTMR2 Period Boundary Status. The `PWM_STAT.TMR2PER` bit indicates whether or not the PWMTMR2 period boundary has been reached.	
		0	PWMTMR2 period boundary not reached
		1	PWMTMR2 period boundary reached
17 (R/W1C)	TMR1PER	PWMTMR1 Period Boundary Status. The `PWM_STAT.TMR1PER` bit indicates whether or not the PWMTMR1 period boundary has been reached.	
		0	PWMTMR1 period boundary not reached
		1	PWMTMR1 period boundary reached
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Status. The `PWM_STAT.TMR0PER` bit indicates whether or not the PWMTMR0 period boundary has been reached.	
		0	PWMTMR0 period boundary not reached
		1	PWMTMR0 period boundary reached
11 (R/NW)	SRTRIPD	Self-Restart Trip Status for Channel D. The `PWM_STAT.SRTRIPD` bit indicates whether the PWM Channel D self-restart has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel D Self-Restart Trip Status is "not tripped"
		1	Channel D Self-Restart Trip Status is "tripped"

Table 19-71: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	FLTTRIPD	Fault Trip Status for Channel D. The `PWM_STAT.FLTTRIPD` bit indicates whether the PWM Channel D fault has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel D Fault Trip Status is "not tripped"
		1	Channel D Fault Trip Status is "tripped"
9 (R/NW)	SRTRIPC	Self-Restart Trip Status for Channel C. The `PWM_STAT.SRTRIPC` bit indicates whether the PWM Channel C self-restart has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel C Self-Restart Trip Status is "not tripped"
		1	Channel C Self-Restart Trip Status is "tripped"
8 (R/W1C)	FLTTRIPC	Fault Trip Status for Channel C. The `PWM_STAT.FLTTRIPC` bit indicates whether the PWM Channel C fault has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel C Fault Trip Status is "not tripped"
		1	Channel C Fault Trip Status is "tripped"
7 (R/NW)	SRTRIPB	Self-Restart Trip Status for Channel B. The `PWM_STAT.SRTRIPB` bit indicates whether the PWM Channel B self-restart has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel B Self-Restart Trip Status is "not tripped"
		1	Channel B Self-Restart Trip Status is "tripped"
6 (R/W1C)	FLTTRIPB	Fault Trip Status for Channel B. The `PWM_STAT.FLTTRIPB` bit indicates whether the PWM Channel B fault has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel B Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"

Table 19-71: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	SRTRIPA	Self-Restart Trip Status for Channel A. The `PWM_STAT.SRTRIPA` bit indicates whether the PWM Channel A self-restart has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel A Self-Restart Trip Status is "not tripped"
		1	Channel A Self-Restart Trip Status is "tripped"
4 (R/W1C)	FLTTRIPA	Fault Trip Status for Channel A. The `PWM_STAT.FLTTRIPA` bit indicates whether the PWM Channel A fault has been tripped. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Channel A Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
3 (R/NW)	RAWTRIP1	Raw Trip 1 Status. The `PWM_STAT.RAWTRIP1` bit indicates the raw input level for the PWM TRIP1 input.	
		0	TRIP1 Level is Low
		1	TRIP1 Level is High
2 (R/NW)	RAWTRIP0	Raw Trip 0 Status. The `PWM_STAT.RAWTRIP0` bit indicates the raw input level for the PWM TRIP0 input.	
		0	TRIP0 Level is Low
		1	TRIP0 Level is High
1 (R/W1C)	TRIP1	Status bit set when TRIP1 is active low. The `PWM_STAT.TRIP1` bit indicates whether the PWM TRIP1 fault has been tripped with an active-low input.	
		0	TRIP1 status is "not tripped"
		1	TRIP1 status is "tripped" (active low)
0 (R/W1C)	TRIP0	Status bit set when TRIP0 is active low. The `PWM_STAT.TRIP0` bit indicates whether the PWM TRIP0 fault has been tripped with an active-low input.	
		0	TRIP0 status is "not tripped"
		1	TRIP0 status is "tripped" (active low)

Sync Pulse Width Register

The `PWM_SYNC_WID` register selects the pulse width for the external sync pulse available on the `PWM_SYNC` pin. The relation between the `PWM_SYNC_WID` register value and the pulse width (T_{PWM_SYNC}) is give by the formula:

$$PWM\_SYNC\_WID = (T_{PWM\_SYNC} / t_{CK}) - 1$$

For more information about applying the sync pulse width, see the PWM Functional Description section. Note that if the pulse width is changed in between sync pulses, the PWM applies the changed width on the next internal sync pulse. If, while the sync pulse is active, the chosen timer reaches its period boundary, the changed pulse width takes effect on that period boundary.

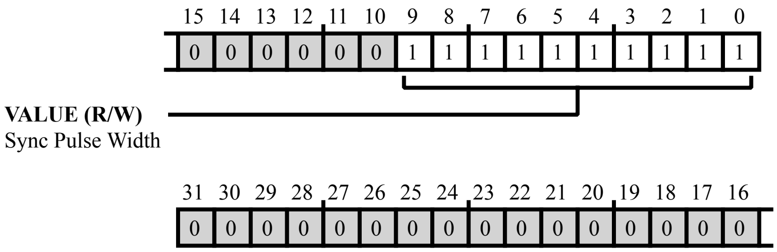

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
VALUE (R/W)
Sync Pulse Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-94: PWM_SYNC_WID Register Diagram

Table 19-72: PWM_SYNC_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Sync Pulse Width. The `PWM_SYNC_WID.VALUE` bits select the pulse width for the external sync pulse available on the `PWM_SYNC` pin.

Timer 0 Period Register

The `PWM_TM0` register controls the switch period T_{SP} of the PWMTMR0 timer. The `PWM_TM0` value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM0= $(T_{SP}) / 2 \times t_{CK}$

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that `PWM_TM0` values of 0 and 1 are not defined and must not be used when the PWM is enabled.

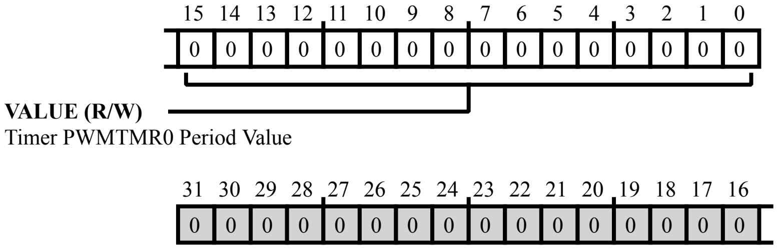

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Timer PWMTMR0 Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-95: PWM_TM0 Register Diagram

Table 19-73: PWM_TM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR0 Period Value. The `PWM_TM0.VALUE` bits select the period for the PWMTMR0 timer.

Timer 1 Period Register

The `PWM_TM1` register controls the switch period (T_{SP} of the PWMTMR1 timer. The `PWM_TM1` value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that `PWM_TM1` values of 0 and 1 are not defined and must not be used when the PWM is enabled.

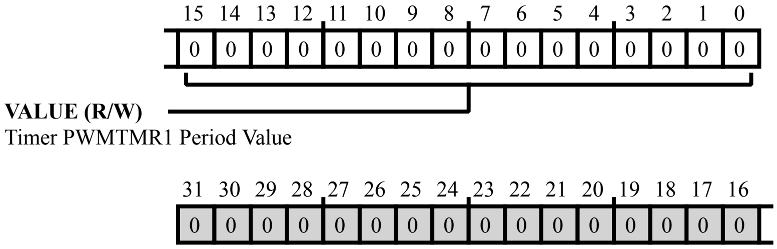

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Timer PWMTMR1 Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-96: PWM_TM1 Register Diagram

Table 19-74: PWM_TM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR1 Period Value. The `PWM_TM1.VALUE` bits select the period for the PWMTMR1 timer.

Timer 2 Period Register

The PWM_TM2 register controls the switch period (T_{SP} of the PWMTMR2 timer. The PWM_TM2 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM2 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

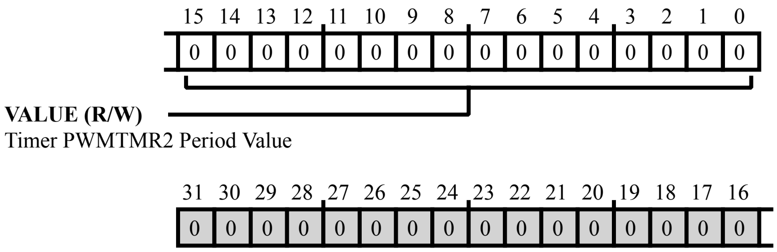

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Timer PWMTMR2 Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-97: PWM_TM2 Register Diagram

Table 19-75: PWM_TM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR2 Period Value. The `PWM_TM2.VALUE` bits select the period for the PWMTMR2 timer.

Timer 3 Period Register

The PWM_TM3 register controls the switch period (T_{SP} of the PWMTMR3 timer. The PWM_TM3 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM3= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM3 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

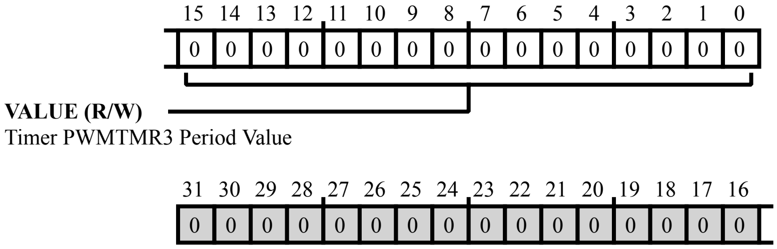

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Timer PWMTMR3 Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-98: PWM_TM3 Register Diagram

Table 19-76: PWM_TM3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR3 Period Value. The PWM_TM3.VALUE bits select the period for the PWMTMR3 timer.

Timer 4 Period Register

The `PWM_TM4` register controls the switch period (T_{SP} of the PWMTMR4 timer. The `PWM_TM4` value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM4= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that `PWM_TM4` values of 0 and 1 are not defined and must not be used when the PWM is enabled.

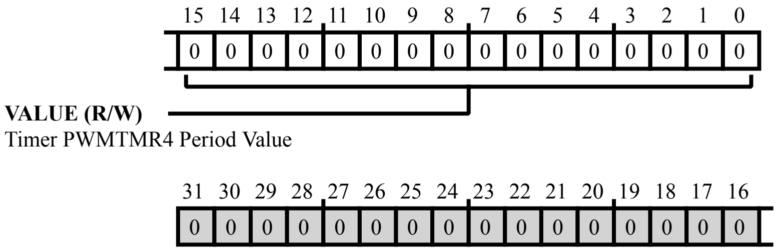

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Timer PWMTMR4 Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 19-99: PWM_TM4 Register Diagram

Table 19-77: PWM_TM4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR4 Period Value. The `PWM_TM4.VALUE` bits select the period for the PWMTMR4 timer.

Trip Configuration Register

The `PWM_TRIPCFG` register configures Channel A, B, C, and D trip operation for trip inputs TRIP0 and TRIP1.

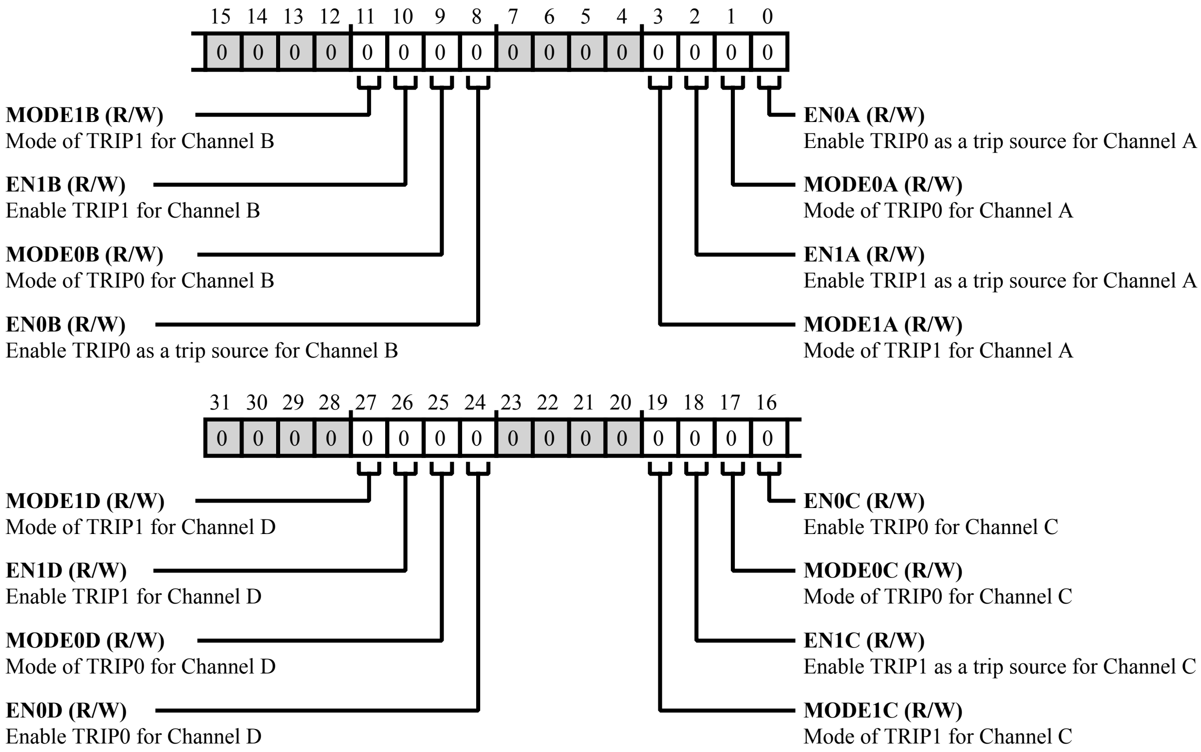

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MODE1B (R/W)
Mode of TRIP1 for Channel B
EN1B (R/W)
Enable TRIP1 for Channel B
MODE0B (R/W)
Mode of TRIP0 for Channel B
EN0B (R/W)
Enable TRIP0 as a trip source for Channel B
EN0A (R/W)
Enable TRIP0 as a trip source for Channel A
MODE0A (R/W)
Mode of TRIP0 for Channel A
EN1A (R/W)
Enable TRIP1 as a trip source for Channel A
MODE1A (R/W)
Mode of TRIP1 for Channel A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MODE1D (R/W)
Mode of TRIP1 for Channel D
EN1D (R/W)
Enable TRIP1 for Channel D
MODE0D (R/W)
Mode of TRIP0 for Channel D
EN0D (R/W)
Enable TRIP0 for Channel D
EN0C (R/W)
Enable TRIP0 for Channel C
MODE0C (R/W)
Mode of TRIP0 for Channel C
EN1C (R/W)
Enable TRIP1 as a trip source for Channel C
MODE1C (R/W)
Mode of TRIP1 for Channel C

Figure 19-100: PWM_TRIPCFG Register Diagram

Table 19-78: PWM_TRIPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	MODE1D	Mode of TRIP1 for Channel D. The `PWM_TRIPCFG.MODE1D` bit selects the trip mode of TRIP1 for Channel D. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
26 (R/W)	EN1D	Enable TRIP1 for Channel D. The `PWM_TRIPCFG.EN1D` bit enables TRIP1 as a trip source for Channel D.	
		0	Disable TRIP1 for Channel D
		1	Enable TRIP1 for Channel D

Table 19-78: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODE0D	Mode of TRIP0 for Channel D. The `PWM_TRIPCFG.MODE0D` bit selects the trip mode of TRIP0 for Channel D. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
24 (R/W)	EN0D	Enable TRIP0 for Channel D. The `PWM_TRIPCFG.EN0D` bit enables TRIP0 as a trip source for Channel D.	
		0	Disable TRIP0 for Channel D
		1	Enable TRIP0 for Channel D
19 (R/W)	MODE1C	Mode of TRIP1 for Channel C. The `PWM_TRIPCFG.MODE1C` bit selects the trip mode of TRIP1 for Channel C. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
18 (R/W)	EN1C	Enable TRIP1 as a trip source for Channel C. The `PWM_TRIPCFG.EN1C` bit enables TRIP1 as a trip source for Channel C.	
		0	Disable TRIP1 for Channel C
		1	Enable TRIP1 for Channel C
17 (R/W)	MODE0C	Mode of TRIP0 for Channel C. The `PWM_TRIPCFG.MODE0C` bit selects the trip mode of TRIP0 for Channel C. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
16 (R/W)	EN0C	Enable TRIP0 for Channel C. The `PWM_TRIPCFG.EN0C` bit enables TRIP0 as a trip source for Channel C.	
		0	Disable TRIP0 for Channel C
		1	Enable TRIP0 for Channel C
11 (R/W)	MODE1B	Mode of TRIP1 for Channel B. The `PWM_TRIPCFG.MODE1B` bit selects the trip mode of TRIP1 for Channel B. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input

Table 19-78: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	EN1B	Enable TRIP1 for Channel B. The `PWM_TRIPCFG.EN1B` bit enables TRIP1 as a trip source for Channel B.	
		0	Disable TRIP1 for Channel B
		1	Enable TRIP1 for Channel B
9 (R/W)	MODE0B	Mode of TRIP0 for Channel B. The `PWM_TRIPCFG.MODE0B` bit selects the trip mode of TRIP0 for Channel B. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
8 (R/W)	EN0B	Enable TRIP0 as a trip source for Channel B. The `PWM_TRIPCFG.EN0B` bit enables TRIP0 as a trip source for Channel B.	
		0	Disable TRIP0 for Channel B
		1	Enable TRIP0 for Channel B
3 (R/W)	MODE1A	Mode of TRIP1 for Channel A. The `PWM_TRIPCFG.MODE1A` bit selects the trip mode of TRIP1 for Channel A. For more information, see the `PWM_TRIPCFG.MODE0A` bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
2 (R/W)	EN1A	Enable TRIP1 as a trip source for Channel A. The `PWM_TRIPCFG.EN1A` bit enables TRIP1 as a trip source for Channel A.	
		0	Disable TRIP1 for Channel A
		1	Enable TRIP1 for Channel A

Table 19-78: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODE0A	Mode of TRIP0 for Channel A. The `PWM_TRIPCFG.MODE0A` bit selects the trip mode of TRIP0 for Channel A. In fault-trip mode (`PWM_TRIPCFG.MODE0A` =0), after the input is tripped, the trip status appears in the corresponding channels fault-trip status bit (for example, `PWM_STAT.FLTTRIPA`), and the PWM immediately shuts down outputs of that channel. After a fault trip occurs, when the trip condition is no longer active, the processor may cause channel outputs to resume by completing a write-1-to-clear the corresponding fault-trip status bit. The raw (input level) trip input state is available from the `PWM_STAT.RAWTRIP0` and `PWM_STAT.RAWTRIP0` bits. In self-restart mode (`PWM_TRIPCFG.MODE0A` =1), after the input is tripped, the trip status appears in the corresponding channels self-restart status bit (for example, `PWM_STAT.SRTRIPA`), and the PWM immediately shuts down outputs of that channel. On the next timer period boundary (of the PWMTMRx used by that channel), if the trip condition is not active, the PWM clears the status and restarts the channels output.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
0 (R/W)	EN0A	Enable TRIP0 as a trip source for Channel A. The `PWM_TRIPCFG.EN0A` bit enables TRIP0 as a trip source for Channel A.	
		0	Disable TRIP0 for Channel A
		1	Enable TRIP0 for Channel A

20 General-Purpose Timer (TIMER)

The general-purpose timer (GP Timer) module serves as a collection of system timers that support various system-level functions. These functions include:

- Synchronized PWM waveform output capability
- External signal capture
- External event count
- General time-base functionality

Additionally, interrupt requests can be generated upon completion of timer events. Moreover, GP timers can act both as trigger masters and trigger slaves.

GP Timer Features

Each timer can be individually configured in any of these modes:

- Pin interrupt capture mode
- Windowed watchdog mode
- Pulse-width count and capture (WDTH_CAP) mode
- External Event (EXT_CLK) mode
- Pulse-width modulation (PWM_OUT) mode

Other features include:

- Synchronous operation
- Consistent management of period and pulse width values
- Autobaud detection for UART module (where available)
- Graceful bit pattern termination when stopping
- Support for center-aligned PWM patterns
- Error detection on implausible pattern values

- All read and write accesses to 32-bit registers are atomic
- Every timer has its dedicated interrupt request output

ADSP-SC58x TIMER Register List

The General-Purpose Timer block (TIMER) provides timers that may be used for external event capture and measurement, system timing, and PWM waveform generation. A set of registers governs TIMER operations. For more information on TIMER functionality, see the TIMER register descriptions.

Table 20-1: ADSP-SC58x TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

ADSP-SC58x TIMER Interrupt List

Table 20-2: ADSP-SC58x TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
26	TIMER0_TMR0	TIMER0 Timer 0	Level	
27	TIMER0_TMR1	TIMER0 Timer 1	Level	
28	TIMER0_TMR2	TIMER0 Timer 2	Level	
29	TIMER0_TMR3	TIMER0 Timer 3	Level	
72	TIMER0_TMR4	TIMER0 Timer 4	Level	
73	TIMER0_TMR5	TIMER0 Timer 5	Level	
74	TIMER0_TMR6	TIMER0 Timer 6	Level	
75	TIMER0_TMR7	TIMER0 Timer 7	Level	
76	TIMER0_STAT	TIMER0 Status	Level	

ADSP-SC58x TIMER Trigger List

Table 20-3: ADSP-SC58x TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
6	`TIMER0_TMR0_MST`	TIMER0 Timer 0	Edge
7	`TIMER0_TMR1_MST`	TIMER0 Timer 1	Edge
8	`TIMER0_TMR2_MST`	TIMER0 Timer 2	Edge
9	`TIMER0_TMR3_MST`	TIMER0 Timer 3	Edge
10	`TIMER0_TMR4_MST`	TIMER0 Timer 4	Edge
11	`TIMER0_TMR5_MST`	TIMER0 Timer 5	Edge
12	`TIMER0_TMR6_MST`	TIMER0 Timer 6	Edge
13	`TIMER0_TMR7_MST`	TIMER0 Timer 7	Edge

Table 20-4: ADSP-SC58x TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	`TIMER0_TMR0_SLV0`	TIMER0 Timer 0	Pulse
1	`TIMER0_TMR1_SLV0`	TIMER0 Timer 1	Pulse
2	`TIMER0_TMR2_SLV0`	TIMER0 Timer 2	Pulse
3	`TIMER0_TMR3_SLV0`	TIMER0 Timer 3	Pulse
4	`TIMER0_TMR4_SLV0`	TIMER0 Timer 4	Pulse
5	`TIMER0_TMR5_SLV0`	TIMER0 Timer 5	Pulse

Table 20-4: ADSP-SC58x TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
6	`TIMER0_TMR6_SLV0`	TIMER0 Timer 6	Pulse
7	`TIMER0_TMR7_SLV0`	TIMER0 Timer 7	Pulse

Timer Block Diagram

The Timer block diagram figure shows all of the possible clock sources.

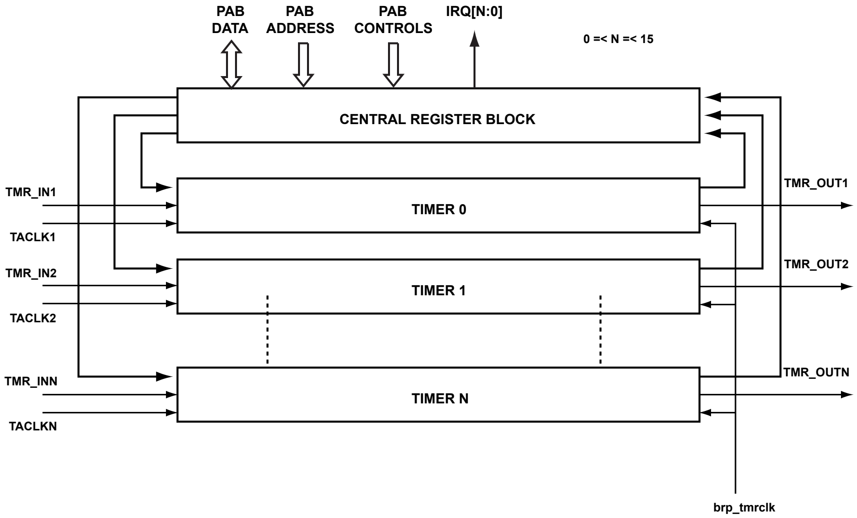

PAB DATA
PAB ADDRESS
PAB CONTROLS
IRQ[N:0]
0 =< N =< 15
CENTRAL REGISTER BLOCK
TMR_IN1
TACLK1
TIMER 0
TMR_OUT1
TMR_IN2
TACLK2
TIMER 1
TMR_OUT2
TMR_INN
TACLKN
TIMER N
TMR_OUTN
brp_tmrclk

Figure 20-1: Timer Block Diagram

Internal Interface

The processor core always accesses the timer registers through the MMR access bus. Hardware ensures that all read and write operations from and to 32-bit timer registers are atomic. Every timer has a dedicated data interrupt request. There is also one common timer status and error interrupt request output that connects to the system event controller. Whenever a data interrupt request is generated, a data trigger master pulse is also driven out, if enabled. Each timer has an individual trigger input line, and each timer can be either started or stopped as a trigger slave.

In total, the GP timer module can have up to (N + 1) interrupt request output lines and N data trigger lines.

Internal Timer Connections

The Timers support alternate inputs for the clock/capture (see External Interface). Some signals have internal default alternate connections according to the ***Timer Signal Routing*** table.

Table 20-5: Timer Signal Routing

Timer Signal	Connection
TM0_ACLK0	SYS_CLKIN1

Table 20-5: Timer Signal Routing (Continued)

Timer Signal	Connection
TM0_ACI5	DAI0_CRS_PB04
TM0_ACLK5	DAI0_CRS_PB03
TM0_ACI6	DAI1_CRS_PB04
TM0_ACLK6	DAI1_CRS_PB03
TM0_ACI07	CNT0_TO signal
TM0_ACLK7	SYS_CLKIN0

External Interface

Each GP timer module can support up to 16 individual timers. However, most processors have less than this number. The exact number of timers available on a given processor is available in the data sheet for the processor.

Every timer has one main input/output signal (`TIMER_TMR[n]`) and, usually, one auxiliary input pin, used as an alternate capture input (`TIMER_ACI[n]`). Each timer can either run with a time base of SCLK0_0 or can reference an external clock on one of two `TIMER_ACLK[n]` pins. The TMR_ALT_CLK0 signal maps to individual alternate clock (`TIMER_ACLK[n]`) pins for one or more timers. For instance, a TM_ACLK3 pin would provide an alternate site to supply an external signal that would serve as reference clock for TMR3. Likewise, the TMR_ALT_CLK1 signal from each timer unit connects together internally to provide a single global timer clock pin (`TIMER_CLK`) for the GP timer module. It is used as an additional time base.

GP Timer Operating Modes

The following sections provide information on the various operating modes of the GP timer.

General Operation

The core of every timer is a 32-bit counter that can be interrogated through the read-only `TIMER_TMR[n]_CNT` register. Once the module enables a timer, it loads the timer `TIMER_TMR[n]_CNT` register with a starting value.

A timer can operate in one of several different modes, configured through the `TIMER_TMR[n]_CFG` register for that timer. These modes are: PWMOUT, EXTCLK, WIDCAP, WATCHDOG, PININT, and IDLE. The ***Timer Mode Descriptions*** table summarizes the modes.

Table 20-6: Timer Mode Descriptions

Timer Mode	Description
PWMOUT	Generates single or continuous PWM waveforms with programmable pulse width, period, and delay
EXTCLK	Counts edges of an externally applied waveform
WIDCAP	Captures pulse width or period of an externally applied waveform

Table 20-6: Timer Mode Descriptions (Continued)

Timer Mode	Description
WATCHDOG	Monitors pulse width or period of an external signal and compares against a window of acceptable values, optionally generating an interrupt when it falls inside or outside of that window
PININT	Can generate an interrupt request on an active edge applied to a timer pin
IDLE	Idle; no activity

Period, Width and Delay Register Interaction

When the timer is started, writes to the buffer registers are immediately copied through to the double-buffered period, pulse width, and delay registers. These values are then ready for use in the first timer period. When a timer is already running, software can write new values to the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers. The written values are buffered and do not update into the registers until the end of the current period. (The update occurs when the value in the `TIMER_TMR[n]_CNT` register equals the value in the `TIMER_TMR[n]_PER` register.)

If new values are not written to these registers, the value from the previous period is reused. Writes to these registers are atomic; it is not possible for the high word to be written without the low word also being written. Values written to the period, pulse width, and delay registers are always stored in the buffer registers. Reads from the same register always return the current, active value of period, pulse width, or delay value. Written values are not readback until they become active.

The usage of the `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` registers varies, depending on the mode of the timer specified by the `TIMER_TMR[n]_CFG.TMODE` bits. See the ***Usage of the Period, Width, and Delay Registers in Different Timer Modes*** table for more information.

Table 20-7: Usage of the Period, Width, and Delay Registers in Different Timer Modes

Timer Mode	**`TIMER_TMR[n]_PER`**	**`TIMER_TMR[n]_WID`**	**`TIMER_TMR[n]_DLY`**
IDLE	Not writable	Not writable	Not writable
WATCHDOG	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.	Read-only. Retains value of last measured width or period of the input signal.	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.
WIDCAP	Read-only. Period value captured at the appropriate time and updated from its buffer register simultaneously with the Width register.	Read-only. Width value captured at the appropriate time and updated from its buffer register simultaneously with the Period register.	Not used
PWMOUT	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.
EXTCLK	Can be updated on-the-fly.	Not used	Not used

Table 20-7: Usage of the Period, Width, and Delay Registers in Different Timer Modes (Continued)

Timer Mode	TIMER_TMR[n]_PER	TIMER_TMR[n]_WID	TIMER_TMR[n]_DLY
PININT	Not used	Not used	Not used

If any of the period, pulse width, and delay registers are not used, then programs cannot write into that register. For example, in WIDCAP mode, the delay registers are not used. So, the program is not allowed to write any value to the TIMER_TMR[n]_DLY register. To prevent undesired operation, program the TIMER_TMR[n]_CFG.TMODE bits before programming the period, width, or delay registers.

If a program changes the TIMER_TMR[n]_CFG.TMODE bits from a status register to writable register (for example in PWMOUT mode), hardware does not clear these registers. These values are automatically overwritten by new values specified by software.

In PWMOUT mode with small periods, there may not be enough time between updates from the buffer registers to write these registers. The next period can use one old value and one new value. To prevent (width + pulse delay) > period errors, write the width and delay registers before the period register when decreasing the values. Write the period register before the width and delay registers when increasing the value.

Single-Pulse PWMOUT Mode

In single-pulse PWMOUT mode, the timer generates a single pulse on the TIMER_TMR[n] pin. This mode is frequently used to implement a precise delay, often with generating an output trigger. The timer module uses the value in the TIMER_TMR[n]_DLY register to control the assertion of a pulse. The value in the TIMER_TMR[n]_WID register defines the pulse width. The TIMER_TMR[n]_PER is not used and cannot be written in this mode. After completion of the pulse, the timer is automatically stopped, and optionally generates an interrupt. The timer uses the TIMER_TMR[n]_CFG.PULSEHI bit to control pulse polarity.

The timer can be configured to generate a data interrupt request after satisfying various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is not necessary to clear the relevant TIMER_RUN bit to stop the timer cleanly. At the end of the pulse, the timer stops automatically and the corresponding TIMER_RUN bit is cleared. To generate multiple discrete pulses (as opposed to a continuous PWM waveform), write a 1 to the appropriate TIMER_RUN bit, and wait for the timer to stop. Then, write another 1 to the same TIMER_RUN bit.

Continuous PWMOUT Mode

In continuous PWMOUT mode, the timer generates a repetitive pulse with a well-defined period, duty cycle, and pulse position. The TIMER_TMR[n]_DLY, TIMER_TMR[n]_PER, and TIMER_TMR[n]_WID registers are programmed with the values of the required PWM pulse. After the timer is started, the counter counts towards the value programmed in the TIMER_TMR[n]_PER register. Initially, the TIMER_TMR[n] pin remains in a deasserted state. The pin toggles to an asserted state when the value in the TIMER_TMR[n]_CNT register equals the value in the TIMER_TMR[n]_DLY register.

The timer can control the assertion sense of the TIMER_TMR[n] pin with the TIMER_TMR[n]_CFG.PULSEHI bit. The TIMER_TMR[n] pin holds this value for the number of clock cycles specified in the TIMER_TMR[n]_WID register. Then, the pin deasserts and holds this value until the completion of the programmed period. The same waveform is generated repeatedly until the timer is disabled.

The timer can be configured to generate a data interrupt request after satisfying any of various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is important to guarantee that the programmed period is greater than or equal to the sum of width and delay. Similarly, delay must be less than period. Violating either of these criteria results in an unpredictable waveform on the TIMER_TMR[n] pin until the situation is rectified by writing proper values to these registers.

The maximum frequency possible to generate on the TIMER_TMR[n] pin is achieved by setting TIMER_TMR[n]_PER to 2 and TIMER_TMR[n]_WID to 1. This operation makes the TIMER_TMR[n] pin toggle each SCLK0_0 clock cycle (assuming the timer is configured to clock internally), producing a duty cycle of 50%.

When the TIMER_STOP_CFG.TMR[nn] bit of a timer is 0, the timer treats a stop operation as a stop-is-pending condition. When terminated with this setting, the timer automatically completes the current waveform and then stops cleanly, remaining in a deasserted state. This funtionality prevents truncation of the current pulse and unwanted PWM patterns at the TIMER_TMR[n] pin. The processor can determine when the timer stops running by polling the corresponding TIMER_RUN.TMR[nn] bit until it reads 0 or by waiting for the last interrupt (if enabled).

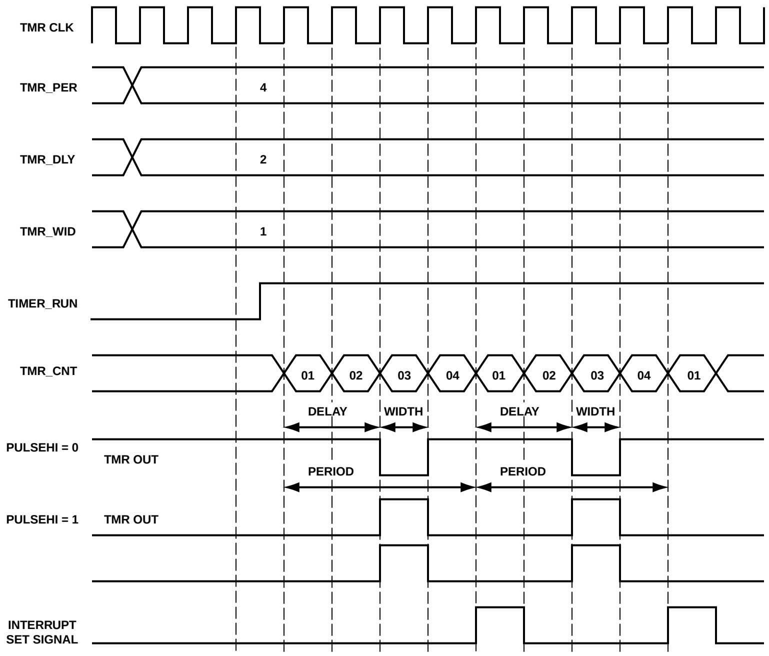


Figure 20-2: Signal Generation in Continuous PWMOUT Mode

The TIMER_TMR[n]_CFG register cannot be reconfigured until after the timer stops and the TIMER_RUN register reads 0.

Programs can force a timer to stop immediately in PWMOUT mode by writing a 1 to the `TIMER_STOP_CFG` register followed by writing a 1 to the `TIMER_RUN_CLR` register. (Or, a program can stop a timer by writing a 0 to the appropriate `TIMER_RUN.TMR[nn]` bit.) This operation stops the timer whether the pending stop is waiting for the end of the current period or the end of the current pulse width. The timer can use this feature to regain immediate control of a timer during an error recovery sequence.

Use this feature carefully, as it can corrupt the PWM pattern generated at the `TIMER_TMR[n]` pin, though after such a stop the pin deasserts automatically. Each timer samples its `TIMER_RUN.TMR[nn]` bit at the end of each period. It stops cleanly at the end of the first period after the `TIMER_RUN.TMR[nn]` bit is low. A timer that is disabled and then restarted (before the end of the current period), continues to run as if nothing happened. Typically, the program disables a PWMOUT timer and then waits for it to stop itself.

Width Capture (WIDCAP) Mode

The timer uses WIDCAP mode, often called capture mode, to measure pulse widths on the `TIMER_TMR[n]` or `TIMER_ACI[n]` inputs. The polarity (active high or low) of the input signal can be selected with the `TIMER_TMR[n]_CFG.PULSEHI` bit. The ***Timer Signal Flow in Width Capture Mode*** figure shows the control signal flow for WIDCAP_CAP mode.

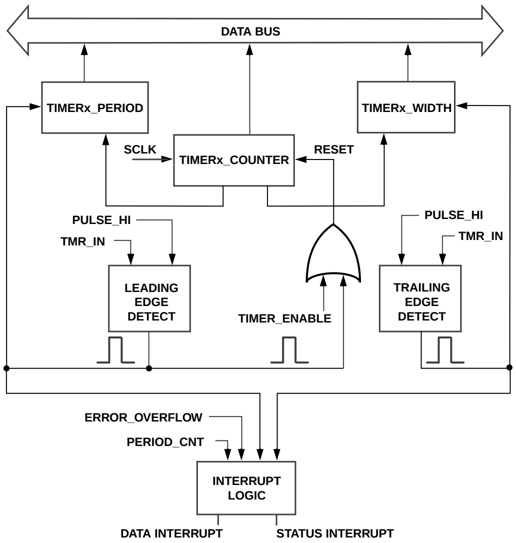

DATA BUS
TIMERx_PERIOD
TIMERx_WIDTH
SCLK
TIMERx_COUNTER
RESET
PULSE_HI
TMR_IN
PULSE_HI
TMR_IN
LEADING EDGE DETECT
TIMER_ENABLE
TRAILING EDGE DETECT
ERROR_OVERFLOW
PERIOD_CNT
INTERRUPT LOGIC
DATA INTERRUPT
STATUS INTERRUPT

Figure 20-3: Timer Signal Flow in Width Capture Mode

NOTE: SCLK in the ***Timer Signal Flow in Width Capture Mode*** figure is SCLK0_0.

In this mode, the timer uses the `TIMER_TMR[n]_CFG.TINSEL` bit to select between the `TIMER_TMR[n]` or `TIMER_ACI[n]` input. The internally clocked timer is used to determine the period and pulse width of the externally applied rectangular waveforms.

When a timer is enabled in this mode, the timer resets the count in its `TIMER_TMR[n]_CNT` register to 0x0000 0001. It does not start counting until it detects a leading edge on the selected input pin.

When the timer detects the first leading edge, it starts incrementing. When it detects a trailing edge of a waveform, it captures the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its width buffer register. At the next leading edge, the timer transfers the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its period buffer register. The `TIMER_TMR[n]_CNT` register is reset to 0x0000 0001 again, and the timer continues counting and capturing until it is disabled.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. The timer does not use the `TIMER_TMR[n]_DLY` register in this mode. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to control the definition of leading edge and trailing edge of the `TIMER_TMR[n]/TIMER_ACI[n]` pin.

In WIDCAP mode, the following events always occur at the same time as one unit:

1. The `TIMER_TMR[n]_PER` register is updated from the period buffer register.
2. The `TIMER_TMR[n]_WID` register is updated from the width buffer register.
3. The `TIMER_DATA_ILAT.TMR[nn]` bit is set (if enabled).
4. A timer data trigger pulse is generated (if enabled).

The `TIMER_TMR[n]_CFG.TMODE` bit 0 controls the point in time at which this set of events is executed. Taken together, these four events are called a measurement report. The `TIMER_STAT_ILAT` register is not set at a measurement report. A measurement report occurs, at most, once per input signal period. The current `TIMER_TMR[n]_CNT` value is always copied to the width buffer and period buffer registers at the trailing and leading edges of the input signal, respectively. But, these values are not visible to software. A measurement report event samples the captured values into visible registers and sets the timer interrupt request to signal that the `TIMER_TMR[n]_PER` and the `TIMER_TMR[n]_WID` registers are ready to be read.

When the `TIMER_TMR[n]_CFG.TMODE` bit =b#1011, the measurement report occurs just after the width buffer register captures its value at a falling edge. Then, the `TIMER_TMR[n]_WID` register reports the pulse width measured in the pulse that has ended, but the `TIMER_TMR[n]_PER` register reports the pulse period measured at the end of the previous period. If only the first trailing edge has occurred, then the first period value has not yet been measured at the first measurement report. So, the period value is not valid. A read of the `TIMER_TMR[n]_PER` value in this case returns 0. See the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure for more information.

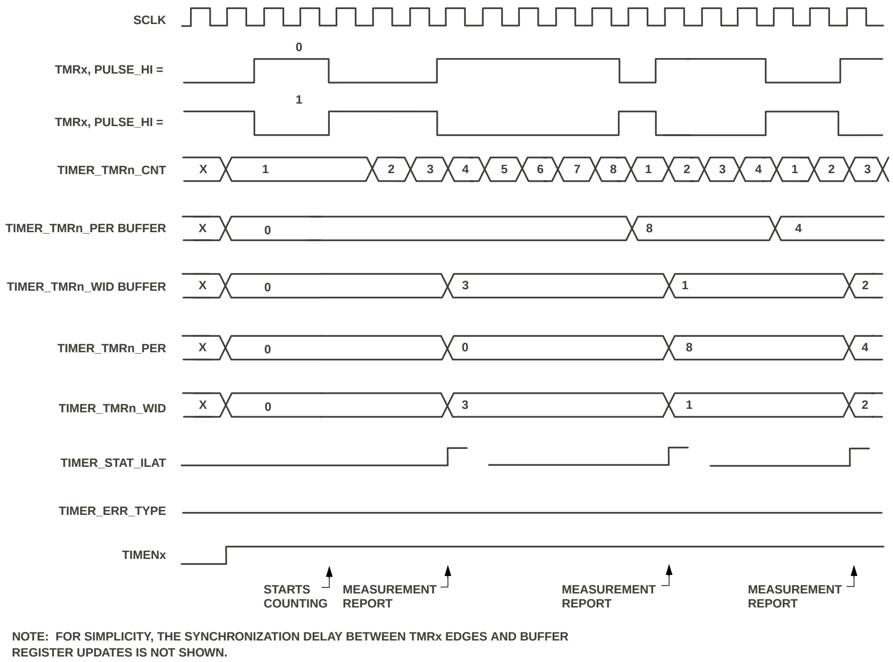

SCLK
TMRx, PULSE_HI =
0
TMRx, PULSE_HI =
1
TIMER_TMRn_CNT
X
1
2
3
4
5
6
7
8
1
2
3
4
1
2
3
TIMER_TMRn_PER BUFFER
X
0
8
4
TIMER_TMRn_WID BUFFER
X
0
3
1
2
TIMER_TMRn_PER
X
0
0
8
4
TIMER_TMRn_WID
X
0
3
1
2
TIMER_STAT_ILAT
TIMER_ERR_TYPE
TIMENx
STARTS COUNTING
MEASUREMENT REPORT
MEASUREMENT REPORT
MEASUREMENT REPORT
NOTE: FOR SIMPLICITY, THE SYNCHRONIZATION DELAY BETWEEN TMRx EDGES AND BUFFER REGISTER UPDATES IS NOT SHOWN.

Figure 20-4: Example of Width Capture Deasserted Mode (TMODE=b#1011)

NOTE: SCLK in the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure is SCLK0_0.

When the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010, the measurement report occurs just after the period buffer register captures its value at a leading edge. Then, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers report the pulse period and pulse width measured in the period that has ended. Refer to the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure for more information.

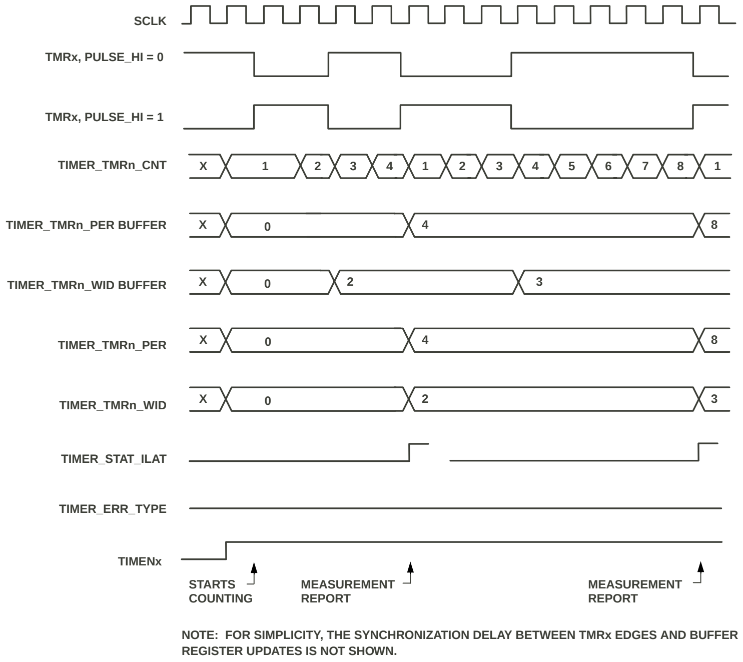

SCLK
TMRx, PULSE_HI = 0
TMRx, PULSE_HI = 1
TIMER_TMRn_CNT
X 1 2 3 4 1 2 3 4 5 6 7 8 1
TIMER_TMRn_PER BUFFER
X 0 4 8
TIMER_TMRn_WID BUFFER
X 0 2 3
TIMER_TMRn_PER
X 0 4 8
TIMER_TMRn_WID
X 0 2 3
TIMER_STAT_ILAT
TIMER_ERR_TYPE
TIMENx
STARTS COUNTING
MEASUREMENT REPORT
MEASUREMENT REPORT
NOTE: FOR SIMPLICITY, THE SYNCHRONIZATION DELAY BETWEEN TMRx EDGES AND BUFFER REGISTER UPDATES IS NOT SHOWN.

Figure 20-5: Example of Width Capture Asserted Mode (TMODE=b#1010)

NOTE: SCLK in the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure is SCLK0_0.

To measure the pulse width of a waveform that has only one leading edge and one trailing edge, set `TIMER_TMR[n]_CFG.TMODE` = b#1011. If `TIMER_TMR[n]_CFG.TMODE` = b#1010 for this case, no period value is captured in the period buffer register. Instead, the timer genererates an error report interrupt request (if enabled) when the `TIMER_TMR[n]_CNT` range is exceeded and the counter wraps around. In this case, both the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers read 0 (because no measurement report occurred to copy the value captured in the width buffer register to the `TIMER_TMR[n]_WID` register).

If using the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010 mode to measure the width of a single pulse, programs can disable the timer after taking the interrupt that ends the measurement interval. If desired, restart the timer as appropriate in preparation for another measurement. This procedure prevents the timer from free-running after the width measurement and logging errors generated by the timer count overflowing.

Width Capture Mode Overflow

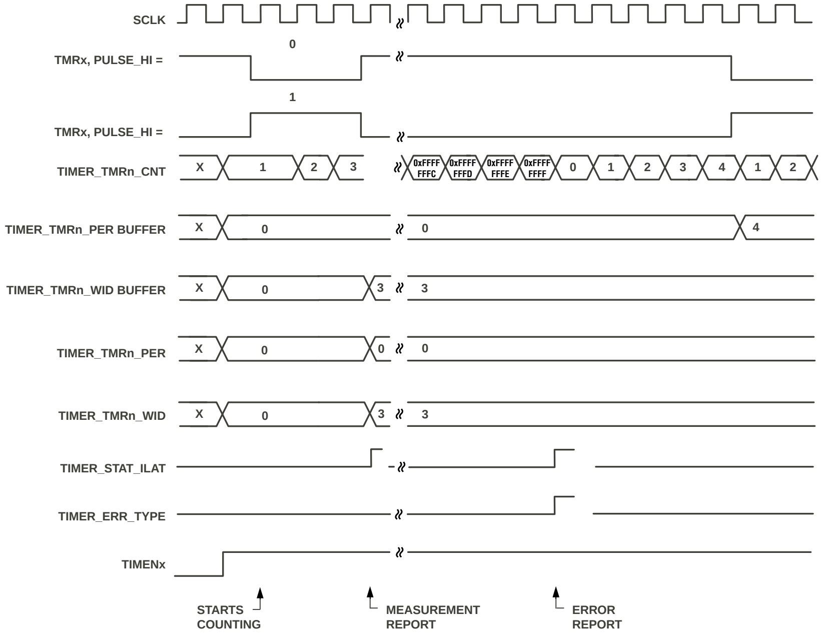
A timer status interrupt request (when enabled) is generated when the `TIMER_TMR[n]_CNT` register wraps around from 0xFFFF FFFF to 0 in the absence of a leading edge. At that point, the `TIMER_STAT_ILAT` bit is set and the `TIMER_ERR_TYPE` bits change to indicate a count overflow due to a period greater than the range of the counter. This indication is referred to as an error report. A data interrupt request in WIDCAP mode indicates that a new measurement is ready to be read (a measurement report). Similarly, an interrupt request on the timer status interrupt line (shared interrupt request for all timers) indicates an overflow error when generated in this mode.

The `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are never updated at the time an overflow error is signaled. If the timer overflows and the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are not updated. If the timer overflows and the

`TIMER_TMR[n]_CFG.TMODE` bit =b#1011, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are updated only if a trailing edge is detected at a previous measurement report.

Software can count the number of error reports between measurement report interrupt requests to measure input signal periods longer than 0xFFFF FFFF. Each error report interrupt request adds a full 2^{32}SCLK0_0 counts to the total for the period, but the width is ambiguous. Ensure that if software monitors only the status interrupt request, then status interrupt requests from all other timers are masked.

Refer to the ***Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)*** figure. The period is 0x1 0000 0004, but the pulse width could be either 0x0 0000 0002 or 0x1 0000 0002.

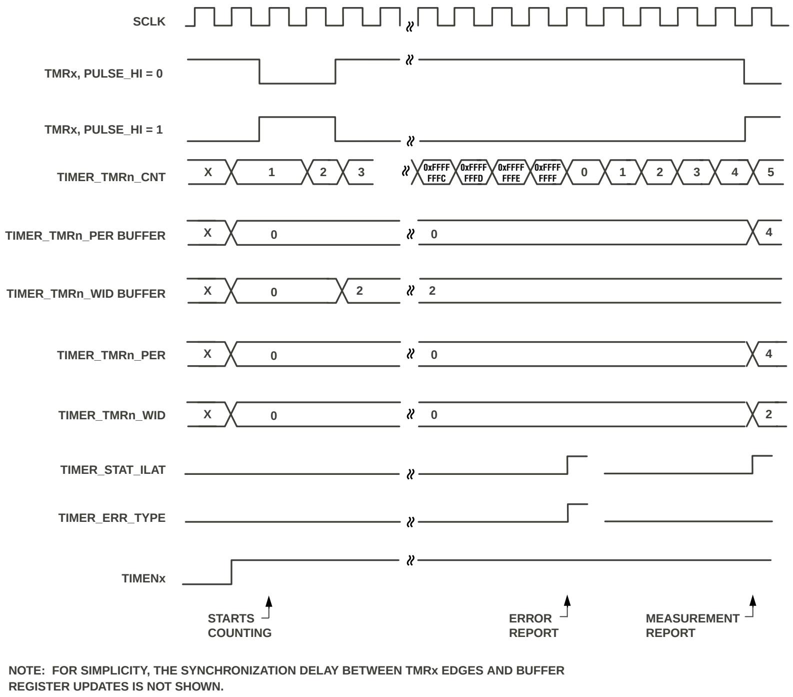

SCLK
TMRx, PULSE_HI =
0
1
TMRx, PULSE_HI =
TIMER_TMRn_CNT
X
1
2
3
0xFFFF FFFC
0xFFFF FFFD
0xFFFF FFFE
0xFFFF FFFF
0
1
2
3
4
1
2
TIMER_TMRn_PER BUFFER
X
0
0
4
TIMER_TMRn_WID BUFFER
X
0
3
3
TIMER_TMRn_PER
X
0
0
0
TIMER_TMRn_WID
X
0
3
3
TIMER_STAT_ILAT
TIMER_ERR_TYPE
TIMENx
STARTS COUNTING
MEASUREMENT REPORT
ERROR REPORT

NOTE: FOR SIMPLICITY, THE SYNCHRONIZATION DELAY BETWEEN TMRx EDGES AND BUFFER REGISTER UPDATES IS NOT SHOWN.

Figure 20-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK0_0.

The waveform applied to the `TIMER_TMR[n]` (or `TIMER_ACI[n]`) pin is not required to have a 50% duty cycle. The minimum input low time is little more than one SCLK0_0 period. The minimum input high time is a little more than one SCLK0_0 period. (Refer to the product data sheet for details.) The maximum `TIMER_TMR[n]` input frequency is less than SCLK0_0/2, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer measures: period =2 and pulse width =1.


SCLK
TMRx, PULSE_HI = 0
TMRx, PULSE_HI = 1
TIMER_TMRn_CNT
X
1
2
3
0xFFFF FFFC
0xFFFF FFFD
0xFFFF FFFE
0xFFFF FFFF
0
1
2
3
4
5
TIMER_TMRn_PER BUFFER
X
0
0
4
TIMER_TMRn_WID BUFFER
X
0
2
2
TIMER_TMRn_PER
X
0
0
4
TIMER_TMRn_WID
X
0
0
2
TIMER_STAT_ILAT
TIMER_ERR_TYPE
TIMENx
STARTS COUNTING
ERROR REPORT
MEASUREMENT REPORT

NOTE: FOR SIMPLICITY, THE SYNCHRONIZATION DELAY BETWEEN TMRx EDGES AND BUFFER REGISTER UPDATES IS NOT SHOWN.

Figure 20-7: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK0_0.

Windowed Watchdog (WATCHDOG) Modes

In windowed watchdog (WATCHDOG) modes, a timer can take inputs from either the `TIMER_TMR[n]` pin or the `TIMER_ACI[n]` pin. With this mode, the timer can monitor pulse width (width watchdog mode) or pulse period (period watchdog mode) on the input line. It also compares the measured value against a minimum required value and maximum allowed value and generates an interrupt request appropriately. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to select polarity of the input signal.

The waveform applied to the input pin in watchdog mode is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet.

Windowed Watchdog Width Mode

In windowed watchdog width mode, the timer counter monitors the pulse width of an input signal on either the `TIMER_TMR[n]` pin or one of the alternate clock pins (`TIMER_ACLK[n]`). Program the minimum pulse width (p_{MIN}) in the `TIMER_TMR[n]_DLY` register and the maximum pulse width (p_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SCLK0_0 or alternate clock). The timer can generate an interrupt if the deasserting pulse edge occurs:

- Inside the window (p_{MIN} < pulse width ≤ p_{MAX}), or
- Outside the window (pulse width ≤ p_{MIN} or pulse width > p_{MAX})

After enabling the timer in this mode, it always starts counting at the asserting edge of the input signal. Any pulse that is already active when the timer is enabled is ignored.

With the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11, the timer generates an interrupt if the timed pulse width exceeds p_{MAX}, or if the pulse width is less than p_{MIN}. After attaining p_{MAX}, the pulse stays at an active level, and the counter keeps on counting until it sees a deasserting edge. When the input pulse is not active, the counter holds its current value until it again sees an asserting edge, or it restarts. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE` =b#10.

In this mode, a trailing edge on the input pin triggers capturing of pulse width into the `TIMER_TMR[n]_WID` register. During the inactive portion of the input signal, the internal counter does not increment. The ***Watchdog Width Mode Timing*** figure shows the signal flow in this mode.

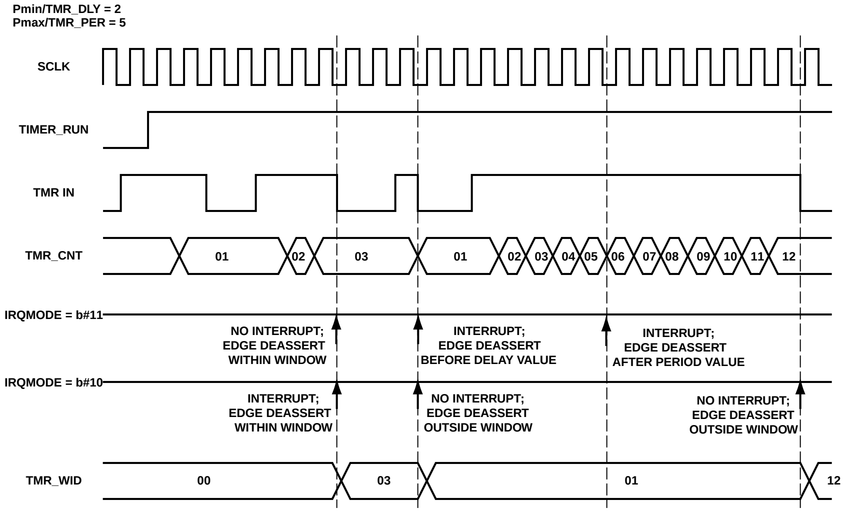

Pmin/TMR_DLY = 2
Pmax/TMR_PER = 5
SCLK
TIMER_RUN
TMR IN
TMR_CNT
01
02
03
01
02
03
04
05
06
07
08
09
10
11
12
IRQMODE = b#11
NO INTERRUPT; EDGE DEASSERT WITHIN WINDOW
INTERRUPT; EDGE DEASSERT BEFORE DELAY VALUE
INTERRUPT; EDGE DEASSERT AFTER PERIOD VALUE
IRQMODE = b#10
INTERRUPT; EDGE DEASSERT WITHIN WINDOW
NO INTERRUPT; EDGE DEASSERT OUTSIDE WINDOW
NO INTERRUPT; EDGE DEASSERT OUTSIDE WINDOW
TMR_WID
00
03
01
12

Figure 20-8: Watchdog Width Mode Timing

NOTE: SCLK in the ***Watchdog Width Mode Timing*** figure is SCLK0_0.

To check only the upper limit on pulse width (p_{MAX} but not p_{MIN}), then program p_{MIN} as 0 or 1. In such a case, it is better to use `TIMER_TMR[n]_CFG.IRQMODE` =b#11. With `TIMER_TMR[n]_CFG.IRQMODE` = b#10, a pulse width of 1 clock cycle results in an interrupt. For details, see the ***Windowed Watchdog Width Mode Interpretation*** table.

Table 20-8: Windowed Watchdog Width Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
0 or 1	Anything ≥ 1	PW = 1	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW ≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
> 1 but ≤ (Period -1)	Anything > 1	PW ≤ TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY < PW≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
≥ Period	-	PW ≤ TMR_PER	Undefined	Undefined	No Error Interrupt
	-	PW > TMR_PER	Undefined	Undefined	b#11 Error Type
-	0	-	Undefined	Undefined	b#10 Error Type

Windowed Watchdog Period Mode

In this mode, the timer monitors the number of clock cycles between two consecutive rising or falling edges of an input signal on either the `TIMER_TMR[n]` or `TIMER_ACI[n]` pin. Program the required minimum number of clock cycles (t_{MIN}) in the `TIMER_TMR[n]_DLY` register and the required maximum allowed number of clock cycles (t_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SCLK0_0) or alternate time clock (`TIMER_ACLK[n]`). The timer can generate an interrupt when two consecutive occurrences of an active edge are:

- Within a specified window (t_{MIN} < Pulse Period ≤ t_{MAX}), or
- Outside a specified window (pulse width ≤ (t_{MIN} or t_{MAX} < pulse width)

When the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11 and the pulse period > t_{MAX} or is ≤ t_{MIN}, the timer generates an interrupt (if unmasked). After attaining the t_{MAX} value, the counter keeps on counting until it sees an active edge on the input line. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE` =b#10. Refer to the ***Watchdog Period Mode Timing*** figure for timer functionality in period watchdog mode.

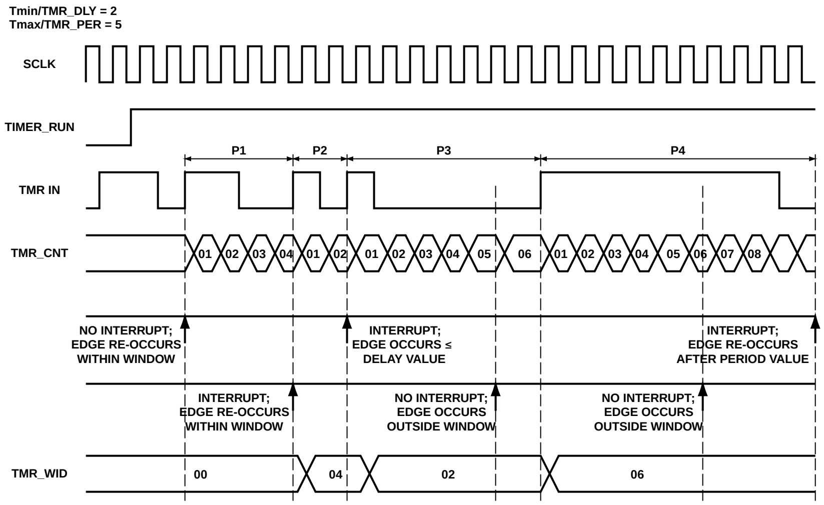

Tmin/TMR_DLY = 2
Tmax/TMR_PER = 5
SCLK
TIMER_RUN
P1
P2
P3
P4
TMR IN
TMR_CNT
01
02
03
04
01
02
01
02
03
04
05
06
01
02
03
04
05
06
07
08
NO INTERRUPT; EDGE RE-OCCURS WITHIN WINDOW
INTERRUPT; EDGE OCCURS ≤ DELAY VALUE
INTERRUPT; EDGE RE-OCCURS AFTER PERIOD VALUE
INTERRUPT; EDGE RE-OCCURS WITHIN WINDOW
NO INTERRUPT; EDGE OCCURS OUTSIDE WINDOW
NO INTERRUPT; EDGE OCCURS OUTSIDE WINDOW
TMR_WID
00
04
02
06

Figure 20-9: Watchdog Period Mode Timing

NOTE: SCLK in the ***Watchdog Period Mode Timing*** figure is SCLK0_0.

To check only the upper limit on period (the t_{MAX} value, not the t_{MIN} value), program t_{MIN} as 0 or 1. For details, refer to the ***Windowed Watchdog Period Mode Interpretation*** table.

Table 20-9: Windowed Watchdog Period Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
0 or 1	Anything ≥ 2	Pulse Period ≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse period crosses Pmax (Period Register) value	No Error Interrupt
> 1 but ≤ Period -1	Anything ≥ 2	Pulse Period ≤ TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY < Pulse Period ≤TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) value	No Error Interrupt

Table 20-9: Windowed Watchdog Period Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
≥ Period	-	Pulse Period < TMR_PER	Undefined	Undefined	No Error Interrupt
		Pulse Period ≥ TMR_PER	Undefined	Undefined	b#11 Error Type
-	0 or 1	-	Undefined	Undefined	b#10 Error Type

Pin Interrupt (PININT) Mode

In PININT mode, any active edges on either the `TIMER_TMR[n]` pin or the `TIMER_ACI[n]` pin can cause an edge-based interrupt, if enabled. (The timer uses the `TIMER_TMR[n]_CFG.TINSEL` register to select the pin). The event on the input pin can set the `TIMER_DATA_ILAT.TMR[nn]` bit and issue a system interrupt request. Program the `TIMER_TMR[n]_CFG.PULSEHI` bit to change active edge polarity.

Since the interrupt request is generated in the SCLK0_0 clock domain, the width of the input signal must be more than one SCLK0_0 period. Along with generating the interrupt request, the timer also generates a trigger pulse (configured using the `TIMER_TRG_MSK` register). Due to the configuration of polarity, glitches can cause the generation of an undesired interrupt request at the input. To avoid this problem, programs must ensure that interrupt requests are unmasked only after configuring the desired polarity.

External Clock (EXTCLK) Mode

The timer uses EXTCLK mode, sometimes referred to as the counter mode, to count external events (signal edges), on either the `TIMER_TMR[n]` or `TIMER_ACI[n]` input pin. The timer works as a counter clocked by an external source (the signal at the pin), which can be asynchronous to SCLK0_0. The current count in the `TIMER_TMR[n]_CNT` register represents the number of leading-edge events detected. The `TIMER_TMR[n]_PER` register is programmed with the value of the maximum timer external count before stopping or issuing an interrupt request or trigger.

The `TIMER_TMR[n]_CFG.PULSEHI` bit determines the polarity of the leading edge on the input pin. The timer uses the `TIMER_TMR[n]_CFG.TINSEL` bit to select whether the event is counted on the `TIMER_TMR[n]` or on the `TIMER_ACI[n]` pin. The `TIMER_STAT_ILAT.TMR[nn]` and `TIMER_ERR_TYPE` bits are set if *one* of these conditions is met:

- `TIMER_TMR[n]_CNT` wraps around from 0xFFFF FFFF to 0
- The period = 0 at startup
- `TIMER_TMR[n]_CNT` register rolls over (from count = period to count = 0x1)

The `TIMER_TMR[n]_WID` and `TIMER_TMR[n]_DLY` registers are unused in this mode and must not be written.

The ***EXTCLK Mode Control Flow*** figure shows a flow diagram for EXTCLK mode.

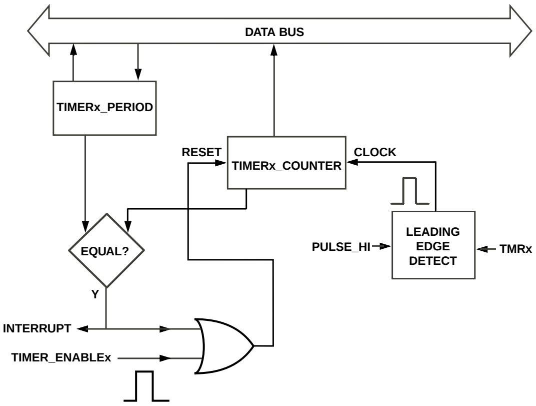


Figure 20-10: EXTCLK Mode Control Flow

The waveform applied to the input pin is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet. Program the period to any value from 1 to (2^{32} – 1), inclusive.

After the timer has started, it resets the `TIMER_TMR[n]_CNT` register to 0x0 and then waits for the first leading edge on the input pin. This edge causes `TIMER_TMR[n]_CNT` to be incremented to the value 0x1, and every subsequent leading edge increments it by one. After the `TIMER_TMR[n]_CNT` register reaches the value programmed in the `TIMER_TMR[n]_PER` register, the corresponding `TIMER_DATA_ILAT` bit is set, and an interrupt and trigger are both generated (if enabled). The next leading-edge reloads the `TIMER_TMR[n]_CNT` register with 0x1, and the timer continues counting until it is disabled.

GP Timer Programming Concepts

Using the features, operating modes, and event control for the GP timer to their greatest potential requires an understanding of some GP timer-related concepts.

Setting Up Constantly Changing Timer Conditions

This task shows how to use different period, pulse width, and delay settings for each of the first three timer periods after the timer starts.

1. Program the first set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.
2. Enable the timer using the `TIMER_RUN` register.
3. Immediately program the second set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values, as needed.
4. Wait for the first timer interrupt request.
5. Program the third set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.

Each new setting is then programmed when the preceding timer interrupt request is received.

Configuring, Enabling, and Disabling One or More Timers

1. Configure the relevant timers for the operating mode and other properties using the `TIMER_TMR[n]_CFG` register.
2. Write a 1 to the representative `TIMER_RUN.TMR[nn]` bit. Or, use the `TIMER_RUN_SET` register to avoid disturbing the settings of other timers that are not going through configuration.

   The timer is enabled and operating.
3. To stop one or more timers, first program the `TIMER_STOP_CFG` register to determine whether to stop immediately or gracefully upon receiving a stop command.

   ADDITIONAL INFORMATION: PWMOUT modes are the only modes where a timer can be configured for graceful termination.
4. Write a 0 to the representative `TIMER_RUN.TMR[nn]` bits to stop the timer according to their `TIMER_STOP_CFG` settings. Alternately, write a 1 to the appropriate `TIMER_RUN_CLR.TMR[nn]` bits to avoid disturbing the settings of other timers that are not terminating.

   The timers stop.

Configuring Timer Data and Status Interrupts

1. Configure the `TIMER_TMR[n]_CFG.IRQMODE` bit field with the desired interrupt properties.
2. Unmask the interrupt source at the system event controller.
3. Set the `TIMER_TMR[n]_CFG.IRQMODE` field but leave the interrupt masked at the system level to poll the `TIMER_DATA_ILAT.TMR[nn]` bit of the timer without generating an interrupt.
4. Use the `TIMER_STAT_IMSK` register to generate interrupt requests by overflow or error conditions (incorrect programming values). The timer uses the `TIMER_STAT_ILAT.TMR[nn]` bits to report interrupt errors, when the timer status interrupt source is unmasked at the system event controller.
5. To poll the `TIMER_STAT_ILAT.TMR[nn]` bit of the timer without generating an interrupt, unmask the corresponding bit in the `TIMER_STAT_IMSK` register, but leave the interrupt masked at the system level.

Using the Timer Broadcast Feature

The broadcast feature provides a means to update period, width, and delay registers simultaneously across more than one timer.

1. Enable the appropriate broadcast bits (`TIMER_TMR[n]_CFG.BPEREN`, `TIMER_TMR[n]_CFG.BWIDEN are TIMER_TMR[n]_CFG.BDLYEN`) for the timers involved in the broadcast. The use of these bits depends on which broadcast registers the timer uses (`TIMER_BCAST_PER`, `TIMER_BCAST_WID`, or `TIMER_BCAST_DLY`).

2. Program the `TIMER_BCAST_PER` register (for example), to broadcast the period setting across the multiple timers enabled.

   The enabled timers load their `TIMER_TMR[n]_PER` registers with the value specified in the `TIMER_BCAST_PER` register.

3. Repeat Step 2 as needed for the `TIMER_BCAST_WID` and `TIMER_BCAST_DLY` register settings.

Timer Illegal States

The following sections use these definitions:

- Startup. The first clock period during which the timer counter is running after the timer is started by writing the `TIMER_RUN` register.
- Rollover. The time when the current count in `TIMER_TMR[n]_CNT` matches the value in `TIMER_TMR[n]_PER` and the counter is reloaded with the value 1.
- Overflow. The timer counter was incremented instead of doing a rollover when it was holding the maximum count value of 0xFFFF FFFF. The counter does not have a large enough range to express the next greater value and so it erroneously loads a new value of 0x0000 0000.
- Unchanged. No new error.

When the `TIMER_ERR_TYPE` register is designated unchanged, it displays the previously reported error code orb# 00 when there has been no error since this timer was enabled.

When the `TIMER_STAT_ILAT` register is unchanged, it reads 0 when there has been no error or overflow since this timer was enabled. Or, it reads 0 if software has performed a W1C to clear any previous error. If software has not acknowledged a previous error, the `TIMER_STAT_ILAT` register reads 1. Software can read the `TIMER_STAT_ILAT` register to check for errors. If a particular bit of a timer is set in this register, software can then read the `TIMER_ERR_TYPE` register for more information. Once detected, software can W1C the appropriate `TIMER_STAT_ILAT` bit to acknowledge the error.

Read the following tables as:

- In mode __ at event __,
- if `TIMER_TMR[n]_PER` is __ and `TIMER_TMR[n]_WID` is __ and `TIMER_TMR[n]_DLY` is __,
- then `TIMER_ERR_TYPE` is __ and `TIMER_STAT_ILAT` is __.

Startup error conditions do not prevent the timer from starting. Similarly, overflow and rollover error conditions do not stop the timer. Illegal cases can cause unwanted behavior of the `TIMER_TMR[n]` pin.

NOTE: For PININT mode, the timer does not use error functionality.

Continuous PWMOUT Mode

Table 20-10: Startup Event

`TIMER_TMR[n]_PER`	`TIMER_TMR[n]_DLY`	`TIMER_TMR[n]_WID`	`TIMER_TMR[n]_WID` + `TIMER_TMR[n]_DLY`	`TIMER_ERR_TYPE`	`TIMER_STAT_ILAT` (if enabled)
≤ 1	Anything other than period[8]	Anything	Anything	b#10	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤ PERIOD	Unchanged	Unchanged
	Anything including 0	Anything including 0	> PERIOD	Unchanged[9] (Detected at rollover)	Unchanged (Detected at rollover)
	Anything	Anything	> $2^{32} - 1$	b#11	Set
	=Period	=0	=Period	No error	Unchanged (Detected at rollover)

Table 20-11: Rollover Event

`TIMER_TMR[n]_PER`	`TIMER_TMR[n]_DLY`	`TIMER_TMR[n]_WID`	`TIMER_TMR[n]_WID` + `TIMER_TMR[n]_DLY`	`TIMER_ERR_TYPE`	`TIMER_STAT_ILAT` (if enabled)
≥ 1	Anything	Anything	Anything	b#10[timer running at SCLK0_0] b#11 [timer running at ALT_CLKx]	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤PERIOD	Unchanged	Unchanged
	Anything including 0, excluding TMR_PER value	Anything >0	>PERIOD	b#11	Set
	Anything	Anything	> $2^{32} - 1$	b#11	Set
	= Period[10]	=0	=Period	b#11	Set
	>Period	=0	>Period	Unchanged	Unchanged

Table 20-12: Overflow Event (On TMR_PER Register Programming Error Only)

`TIMER_TMR[n]_PER`	`TIMER_TMR[n]_DLY`	`TIMER_TMR[n]_WID`	`TIMER_TMR[n]_WID` + `TIMER_TMR[n]_DLY`	`TIMER_ERR_TYPE`	`TIMER_STAT_ILAT` (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

Single Pulse PWMOUT Mode

For single pulse PWMOUT mode, there are no rollover events.

Table 20-13: Startup Event

`TIMER_TMR[n]_PER`	**`TIMER_TMR[n]_DLY`**	**`TIMER_TMR[n]_WID`**	**`TIMER_TMR[n]_WID`** + **`TIMER_TMR[n]_DLY`**	**`TIMER_STAT_ILAT`** (if enabled)	**`TIMER_STAT_ILAT`** (if enabled)
N/A	Anything	== 0	Anything	b#11[11]	Set
N/A	Anything including 0	≥1	$> 2^{32} - 1$	Unchanged	Unchanged
N/A	Anything including 0	≥1	$> 2^{32} - 1$	b#11	Set

Table 20-14: Overflow Event (On another error, such as DELAY + WIDTH ≥ $2^{32} - 1$)

	`TIMER_TMR[n]_DLY`		**`TIMER_TMR[n]_WID`** + **`TIMER_TMR[n]_DLY`**		**`TIMER_STAT_ILAT`** (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

WIDCAP Mode

For WIDCAP mode, the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers are read-only and the `TIMER_TMR[n]_DLY` register is not used. Therefore, no startup or rollover errors are possible.

Table 20-15: Overflow Event

`TIMER_TMR[n]_PER`	**`TIMER_TMR[n]_DLY`**	**`TIMER_TMR[n]_WID`**	**`TIMER_TMR[n]_WID`** + **`TIMER_TMR[n]_DLY`**	**`TIMER_ERR_TYPE`**	**`TIMER_STAT_ILAT`** (if enabled)
Anything	N/A	Anything	N/A	b#01	Set

EXTCLK Mode

Table 20-16: Startup Event

`TIMER_TMR[n]_PER`	**`TIMER_TMR[n]_DLY`**	**`TIMER_TMR[n]_WID`**	**`TIMER_TMR[n]_WID`** + **`TIMER_TMR[n]_DLY`**	**`TIMER_ERR_TYPE`**	**`TIMER_STAT_ILAT`** (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 20-17: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 20-18: Overflow Event (On TMR_PER Register = 0 Only)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	N/A	N/A	b#01	Set

WATCHDOG Events

Table 20-19: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[12]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything < PERIOD	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 20-20: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[10]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 20-21: Overflow Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	N/A	N/A	b#01	Set

ADSP-SC58x TIMER Register Descriptions

General-Purpose Timer Block (TIMER) contains the following registers.

Table 20-22: ADSP-SC58x TIMER Register List

Name	Description
`TIMER_BCAST_DLY`	Broadcast Delay Register
`TIMER_BCAST_PER`	Broadcast Period Register
`TIMER_BCAST_WID`	Broadcast Width Register
`TIMER_DATA_ILAT`	Data Interrupt Latch Register
`TIMER_DATA_IMSK`	Data Interrupt Mask Register
`TIMER_ERR_TYPE`	Error Type Status Register
`TIMER_RUN`	Run Register
`TIMER_RUN_CLR`	Run Clear Register
`TIMER_RUN_SET`	Run Set Register
`TIMER_STAT_ILAT`	Status Interrupt Latch Register
`TIMER_STAT_IMSK`	Status Interrupt Mask Register
`TIMER_STOP_CFG`	Stop Configuration Register
`TIMER_STOP_CFG_CLR`	Stop Configuration Clear Register
`TIMER_STOP_CFG_SET`	Stop Configuration Set Register
`TIMER_TMR[n]_CFG`	Timer n Configuration Register
`TIMER_TMR[n]_CNT`	Timer n Counter Register
`TIMER_TMR[n]_DLY`	Timer n Delay Register
`TIMER_TMR[n]_PER`	Timer n Period Register
`TIMER_TMR[n]_WID`	Timer n Width Register
`TIMER_TRG_IE`	Trigger Slave Enable Register
`TIMER_TRG_MSK`	Trigger Master Mask Register

Broadcast Delay Register

For timers with `TIMER_TMR[n]_CFG.BDLYEN` enabled, a write to the `TIMER_BCAST_DLY` register concurrently updates the delay (`TIMER_TMR[n]_DLY`) registers of only those timers. A read of the `TIMER_BCAST_DLY` register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_DLY` register.

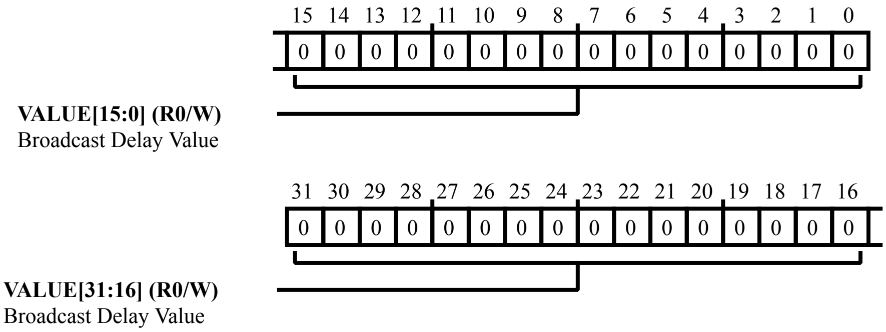

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R0/W)
Broadcast Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R0/W)
Broadcast Delay Value

Figure 20-11: TIMER_BCAST_DLY Register Diagram

Table 20-23: TIMER_BCAST_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Delay Value. A write to the `TIMER_BCAST_DLY.VALUE` bit field concurrently updates the delay (`TIMER_TMR[n]_DLY`) registers of only those timers. A read of the `TIMER_BCAST_DLY.VALUE` bit field returns 0x0000 0000, and no bus error is generated.

Broadcast Period Register

For timers with `TIMER_TMR[n]_CFG.BPEREN` enabled, a write to the `TIMER_BCAST_PER` register concurrently updates the period (`TIMER_TMR[n]_PER`) registers of only those timers. A read of `TIMER_BCAST_PER` returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_PER` register.

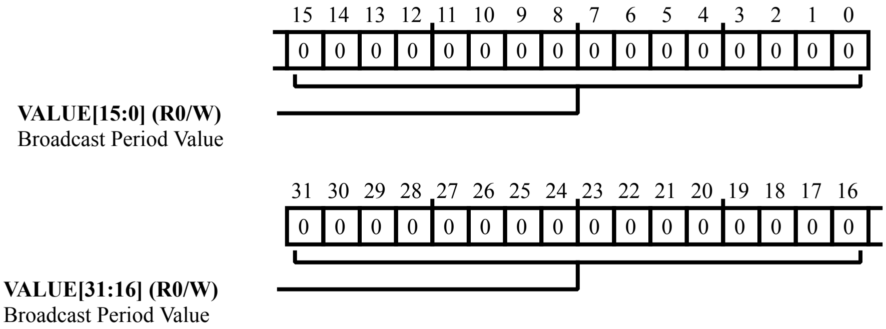

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R0/W)
Broadcast Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R0/W)
Broadcast Period Value

Figure 20-12: TIMER_BCAST_PER Register Diagram

Table 20-24: TIMER_BCAST_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Period Value. A write to the `TIMER_BCAST_PER.VALUE` bit field concurrently updates the period (`TIMER_TMR[n]_PER`) registers of only those timers. A read of the `TIMER_BCAST_PER.VALUE` bit fields returns 0x0000 0000, and no bus error is generated.

Broadcast Width Register

For timers with `TIMER_TMR[n]_CFG.BWIDEN` enabled, a write to the `TIMER_BCAST_WID` register concurrently updates the width (`TIMER_TMR[n]_WID`) registers of only those timers. A read of the `TIMER_BCAST_WID` register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMR[n]_WID` register.

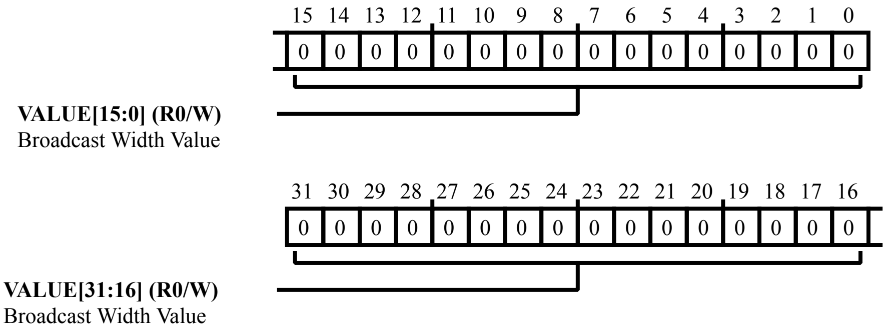

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R0/W)
Broadcast Width Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R0/W)
Broadcast Width Value

Figure 20-13: TIMER_BCAST_WID Register Diagram

Table 20-25: TIMER_BCAST_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Width Value. A write to the `TIMER_BCAST_WID.VALUE` bit field concurrently updates the width (`TIMER_TMR[n]_WID`) registers of only those timers. A read of the `TIMER_BCAST_WID.VALUE` bit field returns 0x0000 0000, and no bus error is generated.

Data Interrupt Latch Register

The TIMER_DATA_ILAT holds the latched interrupt status for interrupt requests that have been unmasked (enabled) by the TIMER_DATA_IMSK register and generated according to the conditions selected by the `TIMER_TMR[n]_CFG.IRQMODE` bits. If a bit in TIMER_DATA_ILAT is already set and the corresponding interrupt is masked in TIMER_DATA_IMSK, the latch holds its old value, leaving the interrupt request asserted until it is reset by software with a W1C operation.

Note that interrupt service routines (ISRs) should clear the appropriate bits in TIMER_DATA_ILAT before returning from the ISR, to ensure that the interrupt is not re-issued. To make sure that no timer event is missed, the latch should be reset at the very beginning of the ISR when in EXTCLK mode.

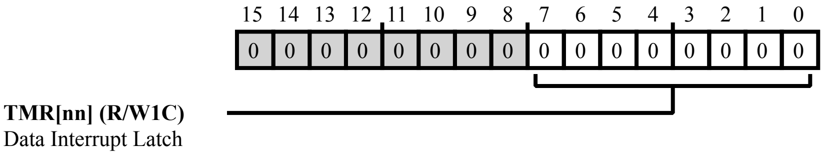


Figure 20-14: TIMER_DATA_ILAT Register Diagram

Table 20-26: TIMER_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Data Interrupt Latch. For all `TIMER_DATA_ILAT.TMR[nn]` bits, status of =0 indicates no interrupt is latched, and status of =1 indicates a latched interrupt (indicating an unmasked interrupt request from a timer with a condition matching the one selected with corresponding `TIMER_TMR[n]_CFG.IRQMODE` bit has occurred).

Data Interrupt Mask Register

Each timer may generate a unique processor data interrupt request signal. The `TIMER_DATA_IMSK` register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the `TIMER_DATA_IMSK` register is 0xFFFF, masking these interrupts after reset.

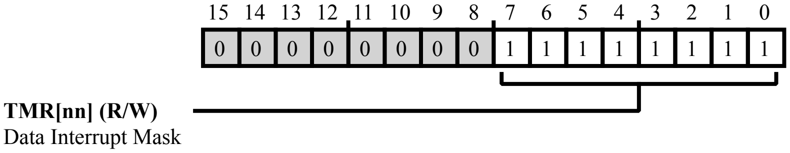


Figure 20-15: TIMER_DATA_IMSK Register Diagram

Table 20-27: TIMER_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Data Interrupt Mask. For all `TIMER_DATA_IMSK.TMR[nn]` bits, write =0 unmasks (enables) the corresponding data interrupt request, and write =1 masks (disables) the corresponding data interrupt request.

Error Type Status Register

The TIMER_ERR_TYPE register contains error type status bits for each timer. These bits indicate the type of error (if any) in a running timer. This register is read-only. These status bits are cleared at reset and when a particular timer is enabled.

Each time an error request interrupt is latched in the TIMER_STAT_ILAT register, the corresponding TERRx bits in the TIMER_ERR_TYPE register are loaded with a code that identifies the type of error that was detected. This status value is held until the next error or until a particular timer is restarted. No bus error is generated if a write is performed on the TIMER_ERR_TYPE register.

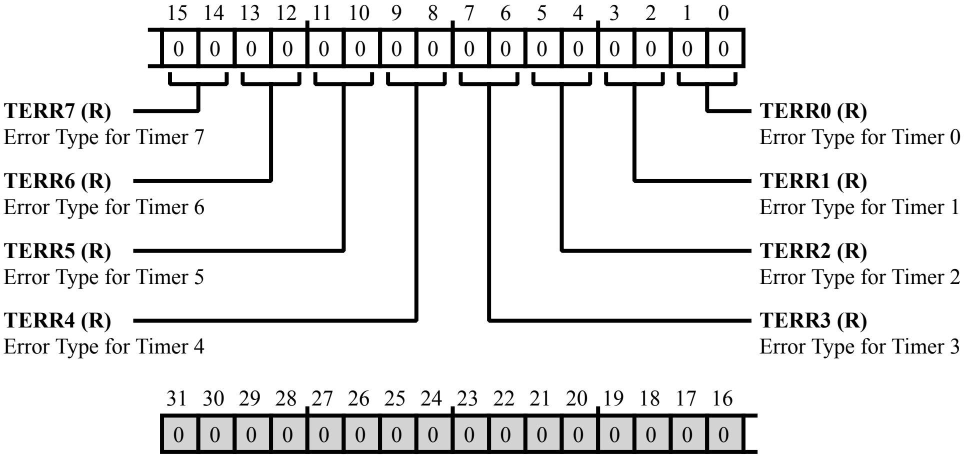

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TERR7 (R)
Error Type for Timer 7
TERR6 (R)
Error Type for Timer 6
TERR5 (R)
Error Type for Timer 5
TERR4 (R)
Error Type for Timer 4
TERR0 (R)
Error Type for Timer 0
TERR1 (R)
Error Type for Timer 1
TERR2 (R)
Error Type for Timer 2
TERR3 (R)
Error Type for Timer 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 20-16: TIMER_ERR_TYPE Register Diagram

Table 20-28: TIMER_ERR_TYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:14 (R/NW)	TERR7	Error Type for Timer 7.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
13:12 (R/NW)	TERR6	Error Type for Timer 6.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Table 20-28: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/NW)	TERR5	Error Type for Timer 5.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
9:8 (R/NW)	TERR4	Error Type for Timer 4.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
7:6 (R/NW)	TERR3	Error Type for Timer 3.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
5:4 (R/NW)	TERR2	Error Type for Timer 2.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
3:2 (R/NW)	TERR1	Error Type for Timer 1.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
1:0 (R/NW)	TERR0	Error Type for Timer 0.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Run Register

The `TIMER_RUN` allows all timers to be enabled simultaneously, permitting them to run synchronously. For each timer, there is a single start/stop control bit. Writing a 1 to this bit starts the corresponding timer; writing a 0 stops the timer with mechanism specified in the timer stop configuration `TIMER_STOP_CFG` register.

The start/stop control bits can be set/reset individually or in any combination. While starting or stopping one particular timer directly with this register, software must perform a read-modify write, so the bits corresponding to other timers remain unchanged. To avoid this need, software can use the `TIMER_RUN_CLR` register.

Reading the `TIMER_RUN` register shows the start status for the corresponding timer. A 1 indicates that the timer is running.

If a timer is in run state (corresponding run bit is =1), a software write of 1 in this bit does not have any effect on the timer state. The write does not result in restarting the timer.

Note that the `TIMER_RUN` register is not used in PININT mode. PININT mode starts as soon as the `TIMER_TMR[n]_CFG.TMODE` bits are set to 111.

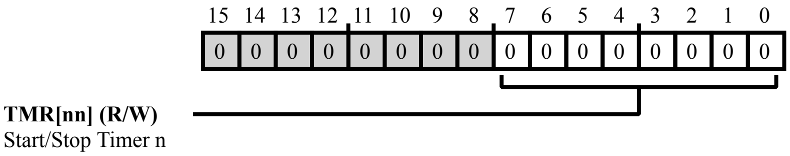


Figure 20-17: TIMER_RUN Register Diagram

Table 20-29: TIMER_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Start/Stop Timer n. For all `TIMER_RUN.TMR[nn]` bits, write =0 for stop, and write =1 for start. Read =1 when timer is running.

Run Clear Register

The `TIMER_RUN_CLR` register is an alias register, providing a mechanism to clear a specific start/stop bit in the `TIMER_RUN` register without affecting other bits in `TIMER_RUN`. To stop a particular timer, software must write a 1 into the corresponding `TIMER_RUN_CLR` bit. Writing a 0 has no effect. Because `TIMER_RUN_CLR` is a write-only register, the result of any write to this register must be checked by reading the `TIMER_RUN` register. A read of the `TIMER_RUN_CLR` returns 0x0000.

Note that the stopping mechanism of a timer may be abrupt or graceful (after completion of current waveform period) depending on the selection in the `TIMER_STOP_CFG` register.

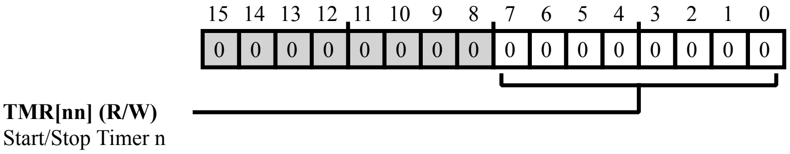


Figure 20-18: TIMER_RUN_CLR Register Diagram

Table 20-30: TIMER_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	RUN Clear Alias. For all `TIMER_RUN_CLR.TMR[nn]` bits, write =0 has no effect, and write =1 for stop (clearing the corresponding in start/stop bit in the `TIMER_RUN` register). Using `TIMER_RUN_CLR` to clear start/stop bits permits stopping specific timers without influencing run status of other timers.

Run Set Register

The TIMER_RUN_SET register is an alias register, providing a mechanism to set a specific start/stop bit in the TIMER_RUN register without affecting other bits in TIMER_RUN. To start a particular timer, software must write a 1 into the corresponding TIMER_RUN_SET bit. Writing a zero has no effect. For an example, to start timer 3 without affecting any other timer, write 0x0008 into TIMER_RUN_SET. Because TIMER_RUN_SET is a write-only register, the result of any write to this register must be checked by reading the TIMER_RUN register. A read of the TIMER_RUN_SET returns 0x0000.

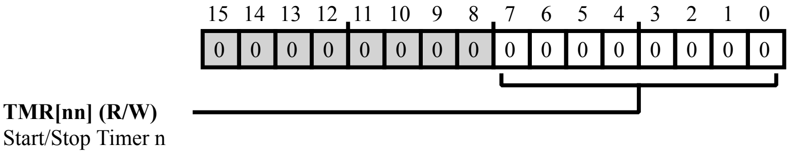


Figure 20-19: TIMER_RUN_SET Register Diagram

Table 20-31: TIMER_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	RUN Set Alias. For all `TIMER_RUN_SET.TMR[nn]` bits, write =0 has no effect, and write =1 for start (setting the corresponding start/stop bit in the TIMER_RUN register). Using TIMER_RUN_SET to set start/stop bits permits starting specific timers without influencing the run status of other timers.

Status Interrupt Latch Register

The `TIMER_STAT_ILAT` holds the latched interrupt status for error interrupt requests, indicating a timer overflow condition or indicating that prohibited programming has occurred for a timer. These interrupt status bits are sticky and are W1C. The bits in the `TIMER_STAT_ILAT` register provide information regarding each timer interrupt request source.

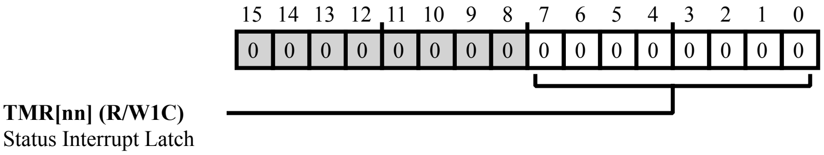


Figure 20-20: TIMER_STAT_ILAT Register Diagram

Table 20-32: TIMER_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Status Interrupt Latch. For all `TIMER_STAT_ILAT.TMR[nn]` bits, status of 0 indicates no error interrupt request is latched, and status of 1 indicates a timer counter overflow or programming error interrupt request is latched.

Status Interrupt Mask Register

While each timer may generate a status interrupt request, these requests are OR'ed to generate a single status interrupt signal to the system event controller. The `TIMER_STAT_IMSK` register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the `TIMER_STAT_IMSK` register is 0xFFFF, masking these interrupts after reset.

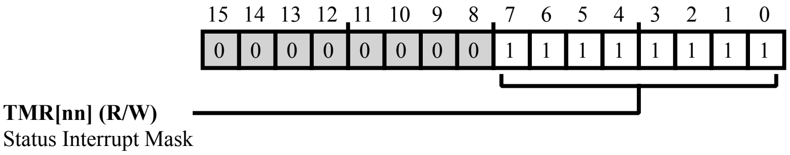


Figure 20-21: TIMER_STAT_IMSK Register Diagram

Table 20-33: TIMER_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Status Interrupt Mask. For all `TIMER_STAT_IMSK.TMR[nn]` bits, write =0 unmasks (enables) the corresponding status interrupt request, and write =1 masks (disables) the corresponding status interrupt request.

Stop Configuration Register

The `TIMER_STOP_CFG` register selects the stop mode for each timer. Timers may be stopped abruptly (immediate halt - all modes) or gracefully in PWMOUT modes (single pulse and continuous). The halt is achieved through either a write =0 to the corresponding bit in `TIMER_RUN` or a write =1 to the corresponding bit in `TIMER_RUN_CLR`. A read of `TIMER_STOP_CFG` returns the last value written.

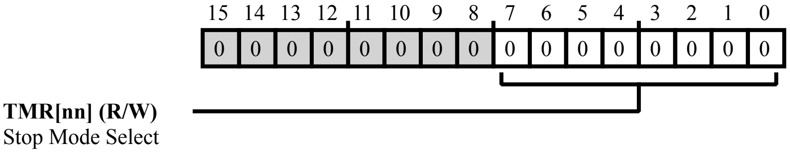


Figure 20-22: TIMER_STOP_CFG Register Diagram

Table 20-34: TIMER_STOP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Stop Mode Select. For all `TIMER_STOP_CFG.TMR[nn]` bits, write =0 for graceful termination (PWMOUT modes only), and write =1 for abrupt (immediate halt) on stop.

Stop Configuration Clear Register

This is an alias register, providing a mechanism to clear a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To clear a bit in TIMER_STOP_CFG, software must write a 1 to the corresponding bit of TIMER_STOP_CFG_CLR register. Writing a zero has no effect. Because the TIMER_STOP_CFG_CLR register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_CLR register returns 0x0000.

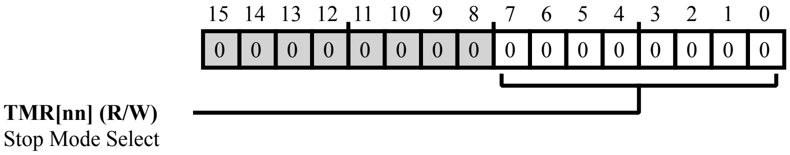


Figure 20-23: TIMER_STOP_CFG_CLR Register Diagram

Table 20-35: TIMER_STOP_CFG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	STOP_CFG Clear Alias. For all `TIMER_STOP_CFG_CLR.TMR[nn]` bits, write =0 has no effect, and write =1 for graceful stop in PWMOUT modes (clearing the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_CLR to clear stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Stop Configuration Set Register

This is an alias register, providing a mechanism to set a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To set a bit in the TIMER_STOP_CFG register, software must write a 1 to the corresponding bit of the TIMER_STOP_CFG_SET register. Writing a zero has no effect. Because the TIMER_STOP_CFG_SET register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_SET register returns 0x0000.

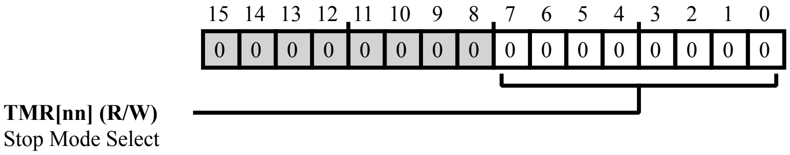


Figure 20-24: TIMER_STOP_CFG_SET Register Diagram

Table 20-36: TIMER_STOP_CFG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	STOP_CFG Set Alias. For all `TIMER_STOP_CFG_SET.TMR[nn]` bits, write =0 has no effect, and write =1 for abrupt stop (setting the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_SET to set stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Timer n Configuration Register

Each timer has a `TIMER_TMR[n]_CFG` register that specifies its operating mode. Only write to a `TIMER_TMR[n]_CFG` register when the corresponding timer is not running.

After disabling a timer operating in PWMOUT mode, verify that the timer has stopped running by checking the start/stop status of the timer in the `TIMER_RUN` register before writing to the timer's `TIMER_TMR[n]_CFG` register.

Note that a timer's `TIMER_TMR[n]_CFG` register may be read at any time.

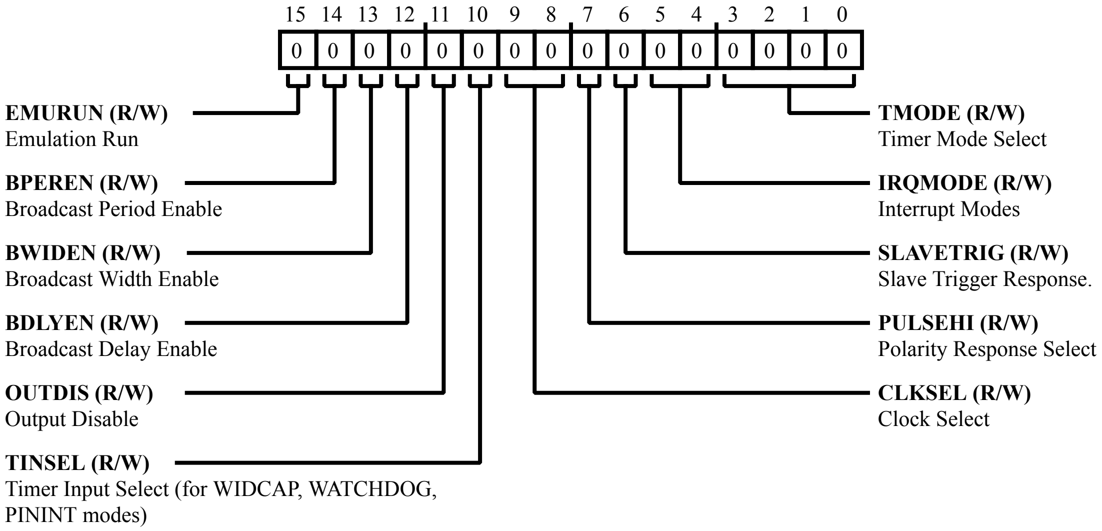

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EMURUN (R/W)
Emulation Run
BPEREN (R/W)
Broadcast Period Enable
BWIDEN (R/W)
Broadcast Width Enable
BDLYEN (R/W)
Broadcast Delay Enable
OUTDIS (R/W)
Output Disable
TINSEL (R/W)
Timer Input Select (for WIDCAP, WATCHDOG, PININT modes)
TMODE (R/W)
Timer Mode Select
IRQMODE (R/W)
Interrupt Modes
SLAVETRIG (R/W)
Slave Trigger Response.
PULSEHI (R/W)
Polarity Response Select
CLKSEL (R/W)
Clock Select

Figure 20-25: TIMER_TMR[n]_CFG Register Diagram

Table 20-37: TIMER_TMR[n]_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EMURUN	Emulation Run. The `TIMER_TMR[n]_CFG.EMURUN` bit causes the timer to run (count) during emulation.	
		0	Stop Timer During Emulation
		1	Run Timer During Emulation
14 (R/W)	BPEREN	Broadcast Period Enable. The `TIMER_TMR[n]_CFG.BPEREN` bit enables updates to the `TIMER_TMR[n]_PER` register simultaneously across more than one timer.	
		0	Disable Broadcast to PER Register
		1	Enable Broadcast to PER Register

Table 20-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	BWIDEN	Broadcast Width Enable. The `TIMER_TMR[n]_CFG.BWIDEN` bit enables updates to the `TIMER_TMR[n]_WID` register simultaneously across more than one timer.	
		0	Disable Broadcast to WID Register
		1	Enable Broadcast to WID Register
12 (R/W)	BDLYEN	Broadcast Delay Enable. The `TIMER_TMR[n]_CFG.BDLYEN` bit enables updates to the `TIMER_TMR[n]_DLY` register simultaneously across more than one timer.	
		0	Disable Broadcast to DLY Register
		1	Enable Broadcast to DLY Register
11 (R/W)	OUTDIS	Output Disable. The `TIMER_TMR[n]_CFG.OUTDIS` bit enables or disables the timer pin output buffer.	
		0	Enable TMR Pin Output Buffer
		1	Disable TMR Pin Output Buffer
10 (R/W)	TINSEL	Timer Input Select (for WIDCAP, WATCHDOG, PININT modes).	
		0	Use TMR Pin Input
		1	Use TMR Alternate Capture Input
9:8 (R/W)	CLKSEL	Clock Select. The `TIMER_TMR[n]_CFG.CLKSEL` bit field selects the TIMER clock to use.	
		0	Use SCLK0_0
		1	Use TMR_ALT_CLK0 as TMR Clock
		3	Use TMR_ALT_CLK1 as TMR Clock
7 (R/W)	PULSEHI	Polarity Response Select. The `TIMER_TMR[n]_CFG.PULSEHI` bit defines specific behaviors of the timer based on the operating mode. For more information, see the specific operating mode in the Programming Concepts section.	
		0	Negative Response or Pulse. A Negative Edge Response or Negative Action Pulse on the TMR pin.
		1	Positive Response or Pulse. A Positive Edge Response or Positive Action Pulse on the TMR pin.

Table 20-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SLAVETRIG	Slave Trigger Response.. The TIMER_TMR[n]_CFG.SLAVETRIG bit controls the trigger response. The trigger pulse has no effect (to stop or start the timer) if the timer is already in the requested state.	
		0	Pulse Stops Timer if it is Running
		1	Pulse Starts Timer if it is Stopped
5:4 (R/W)	IRQMODE	Interrupt Modes. The TIMER_TMR[n]_CFG.IRQMODE bit field selects the interrupt request mode. Note that any mismatched combination of the TIMER_TMR[n]_CFG.IRQMODE and the TIMER_TMR[n]_CFG.TMODE bits results in no interrupt being generated. In WIDCAP modes, the position of the interrupt is controlled with the TIMER_TMR[n]_CFG.TMODE bit, and the TIMER_TMR[n]_CFG.IRQMODE bit is ignored. Whenever an interrupt is generated, a trigger master pulse is also generated, if enabled in the TIMER_TRG_MSK register.	
		0	Active Edge Mode. The timer generates an interrupt at every active edge. The active edge polarity depends on the state of the TIMER_TMR[n]_CFG.PULSEHI bit. Valid for PININT mode only.
		1	Delay Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_DLY register. This mode is valid for all PWMOUT modes.
		2	Width Plus Delay Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_WID register plus the value in the TIMER_TMR[n]_DLY register. (PWMOUT modes only) The timer generates an interrupt if the deasserting edge is within the specified window. (WATCHDOG modes only)
		3	Period Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_PER register. (Continuous PWMOUT and EXTCLK modes only) The timer generates an interrupt if the deasserting edge is outside the specified window.(WATCHDOG modes only)

Table 20-37: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	TMODE	Timer Mode Select. The `TIMER_TMR[n]_CFG.TMODE` bit field selects the operating mode of each timer.	
		0-7	Idle Mode
		8	Period Watchdog Mode
		9	Width Watchdog Mode
		10	Measurement Report at Asserting Edge of Waveform
		11	Measurement Report at Deasserting Edge of Waveform
		12	Continuous PWMOUT Mode
		13	Single Pulse PWMOUT Mode
		14	EXTCLK Mode
		15	PININT (pin interrupt) Mode

Timer n Counter Register

The `TIMER_TMR[n]_CNT` register holds the current timer count. After enabling, the count is re-initialized to either 0x0 or 0x1, depending on the configuration and mode. The `TIMER_TMR[n]_CNT` register is read-only and may be read at any time (whether the timer is running or stopped). Reading the `TIMER_TMR[n]_CNT` register returns an atomic 32-bit value.

Depending on the timer operation mode, the counter increment can be clocked by a number of sources, including SCLK0_0, the TMR or alternate capture input pins, `TIMER_ACLK[n]`. The counter retains its value after the timer is disabled.

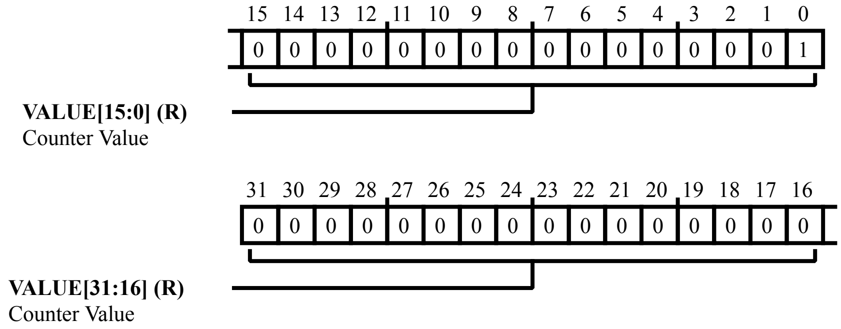

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
VALUE[15:0] (R)
Counter Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Counter Value

Figure 20-26: TIMER_TMR[n]_CNT Register Diagram

Table 20-38: TIMER_TMR[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value. The `TIMER_TMR[n]_CNT.VALUE` bit field holds the current timer count.

Timer n Delay Register

The `TIMER_TMR[n]_DLY` register holds the delay value for the corresponding timer. This register's use is based on the selected timer mode.

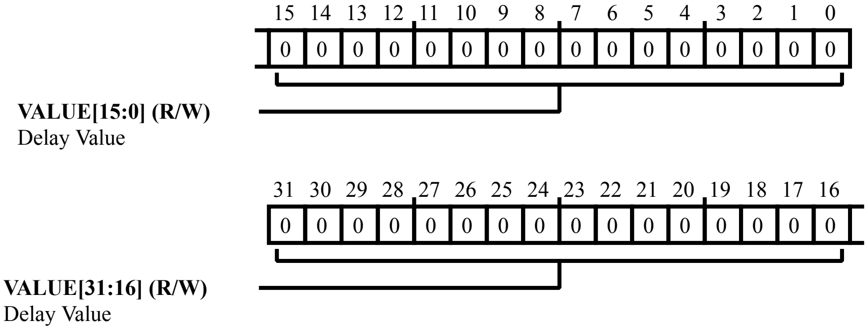

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Delay Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Delay Value

Figure 20-27: TIMER_TMR[n]_DLY Register Diagram

Table 20-39: TIMER_TMR[n]_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Value. The `TIMER_TMR[n]_DLY.VALUE` bit field holds the delay value for the corresponding timer.

Timer n Period Register

The TIMER_TMR[n]_PER register holds the period value for the corresponding timer. This register's use is based on the selected timer mode.

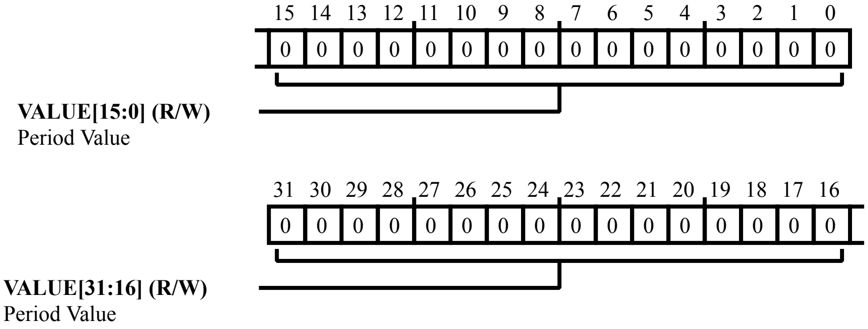

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Period Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Period Value

Figure 20-28: TIMER_TMR[n]_PER Register Diagram

Table 20-40: TIMER_TMR[n]_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Period Value. The TIMER_TMR[n]_PER.VALUE bit field holds the period value for the corresponding timer.

Timer n Width Register

The `TIMER_TMR[n]_WID` register holds the width value for the corresponding timer. This register's use is based on the selected timer mode.

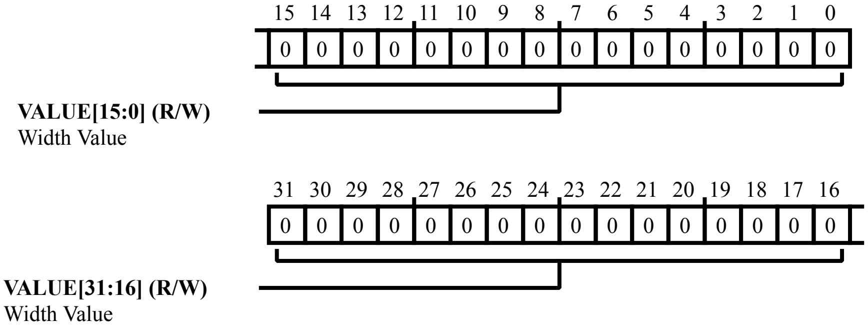

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Width Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Width Value

Figure 20-29: TIMER_TMR[n]_WID Register Diagram

Table 20-41: TIMER_TMR[n]_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Width Value. The `TIMER_TMR[n]_WID.VALUE` bit field holds the width value for the corresponding timer.

Trigger Slave Enable Register

As a trigger slave, each timer can generate a unique data trigger pulse signal. The `TIMER_TRG_IE` contains trigger input enable bits for these signals, disabling or enabling the triggers as programmed. The reset value of the `TIMER_TRG_IE` register is 0xFFFF, masking these triggers after reset.

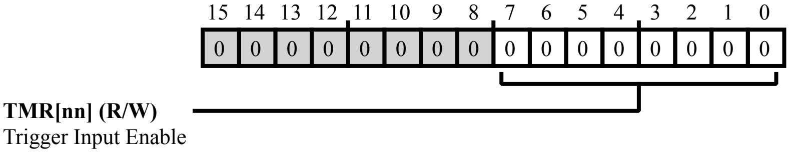


Figure 20-30: TIMER_TRG_IE Register Diagram

Table 20-42: TIMER_TRG_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Input Enable. For all `TIMER_TRG_IE.TMR[nn]` bits, write =0 disables the corresponding trigger input, and write =1 enables the corresponding trigger input.

Trigger Master Mask Register

As a trigger master, each timer can generate a unique data trigger pulse signal. The `TIMER_TRG_MSK` register contains a trigger mask for these outputs, masking (disabling) or unmasking (enabling) the triggers as programmed. The reset value of the `TIMER_TRG_MSK` register is 0xFFFF, masking these triggers after reset.

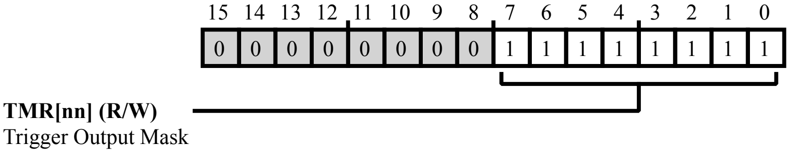


Figure 20-31: TIMER_TRG_MSK Register Diagram

Table 20-43: TIMER_TRG_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Output Mask. For all `TIMER_TRG_MSK.TMR[nn]` bits, write =0 unmasks (enables) the corresponding data trigger output, and write =1 masks (disables) the corresponding data trigger output.

21 Watchdog Timer (WDOG)

The processor has a 32-bit watchdog timer that can be used to verify system reliability by generating an event to the processor core when the watchdog expires before software updates it.

For the ADSP-SC58x processors, a system typically assigns WDG0 to Core ID=0 and WDG1 to Cores ID=1 and ID=2. The system clock (SCLK0_0) clocks the watchdog timers.

WDOG Features

The watchdog timer has the following features:

- Programmable 32-bit watchdog count value
- 8-bit disable bit pattern
- General-purpose core event generation

The watchdog timer can supervise system software stability by periodically reloading it to prevent expiration of the downward-counting timer (such that the count never becomes 0). When used in this fashion, an expiring timer can indicate the system software is not running normally.

Expiration of the WDOG counter generates a general-purpose interrupt, which can be used in a variety of ways:

- as an interrupt vector sent through the System Event Controller (SEC) to the core to be serviced by a handler function, providing full software control of device resources (for example, GPIO management and reset control).
- as a fault condition through the SEC to provide hardware-automated:
  - signalling of the fault condition on external pins to the system,
  - system reset requests to the Reset Control Unit (RCU), and/or
  - trigger outputs (SEC0_FAULT trigger master) through the Trigger Routing Unit (TRU) to initiate activities in a variety of potential trigger slaves (for example, GPIO control).

To facilitate debugging, the watchdog timer does not decrement (even if enabled) when the processor is in emulation mode.

WDOG Functional Description

When enabled, the 32-bit watchdog timer counts downward every SCLK0_0 cycle. If it reaches zero, the watchdog expiration event is generated, which can be used in many ways.

To start the watchdog timer:

1. Program the watchdog timeout period (in SCLK0_0 cycles) in the `WDOG_CNT` register. With the watchdog disabled, this write also preloads the `WDOG_STAT` register.
2. Enable the watchdog timer by writing any value other than 0xAD to the `WDOG_CTL.WDEN` field.

Once the watchdog is enabled, writes to the `WDOG_CNT` register are ignored. The counter begins decrementing, and the current counter value can be read from the 32-bit `WDOG_STAT` register at any time.

To prevent the counter from expiring, software must "kick" the watchdog by writing any value to the `WDOG_STAT` register while the current count is non-zero. While the value written is irrelevant and ignored, this action resets the current counter in the `WDOG_STAT` register to the programmed `WDOG_CNT` value, and decrementing continues. The internal counter continues decrementing until it reaches zero, at which point the expiration event is generated, and the `WDOG_CTL.WDRO` rollover bit is set.

Watchdog operation continues in this manner unless disabled by explicitly writing 0xAD to the `WDOG_CTL.WDEN` field.

The watchdog expiration event can be used in a variety of ways, as listed below.

- The watchdog expiration event itself is one of numerous interrupt sources that is managed by the System Event Controller. Like other peripheral sources, this event can be used to cause a vector to a handler function that executes based on interrupt priority.
- The watchdog expiration event can be used to initiate automated hardware response through the SEC Fault Interface (SFI).

For this, the WDOG expiry has to be configured as the fault source in the SEC. Then the response to the WDOG expiry can be set to any of the below:

- Signalling through the external fault pin.
- System reset
- Trigger outputs to numerous potential trigger slaves.

For further details on how watchdog expiration event can be used with the SFI, see Configuring the WDOG Expiry Event to Issue a System Reset.

ADSP-SC58x WDOG Register List

The Watchdog Timer unit (WDOG) provides a software-based watchdog timer that can improve system reliability by generating an event to the processor core if the watchdog expires before being updated by software. A set of registers governs WDOG operations. For more information on WDOG functionality, see the WDOG register descriptions.

Table 21-1: ADSP-SC58x WDOG Register List

Name	Description
`WDOG_CNT`	Count Register
`WDOG_CTL`	Control Register
`WDOG_STAT`	Watchdog Timer Status Register

ADSP-SC58x WDOG Interrupt List

Table 21-2: ADSP-SC58x WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	WDOG0_EXP	WDOG0 Expiration	Level	
4	WDOG1_EXP	WDOG1 Expiration	Level	

WDOG Block Diagram

The *Watchdog Timer Block Diagram* figure shows the detailed block diagram for the watchdog timer.

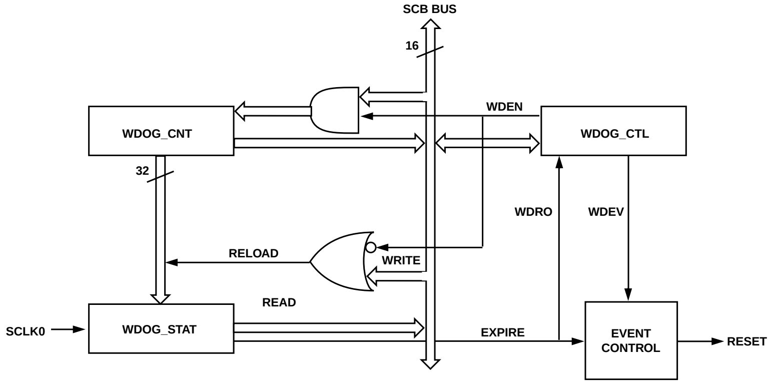


Figure 21-1: Watchdog Timer Block Diagram

Internal Interface

The system clock (SCLK0_0) clocks the watchdog timer. The registers are accessed through the 16-bit peripheral MMR access bus. 32-bit read/write operations always access the 32-bit `WDOG_CNT` and `WDOG_STAT` registers. Hardware ensures that those accesses are atomic. When the counter expires, the WDOG expiration event is generated.

External Interface

The watchdog timer does not directly interact with any external pins.

ADSP-SC58x WDOG Register Descriptions

Watchdog Timer Unit (WDOG) contains the following registers.

Table 21-3: ADSP-SC58x WDOG Register List

Name	Description
`WDOG_CNT`	Count Register
`WDOG_CTL`	Control Register
`WDOG_STAT`	Watchdog Timer Status Register

Count Register

The `WDOG_CNT` register holds the programmable, unsigned count value. A valid write to this register also pre-loads the WDOG counter. For added safety, the `WDOG_CNT` register can be updated only when the WDOG timer is disabled. A write to the `WDOG_CNT` register while the timer is enabled does not modify the contents of this register. This register must be accessed with 32-bit read/writes only.

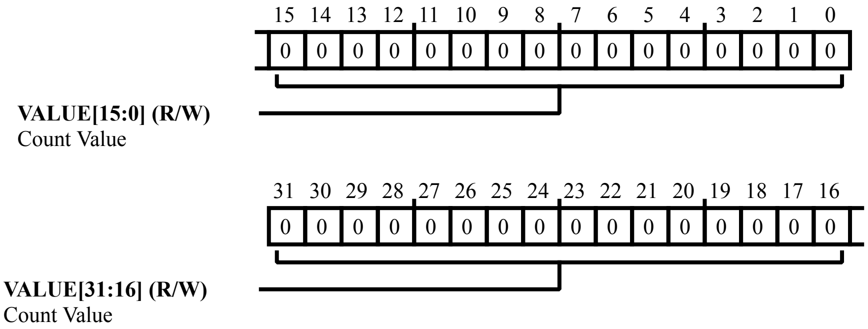

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Count Value

Figure 21-2: WDOG_CNT Register Diagram

Table 21-4: WDOG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count Value. The `WDOG_CNT.VALUE` bit field holds the programmable, unsigned count value.

Control Register

The WDOG_CTL register controls the watchdog timer. This register supports enabling/disabling the watchdog timer and supports checking the timer rollover status. Note that when the processor is in emulation mode, the watchdog timer counter will not decrement even if it is enabled.

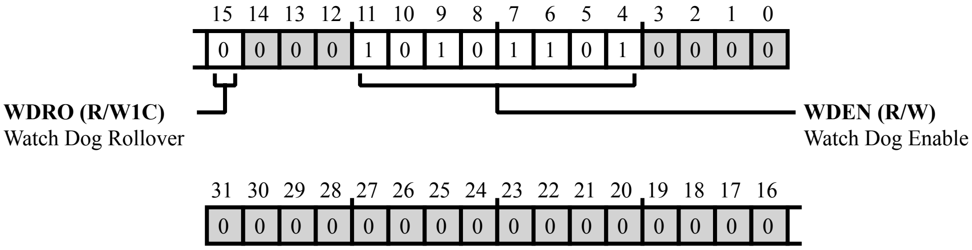

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0
WDRO (R/W1C)
Watch Dog Rollover
WDEN (R/W)
Watch Dog Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 21-3: WDOG_CTL Register Diagram

Table 21-5: WDOG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	WDRO	Watch Dog Rollover. Software can determine whether the timer has rolled over by interrogating the `WDOG_CTL.WDRO` status bit. This is a sticky bit that is set whenever the watch dog timer count reaches 0 and cleared only by disabling the watch dog timer and then writing a 1 to the bit.	
		0	WDT Has Not Expired
		1	WDT Has Expired
11:4 (R/W)	WDEN	Watch Dog Enable. The `WDOG_CTL.WDEN` field is used to enable and disable the watchdog timer. Writing any value other than the disable value into this field enables the watchdog timer. This multi-bit disable key minimizes the chance of inadvertently disabling the watchdog timer.	
		173	Counter Disabled. All other values mean that the counter is enabled.

Watchdog Timer Status Register

The `WDOG_STAT` register contains the current count value of the watchdog timer. Reads of this register return the current count value. When the watchdog timer is enabled, the `WDOG_STAT` register is decremented by 1 on each SCLK0_0 cycle. When the count value reaches 0, the watchdog timer stops counting, and the expiry event is generated. The `WDOG_STAT` register is a 32-bit unsigned system MMR that must be accessed with 32-bit reads and writes.

Values cannot be stored directly in this register but are instead copied from the `WDOG_CNT` register. This copy process can happen in two ways:

- While the watchdog timer is disabled, writing the `WDOG_CNT` register pre-loads the `WDOG_STAT` register.
- While the watchdog timer is enabled, writing the `WDOG_STAT` register loads it with the value in the `WDOG_CNT` register.
- While the watchdog timer is disabled, writing to the `WDOG_STAT` register also reloads it with the value in the `WDOG_CNT` register.

When the processor executes a write (of an arbitrary value) to the `WDOG_STAT` register, the value in the `WDOG_CNT` register is copied into the `WDOG_STAT` register. Typically, software sets the value of `WDOG_CNT` at initialization, then periodically writes to the `WDOG_STAT` register before the watchdog timer expires. This reloads the watchdog timer with the value from `WDOG_CNT` and prevents generation of the expiry event.

If the program does not reload the counter before SCLK0_0 x count register cycles, an expiry event is generated, and the `WDOG_CTL.WDRO` bit is set. When this happens, the counter stops decrementing and remains at zero. If the counter is enabled with a zero loaded to the counter, the `WDOG_CTL.WDRO` bit is set immediately and the counter remains at zero and does not decrement.

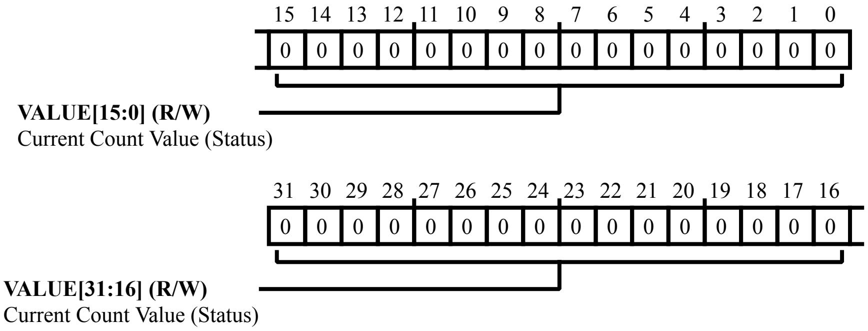


Figure 21-4: WDOG_STAT Register Diagram

Table 21-6: WDOG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Current Count Value (Status). The `WDOG_STAT.VALUE` bit field contains the current count value of the watchdog timer.

22 Real Time Clock (RTC)

The processor contains a Real Time Clock (RTC) which provides a set of digital watch features, including time of day, alarm, and stopwatch countdown. It is typically used to implement either a real-time watch or a life counter which counts the elapsed time since the last system reset. The RTC uses dedicated power supply pins and is independent of any reset, which enables it to maintain functionality even when the rest of the processor is powered down.

RTC Features

The RTC interface has the following features.

- Provides a 1 Hz clock with a second, minute, hour, and day counter (0–32767 days)
- Alarm with a time-of-day interrupt
- Operates on a dedicated supply from an external 3.3 V battery
- Stopwatch function
- Interrupt generation for different events
- Standard two-pin interface with an external 32.768 kHz crystal, 6 pF capacitor on each pin, and a 100 MΩ resistor between the pins
- RTC power switches to the I/O supply when the processor is powered on, saving battery life
- Calibration corrects time once a day; application can use the RTCXTALIN pin to determine calibration settings
- Power down and bus disable

RTC Functional Description

The RTC provides a set of digital watch features to the processor. The RTC external interface consists of two clock pins, which together with the external components form the reference clock circuit for the RTC. It uses an external 32.768 kHz crystal with external capacitor and operates on a dedicated external 3.3 V Lithium coin cell which is never powered off.

The following are functional characteristics of the RTC.

Power supply partitioning

The RTC logic is partitioned between the processor core supply voltage and RTC supply voltage. The core of RTC unit functions on a RTC power supply.

Battery life

To increase the battery life of the external 3.3V cell, most functions reside in the RTC core voltage domain which runs off the processor supply. Only basic clock circuitry resides in the RTC IO voltage.

Reads/writes to 1Hz registers

There is no latency when reading 1 Hz registers, as the values come from the shadow registers.

Writes to the RTC 1 Hz registers are synchronized to the 1 Hz RTC clock.

ADSP-SC58x RTC Register List

The Real-Time Clock (RTC) provides clock-related services, including a set of processor events that can be counted during program execution. A set of registers governs RTC operations. For more information on RTC functionality, see the RTC register descriptions.

Table 22-1: ADSP-SC58x RTC Register List

Name	Description
`RTC_ALM`	RTC Alarm Register
`RTC_CLK`	RTC Clock Register
`RTC_IEN`	Interrupt Enable Register
`RTC_INIT`	RTC Initialization Register
`RTC_INITSTAT`	RTC Initialization Status Register
`RTC_STAT`	RTC Status Register
`RTC_STPWTCH`	RTC Stop Watch Register

ADSP-SC58x RTC Interrupt List

Table 22-2: ADSP-SC58x RTC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
165	RTC0_EVT	RTC0 Event	Level	

ADSP-SC58x RTC Trigger List

Table 22-3: ADSP-SC58x RTC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
134	`RTC0_EVT`	RTC0 Event	Level

Table 22-4: ADSP-SC58x RTC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

RTC Definitions

The following definitions are helpful when using the RTC module.

RTC HV

The RTC HV is the portion of the RTC which runs on 3.3 V domain and contains the logic for RTC counting/ clock functionality

RTC LV

The RTC LV operates on the core voltage supply and provides the control/access functionality and user interface to the application

Oscillator

The oscillator is the heart of the RTC. It works with the dedicated external crystal to generate a 32768 Hz signal which is divided down to a 1Hz clock used to operate the rest of the RTC

RTC Signal Descriptions

The RTC signals are listed in the following table.

Table 22-5: RTC Signal Descriptions

Signal Name	Signal Description
XTALIN	Pin of the dedicated 32.768KHz external crystal
XTALOUT	Pin of the dedicated 32.768KHz external crystal
VDDRTC	Power supply pin (from Lithium Coin Cell) for RTC module

RTC Architectural Concepts

The key use of the RTC is to provide the time keeping function and maintain the time and date in an accurate and reliable manner with minimal power consumption. In addition to time keeping it also provides the stopwatch and alarm features. The RTC uses the internal counters to keep the time of the day in terms of seconds, minutes, hours and days. This data is enough for the user application to extract the date and time information from the RTC.

Interrupts can be issued periodically, either every second, every minute, every hour, or every day. Each of these interrupts can be independently controlled. It is the responsibility of the program to set the correct time by a software write into the `RTC_CLK` register. Once set, the counters maintain time as long as the RTC supply is valid. The RTC provides two alarm features, programmed with the RTC alarm register (`RTC_ALM`). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified. The second alarm feature allows the application to specify a day as well as a time.

The RTC also provides a stopwatch function that acts as a countdown timer. The application can program a second count into the RTC stopwatch count register (`RTC_STPWTCH`). When the stopwatch interrupt is enabled and the specified number of seconds has elapsed, the RTC generates an interrupt.

RTC Block Diagram

The RTC block diagram provides a top level block diagram of the elements used. The main blocks of the RTC are the individual counters, the alarm register, and the event control.

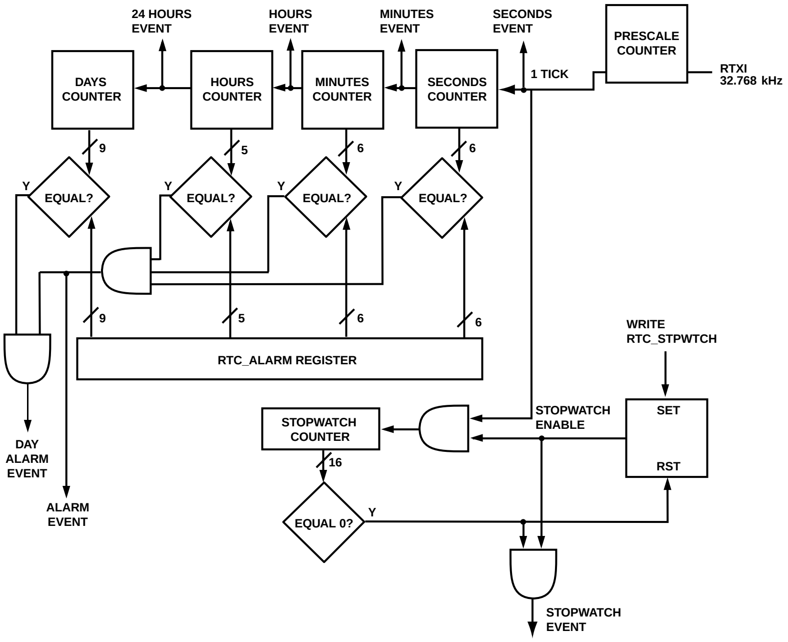

24 HOURS EVENT
HOURS EVENT
MINUTES EVENT
SECONDS EVENT
PRESCALE COUNTER
DAYS COUNTER
HOURS COUNTER
MINUTES COUNTER
SECONDS COUNTER
1 TICK
RTXI 32.768 kHz
9
5
6
6
Y
EQUAL?
Y
EQUAL?
Y
EQUAL?
Y
EQUAL?
9
5
6
6
WRITE RTC_STPWTCH
RTC_ALARM REGISTER
STOPWATCH COUNTER
STOPWATCH ENABLE
SET
RST
DAY ALARM EVENT
16
ALARM EVENT
EQUAL 0?
Y
STOPWATCH EVENT

Figure 22-1: RTC Block Diagram

Power Supply Partitioning

The RTC logic is partitioned between the processor core supply voltage and RTC supply voltage. The core of RTC unit functions on a RTC power supply so that the RTC can maintain the time even if the processor power is off. When the core supply voltage is absent, interrupts are ignored.

When the processor's I/O supply is above a certain threshold, the RTC switches to the I/O supply to conserve battery power.

NOTE: Battery power supply can operate the RTC when the I/O voltage is turned off.

The RTC is partitioned into two blocks. The counting and clock function is provided in the I/O voltage domain (VDD_EXT), while the control and access function and the user interface are provided in the core voltage domain. The RTC I/O operates on a dedicated power supply provided by the external 3.3 V (nominal) lithium coin cell. The RTC also has the ability to switch to the I/O supply (VDD_EXT). The RTC LV operates on the nominal core voltage supply (VDD_INT). The interface between both blocks is provided by a set of level shifters. The partitioning at chip level is shown in the ***Power Supply Partition*** figure.

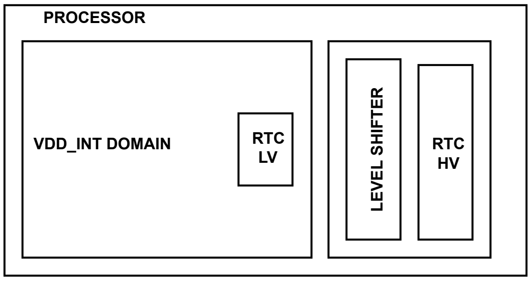


Figure 22-2: Power Supply Partition

Battery Life

To increase the battery life of the external 3.3V cell, most functions reside in the RTC core voltage domain which runs off the processor supply. Only basic clock circuitry resides in the RTC IO voltage. The RTC module provides the following features to extend battery life.

- The seconds, minutes, hours and days counters reside inside the RTC IO voltage.
- The alarm register and comparators reside inside the RTC IO voltage. This allows programs to power down the rest of the chip without the alarm being reset.
- The programmable interface registers, through which the application reads or writes the current time and alarm settings, are part of the RTC core voltage. In order to set the current time and/or alarm, software writes into the shadow registers in the RTC core voltage. The data is then transferred into the corresponding register in the RTC IO voltage by hardware.
- The RTC core voltage runs primarily on the processor's peripheral clock while the RTC IO voltage runs primarily on a self generated 1 Hz clock. The synchronization circuitry sits inside the RTC core voltage.
- The stopwatch circuitry is inside the RTC core voltage and operates on a 1 Hz clock, level shifted from the RTC IO voltage.

Writes to the 1 Hz Registers

Writes to the RTC 1 Hz registers are synchronized to the 1 Hz RTC clock. When setting the time of day, do not factor in the delay when writing to the RTC 1 Hz registers. The most accurate method of setting the RTC is to monitor the seconds (1 Hz) event flag or to program an interrupt for this event and then write the current time to the RTC status register (RTC_STAT) in the interrupt service routine (ISR). The new value is inserted ahead of the incremented value. Hardware adds one second to the written value (with appropriate carries into minutes, hours and days) and loads the incremented value at the next 1 Hz tick, when it represents the then-current time.

Writes posted at any time are properly synchronized to the 1 Hz clock. Writes complete at the rising edge of the 1 Hz clock. A write posted just before the 1 Hz tick may not be completed until the 1 Hz tick one second later.

Reads From the 1 Hz Registers

There is no latency when reading 1 Hz registers, as the values come from the shadow registers. The shadows are updated and ready for reading by the time any RTC interrupts or event flags for that second are asserted. Once the internal core logic completes its initialization sequence after SCLK0_0 starts, there is no point in time when it is unsafe to read the 1 Hz registers (for synchronization reasons). The registers always return coherent values, although the values may be unknown.

RTC Operating Modes

The following sections provide information on the operating modes available to the real-time clock.

Alarm

The RTC provides two alarm features, programmed with the RTC alarm register (RTC_ALM). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified.

Day Alarm

The second alarm feature allows the application to specify a day as well as a time. When the day alarm interrupt is enabled, the RTC generates an interrupt on the day and time specified. The alarm interrupt and day alarm interrupt can be enabled or disabled independently.

Stopwatch

The RTC stopwatch count register (RTC_STPWTCH) contains the countdown value for the stopwatch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if enabled) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new value is written to the RTC_STPWTCH register. Once running, the counter may be overwritten with a new value. This allows the stopwatch to be used as a watchdog timer with a precision of one second.

The stopwatch can be programmed to any value between 0 and (2^{16} – 1) seconds, which is a range of 18 hours, 12 minutes, and 15 seconds. Typically, software should wait for a 1 Hz tick, then write the RTC_STPWTCH register. One second later, the RTC_STPWTCH value changes to the new value and begins decrementing. Because the register write occupies nearly one second, the time from writing a value of N until the stopwatch interrupt is nearly N + 1 seconds. To produce an exact delay, software can compensate by writing N – 1 to get a delay of nearly N seconds. This implies that a delay of 1 second with the stopwatch cannot be achieved. Writing a value of 1 immediately after a 1 Hz tick results in a stopwatch interrupt nearly two seconds later. To wait one second, software should just wait for the next 1 Hz tick

Digital Watch Mode

The primary function of the RTC is to maintain an accurate day count and time of day. The RTC accomplishes this by means of four counters:

- 60-second counter
- 60-minute counter
- 24-hour counter
- 32768-day counter

The RTC provides a set of digital watch features. The internal oscillator generates a 32768 Hz signal using the crystal which is scaled down to 1 Hz and used to clock the second, minute, hour and day counters. The 32768 day counter increments each day at midnight (during the change from 23:59:59). The counter operates on the RTC supply (either the external battery or I/O supply) and is active irrespective of the status of the processor core supply (VDD_INT). When the processor core and I/O supply are valid then the following occurs.

- The current time is updated every second into the RTC clock register (`RTC_CLK`)
- Interrupts can be issued periodically every second, every minute, every hour or every day

Each of the interrupts can be independently controlled, and are described in the RTC Event Control section.

It is the responsibility of the program to set the correct time by a software write into the `RTC_CLK` register. Once set, the counters maintain time as long as the RTC supply is valid.

Calibration for Accuracy

To guard against the possibility of long term (> 1 day) errors, the RTC provides a calibration feature using 4 bits of the `RTC_INIT` register (not available for the stopwatch function).

This is a simple a time correction at the end of every day (when the clock register changes from a Day:Hour:Min:Sec value of XXX:23:59:59 to YYY:00:00:00). It functions by adding or subtracting an integer number of seconds (to a maximum of 7) from the start of the next day, to correct accumulated time error over the course of the previous day. The number of seconds that are added or subtracted is defined in the `RTC_INIT.CAL` bit field.

As an example, if there is a –50 ppm error in the 1 Hz frequency, this translates into a 86400 ´ 50 ppm seconds (+4.32 seconds) error. That is at time 00:00:00, RTC time is 4.32 seconds ahead of actual time. The RTC can correct this by adding 4 seconds (if 4 is the value written into the calibration register) to the time at 00:00:00. Therefore, from 23:59:59, the timer counter jumps directly to 00:00:04, (there is no 00:00:00 to 00:00:03 time occurrences). At the instant it jumps to 00:00:04, the error reduces to +0.32 seconds over the course of the day, which is only 3.7 ppm.

As a second example, if there is a +50 ppm error in the 1 Hz frequency, this also translates into 86400 ´ 50 ppm seconds (–4.32 seconds) error. In this case the time must be subtracted. This is corrected in the RTC by counting

00:00:00 to 00:00:03 twice, so that the time is effectively subtracted. As soon as the RTC reaches 00:00:04, the error reduces to –0.32 seconds over the course of the previous day and accumulated error is minimized.

When the RTC is powered up for the first time, the calibration values are written once to ensure proper function (if they are not to be used, write 0000). These register bits are sticky, which means that once set, they retain their value irrespective of the status of the core power supply.

The addition or subtraction of time can only be in integer multiples of seconds. Zero to seven seconds can be added or subtracted using 4 bits. The MSB indicates addition (0) or subtraction (1). The three LSBs indicate number of seconds (0 – 7 represented by their binary 3 bit equivalents). Because the clock runs at a time period of one second (@ 1 Hz frequency) 0.25 or 0.5 second resolution is not possible.

The calibration technique introduces a guard band for alarm by definition. In case the alarm is set within the duration of the time (Day:00:00:00 to Day:00:00:06) corrected by the calibration register, then it occurs at the nearest corrected time. This is shown in the following two examples.

If the `RTC_INIT.CAL` is 0101, the RTC clock jumps from 23:59:59 to 00:00:05 due to calibration. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

If the `RTC_INIT.CAL` is 1101, then the RTC clock counts from 00:00:00 to 00:00:04 twice and then moves to 00:00:05. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

Accuracy

In order to perform calibration on the bench, use the RTXI pin and check the ppm deviation from 32.768 kHz. This ppm error is the same as in the internal 1Hz clock and the calibration register should be updated with the corresponding values as explained in Calibration for Accuracy above.

Note that total accuracy is ≤ ±35 ppm, < ±1.5 minutes per month of error, inclusive of any inaccuracies of the RTC input crystal at room temperature. This is achieved with a crystal of ±10 ppm error at 25°C. A crystal error of ±20 ppm translates into a maximum inaccuracy of ±45 ppm.

RTC Event Control

The RTC generates multiple events depending on the value on `RTC_CLK`, `RTC_ALM` and `RTC_STPWTCH` registers. It generates an event on each second, minute, hour, day. It also generates an event when an alarm or day alarm or stopwatch expiry occurs.

RTC Events

The RTC can provide interrupts at several programmable intervals

- Per Second
- Per minute (clock counter x:y:z:59)
- Per hour (clock counter x:y:59:59)
- Per day (clock counter x:23:59:59)

- Everyday at a particular time (day alarm)
- On a particular day and time (time of the day alarm)
- Expiry of Stop Watch counter
- When the 1 Hz clock from RTC HV fails

The RTC can also be programmed to provide an interrupt at the completion of all pending writes to any of the RTC 1Hz registers (`RTC_ALM`, `RTC_CLK`, `RTC_INIT` and `RTC_STPWTCH`). Any of these interrupts can be individually enabled/disabled through the bits in `RTC_IEN` register.

Programs can disable all the RTC interrupts when the processor enters emulation space by setting the `RTC_IEN.EMUDIS` bit. Interrupts are not generated even if the individual interrupt enable bits in the `RTC_IEN` register are set.

In the service routine, the `RTC_STAT` register should be read to identify the cause of the interrupt. While reading the status register the RTC automatically clears the respective status bit, ensuring that the cause has been cleared before ending the routine. Note that the pending RTC interrupt is cleared whenever all enabled and set (=1) bits in the `RTC_STAT` register are read, or when all bits in the `RTC_IEN` register corresponding to pending events are cleared (=0).

RTC Status and Error Signals

The RTC contains a status register (`RTC_STAT`) that provides the status on the errors and other events generated by the module. The module captures the 1 Hz clock failures in this register as well. The `RTC_STAT` register contains the RTC event flags and RTC interrupt status. Once set by the event, each bit remains set until cleared by a software read of this register. These sticky bits are independent of the interrupt enable bits in the `RTC_IEN` register. Values are cleared by reading `RTC_STAT` register, except for the `RTC_STAT.WRPEND`, `RTC_STAT.ALM` and `RTC_STAT.DAYALM` bits. Writes to the `RTC_STAT` register have no effect.

RTC Programming Model

The following sections provide basic programming steps for the RTC interface.

Power-Up, Power-Down and Reset

The `RTC_INIT.PWDN` programmable bit is provided to power down the RTC. Power-down is interpreted as a crystal oscillator disable, which reduces power dissipation to only leakage current. Once set or reset, this bit retains its value unless changed, irrespective of the status of core supply.

The inclusion of the power-down bit (`RTC_INIT.PWDN`) as well as the possibility that the RTC may not be used in certain applications introduces specific constraints on the power-up and reset behavior of the RTC. These are described below.

1. When the RTC is powered-up for the first time, it remains in an undefined state until the core powers-up and the corresponding `RTC_INIT.PWDN` bit is written by software. Programs should clear (=0) the `RTC_INIT.PWDN` bit if the RTC function is desired and set it (=1) if it is not.
2. After clearing the `RTC_INIT.PWDN` bit the application must wait at least until the first seconds' event before it writes the timer and alarm registers. This is because the oscillator has a startup time before the clock is generated.

   This sequence applies only to the first time the RTC supply (battery or I/O) is connected. Once the `RTC_INIT.PWDN` bit is set or reset, its value is retained as long as RTC supply (battery or I/O) is valid.
3. After the RTC supply is connected for the first time and the `RTC_INIT.PWDN` bit =0, the application is free to power-up and power-down the core supply any number of times without loss of RTC function (provided the RTC supply, battery or I/O, is valid). Conversely, if the `RTC_INIT.PWDN` bit =1, then the RTC oscillator remains disabled irrespective of the status of the core supply.
4. The current status of the RTC power-down is updated by hardware into the initialization status register (`RTC_INITSTAT.PWDN`). This is useful when the rest of the processor wakes up from power-down and needs to know the status of the RTC.
5. Whenever the processor core wakes up from power-down, the values of the `RTC_CLK`, `RTC_ALM` and `RTC_STPWTCH` registers is zero until the first seconds' event after power-up. At the first seconds' event, an arbitrary value is uploaded into these registers. To put them in a defined state software must write the desired value into these registers. If the `RTC_CLK` and `RTC_ALM` registers have been set before core power-down and subsequent power-up, their values are valid throughout, but can be read by the program only after the first seconds' event after power-up.

Register Access

The interface to the RTC is through a set of memory-mapped registers. The RTC is configured through software and the current state of the RTC is also determined through reads and writes to these registers. Writes of the alarm, clock, stopwatch and initialization registers is performed in a two step sequence.

1. The desired values are programmed into a shadow register in the processor's core VDD_INT domain and operating on the processor's peripheral clock.
2. The contents of the shadow register are synchronized onto the contents of the RTC's internal clock register which operates on the 1 Hz clock in the RTC power domain.

To ensure that writes between the core voltage and RTC voltage domain are properly synchronized, all write commands should be issued immediately after a seconds' event in the RTC status register (`RTC_STAT`). This two step sequence results in a write latency of up to 1 second.

While the write sequence is ongoing, the write pending (`RTC_STAT.WRPEND`) bit is set and is cleared by hardware when the process is complete. Resetting or powering down the peripherals while a write is in progress, (that is when this bit is set) is forbidden. Subsequent writes to the same register before completion of the previous write are ignored.

- Do not attempt write to any of `RTC_CLK`, `RTC_ALM` or `RTC_STPWTCH` registers when the RTC oscillator is powered down or when the `RTC_INIT.RDEN` bit is set.
- During initialization, after a write of the `RTC_INIT` register, make sure that the `RTC_STAT.WRPEND` bit is cleared before attempting writes to other registers.

The `RTC_INIT` register can be written any time. However, programs must ensure that the interrupt enable bits corresponding to Clock, Alarm and Stopwatch features are set only after initializing the `RTC_CLK`, `RTC_ALM` and `RTC_STPWTCH` registers. The `RTC_STAT` and `RTC_INITSTAT` registers can be read any time.

ADSP-SC58x RTC Register Descriptions

Real Time Clock (RTC) contains the following registers.

Table 22-6: ADSP-SC58x RTC Register List

Name	Description
`RTC_ALM`	RTC Alarm Register
`RTC_CLK`	RTC Clock Register
`RTC_IEN`	Interrupt Enable Register
`RTC_INIT`	RTC Initialization Register
`RTC_INITSTAT`	RTC Initialization Status Register
`RTC_STAT`	RTC Status Register
`RTC_STPWTCH`	RTC Stop Watch Register

RTC Alarm Register

The `RTC_ALM` register is programmed by software for the time (in hours, minutes, and seconds) the alarm interrupt occurs. Reads and writes can occur at any time. The alarm interrupt occurs whenever the hour, minute, and second fields first match those of the `RTC_CLK` register. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the `RTC_CLK` status register.

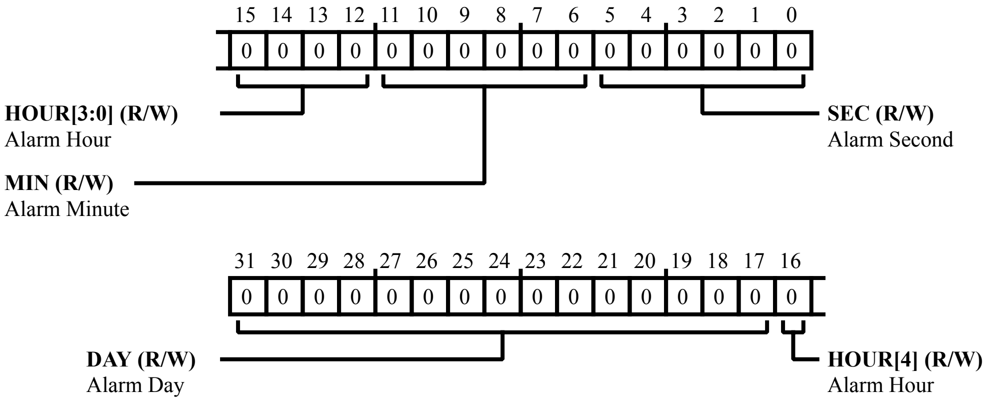

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HOUR[3:0] (R/W)
Alarm Hour
MIN (R/W)
Alarm Minute
SEC (R/W)
Alarm Second
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DAY (R/W)
Alarm Day
HOUR[4] (R/W)
Alarm Hour

Figure 22-3: RTC_ALM Register Diagram

Table 22-7: RTC_ALM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:17 (R/W)	DAY	Alarm Day. The `RTC_ALM.DAY` bit provides the alarm day from 0 to 32767. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the RTC status register.
16:12 (R/W)	HOUR	Alarm Hour. The `RTC_ALM.HOUR` bit field configures the alarm to occur whenever the alarm hour (0 to 23) first matches that of the RTC status register.
11:6 (R/W)	MIN	Alarm Minute. The `RTC_ALM.MIN` bit field configures the alarm to occur whenever the alarm minute (0 to 59) first matches that of the RTC status register.
5:0 (R/W)	SEC	Alarm Second. The `RTC_ALM.SEC` bit field configures the alarm to occur whenever the seconds (0 to 59) first match those of the RTC status register.

RTC Clock Register

The `RTC_CLK` register is used to read or write the current time. It has no reset and an undefined value when the RTC VDD is first powered up. Writing invalid time values is forbidden (for example, an hour value more than 23 and a minute value more than 59). The `RTC_CLK` register is updated every second. If the RTC is already running when the core starts up, the values read from `RTC_CLK` are zero until the first second event comes. In this case, programs must wait for the second event and then read this register.

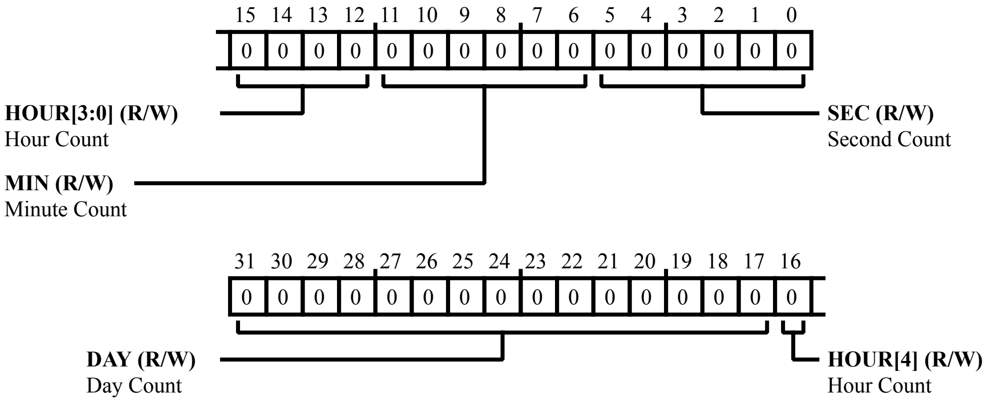

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HOUR[3:0] (R/W)
Hour Count
MIN (R/W)
Minute Count
SEC (R/W)
Second Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DAY (R/W)
Day Count
HOUR[4] (R/W)
Hour Count

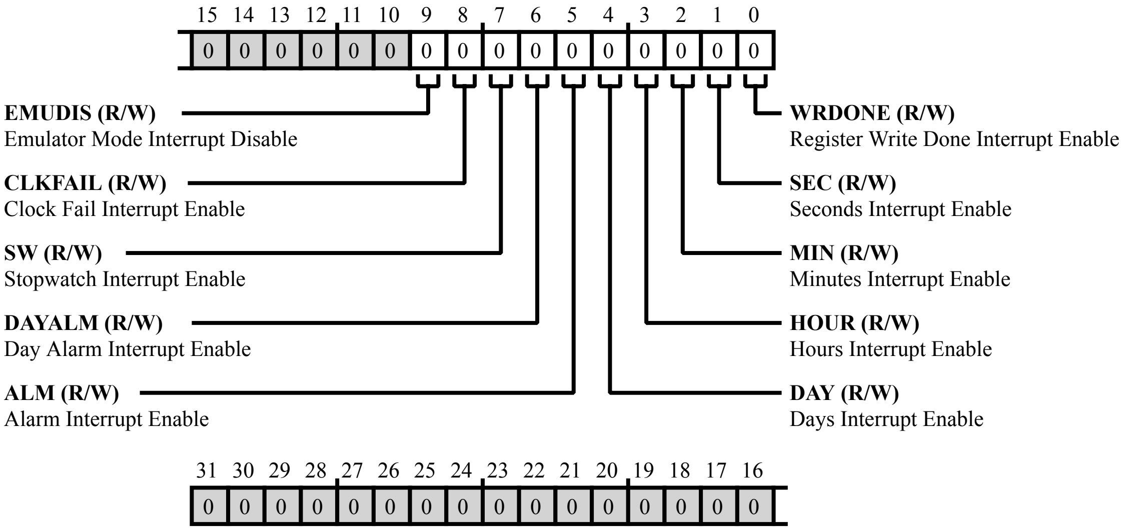
Figure 22-4: RTC_CLK Register Diagram

Table 22-8: RTC_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:17 (R/W)	DAY	Day Count. The `RTC_CLK.DAY` bit provides the day count from 0 to 32767.
16:12 (R/W)	HOUR	Hour Count. The `RTC_CLK.HOUR` bit provides the hour count from 0 to 23.
11:6 (R/W)	MIN	Minute Count. The `RTC_CLK.MIN` bit provides the minute count from 0 to 59.
5:0 (R/W)	SEC	Second Count. The `RTC_CLK.SEC` bit provides the second count from 0 to 59.

Interrupt Enable Register

The RTC_IEN register enables interrupts (when the bits are set) or disables interrupts (when bits are cleared).



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EMUDIS (R/W)
Emulator Mode Interrupt Disable
CLKFAIL (R/W)
Clock Fail Interrupt Enable
SW (R/W)
Stopwatch Interrupt Enable
DAYALM (R/W)
Day Alarm Interrupt Enable
ALM (R/W)
Alarm Interrupt Enable
WRDONE (R/W)
Register Write Done Interrupt Enable
SEC (R/W)
Seconds Interrupt Enable
MIN (R/W)
Minutes Interrupt Enable
HOUR (R/W)
Hours Interrupt Enable
DAY (R/W)
Days Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 22-5: RTC_IEN Register Diagram

Table 22-9: RTC_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	EMUDIS	Emulator Mode Interrupt Disable. The RTC_IEN register disables or enables RTC interrupts in emulation mode. Interrupts are not generated even if individual interrupt enable bits in the RTC_IEN register are set.	
		0	Enable interrupt
		1	Disable interrupt
8 (R/W)	CLKFAIL	Clock Fail Interrupt Enable. The RTC_IEN.CLKFAIL bit enables the RTC 1Hz clock fail interrupt.	
		0	Disable interrupt
		1	Enable interrupt
7 (R/W)	SW	Stopwatch Interrupt Enable. The RTC_IEN.SW bit enables the stopwatch interrupt.	
		0	Disable interrupt
		1	Enable interrupt

Table 22-9: RTC_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	DAYALM	Day Alarm Interrupt Enable. The `RTC_IEN.DAYALM` bit enables the alarm (day, hour, minute, second) interrupt.	
		0	Disable interrupt
		1	Enable interrupt
5 (R/W)	ALM	Alarm Interrupt Enable. The `RTC_IEN.ALM` bit enables the alarm (hour, minute, second) interrupt.	
		0	Disable interrupt
		1	Enable interrupt
4 (R/W)	DAY	Days Interrupt Enable. The `RTC_IEN.DAY` bit enables the days interrupt.	
		0	Disable interrupt
		1	Enable interrupt
3 (R/W)	HOUR	Hours Interrupt Enable. The `RTC_IEN.HOUR` bit enables the hours interrupt.	
		0	Disable interrupt
		1	Enable interrupt
2 (R/W)	MIN	Minutes Interrupt Enable. The `RTC_IEN.MIN` bit enables the minutes interrupt.	
		0	Minute interrupt disabled
		1	Enable interrupt
1 (R/W)	SEC	Seconds Interrupt Enable. The `RTC_IEN.SEC` bit enables the seconds interrupt.	
		0	Disable interrupt
		1	Enable interrupt
0 (R/W)	WRDONE	Register Write Done Interrupt Enable. The `RTC_IEN.WRDONE` bit enables the interrupt for register write completion. The `RTC_IEN.WRDONE` bit is applicable only for the alarm, clock, and stopwatch registers.	
		0	Disable interrupt
		1	Enable interrupt

RTC Initialization Register

The `RTC_INIT` register provides the calibration function, powers down the unit, and disables the output buses.

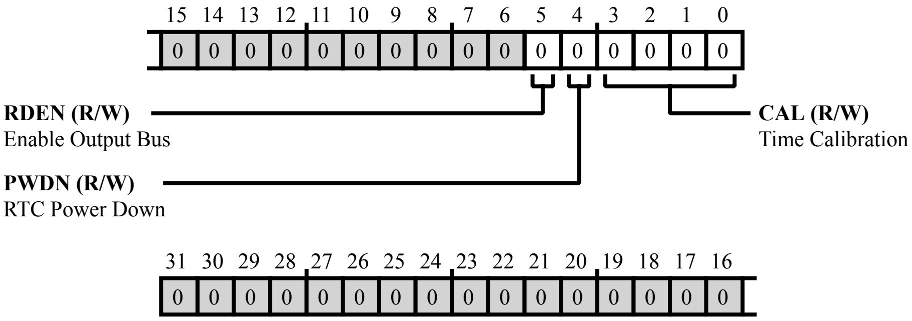

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDEN (R/W)
Enable Output Bus
PWDN (R/W)
RTC Power Down
CAL (R/W)
Time Calibration
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 22-6: RTC_INIT Register Diagram

Table 22-10: RTC_INIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	RDEN	Enable Output Bus. The `RTC_INIT.RDEN` bit disables the output bus.	
		0	Enable output bus
		1	Disable output bus
4 (R/W)	PWDN	RTC Power Down. The `RTC_INIT.PWDN` bit powers down the RTC module. For complete information on this function, see the "Power-Up, Power-Down and Reset" section of this chapter.	
		0	Power up
		1	Power down
3:0 (R/W)	CAL	Time Calibration. The `RTC_INIT.CAL` bit performs calibration the RTC module. For complete information on this function, see the "Calibration for Accuracy" section of this chapter.	

RTC Initialization Status Register

The RTC_INITSTAT register contains values of various status bits which can be used to check the status of RTC when the core comes out of reset.

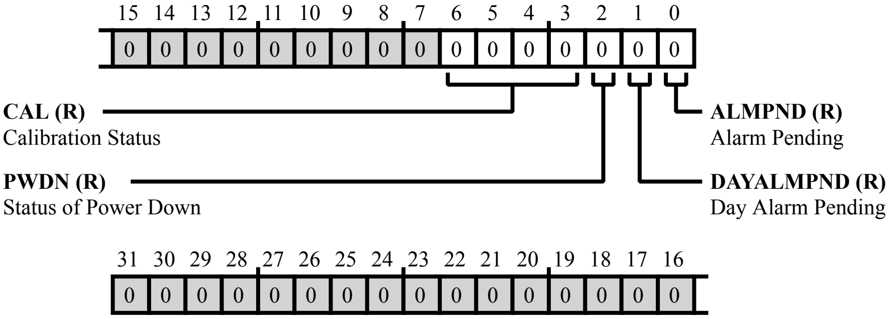

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CAL (R)
Calibration Status
PWDN (R)
Status of Power Down
ALMPND (R)
Alarm Pending
DAYALMPND (R)
Day Alarm Pending
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 22-7: RTC_INITSTAT Register Diagram

Table 22-11: RTC_INITSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6:3 (R/NW)	CAL	Calibration Status.	
2 (R/NW)	PWDN	Status of Power Down. The RTC_INITSTAT.PWDN bit indicates the status of the RTC.	
		0	Oscillator is powered down
		1	Oscillator is running
1 (R/NW)	DAYALMPND	Day Alarm Pending. The RTC_INITSTAT.DAYALMPND bit indicates that an alarm has occurred. This indication is useful when the core has powered down or reset in the middle of operation. This bit is cleared when the RTC_INITSTAT register is read.	
		0	Alarm occurred
		1	No alarm occurred
0 (R/NW)	ALMPND	Alarm Pending. The RTC_INITSTAT.ALMPND bit indicates that an alarm has occurred. This indication is useful when the core has powered down or reset in the middle of operation. This bit is cleared when the RTC_INITSTAT register is read.	
		0	No alarm occurred
		1	Alarm occurred

RTC Status Register

The RTC_STAT register contains the RTC event flags and RTC interrupt status. These bits are sticky. Once set by the event, each bit remains set until cleared by a software read. These sticky bits are independent of the interrupt enable bits in the RTC_IEN register. Values are cleared by reading the RTC_STAT register, except for the RTC_STAT.WRPEND bit, which is read-only. Writes of 0 or 1 to any bit of this register has no effect. This register is cleared at reset.

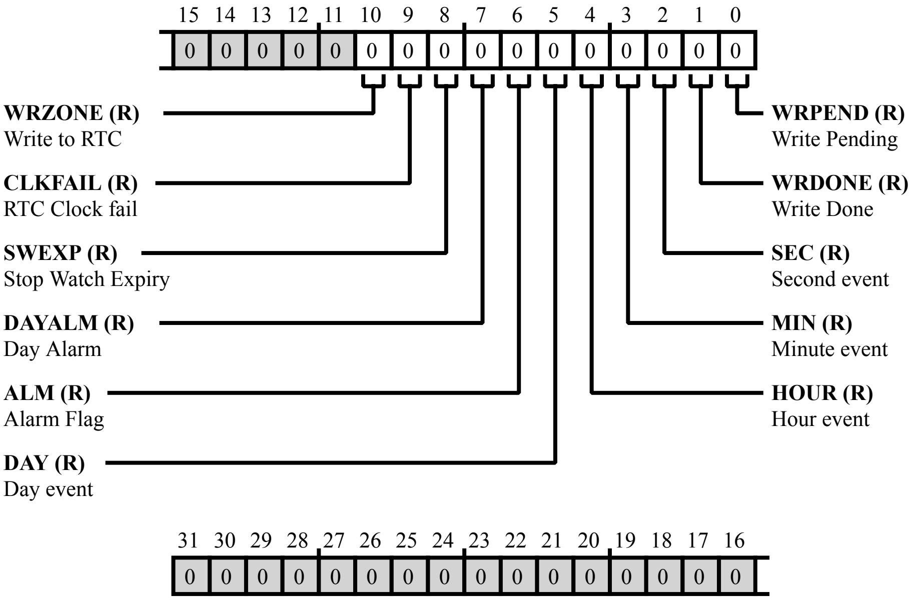

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WRZONE (R)
Write to RTC
CLKFAIL (R)
RTC Clock fail
SWEXP (R)
Stop Watch Expiry
DAYALM (R)
Day Alarm
ALM (R)
Alarm Flag
DAY (R)
Day event
WRPEND (R)
Write Pending
WRDONE (R)
Write Done
SEC (R)
Second event
MIN (R)
Minute event
HOUR (R)
Hour event
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 22-8: RTC_STAT Register Diagram

Table 22-12: RTC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	WRZONE	Write to RTC.	
9 (R/NW)	CLKFAIL	RTC Clock fail. The RTC_STAT.CLKFAIL bit indicates whether the RTC 1 Hz clock is functional. 0 = RTC 1 Hz clock is functional 1 = RTC 1 Hz clock failed	
		0	Clock functioning
		1	Clock failed

Table 22-12: RTC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	SWEXP	Stop Watch Expiry. 1 = stopwatch counter expired 0 = stopwatch counter running	
		0	Counter running
		1	Counter expired
7 (R/NW)	DAYALM	Day Alarm. The `RTC_STAT.DAYALM` bit indicates that a time of day alarm has occurred (=1).	
		0	Alarm did not occur
		1	Alarm occurred
6 (R/NW)	ALM	Alarm Flag. The `RTC_STAT.ALM` bit indicates that an alarm has occurred (=1).	
		0	Alarm did not occur
		1	Alarm occurred
5 (R/NW)	DAY	Day event. The `RTC_STAT.DAY` bit indicates that a day event (clock counter value x:23:59:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
4 (R/NW)	HOUR	Hour event. The `RTC_STAT.HOUR` bit indicates that a hour event (clock counter value x:y:59:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
3 (R/NW)	MIN	Minute event. The `RTC_STAT.MIN` bit indicates that a minute event (clock counter value x:y:z:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
2 (R/NW)	SEC	Second event. The `RTC_STAT.SEC` bit indicates that a second event has occurred (=1).	
		0	Event did not occur
		1	Event occurred

Table 22-12: RTC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	WRDONE	Write Done. The `RTC_STAT.WRDONE` bit shows that the register write is complete. The `RTC_STAT.WRDONE` bit is applicable only for the alarm, clock and stopwatch registers.	
		0	Write is done
		1	Write is not done
0 (R/NW)	WRPEND	Write Pending. The `RTC_STAT.WRPEND` bit shows that a write to the `RTC_CLK`, `RTC_ALM`, `RTC_STPWTCH`, or `RTC_INIT` register is pending. This bit is automatically cleared and set by the hardware.	
		0	No write pending
		1	Write pending

RTC Stop Watch Register

The RTC_STPWTCH register contains the countdown value for the stop watch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if SW_INTEN=1) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new nonzero value is written to the RTC_STPWTCH register. Writing a value of 0 to the running stopwatch forces it to stop; no interrupt is generated in this case. The register can be programmed to any value between 0 and (2^{16-1}) seconds (that is, a range of 18 hours, 12 minutes and 15 seconds).

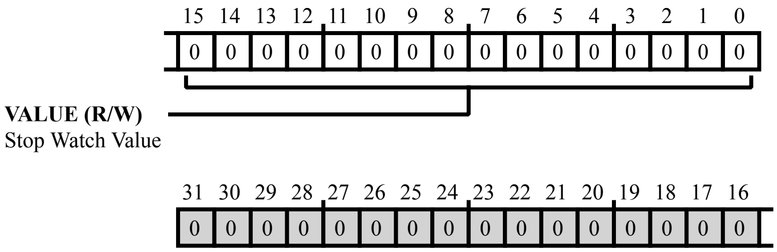

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Stop Watch Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 22-9: RTC_STPWTCH Register Diagram

Table 22-13: RTC_STPWTCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Stop Watch Value. The RTC_STPWTCH register contains the countdown value for the stop watch.

23 General-Purpose Counter (CNT)

The GP counter converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

The GP counter provides flexible ways to establish position information. When used with the GP timer block, the GP counter can allow for the acquisition of coherent position or time stamp information that enables speed calculation.

GP Counter Features

The GP counter includes the following features:

- 32-bit up or down counter
- Quadrature encode mode (Gray code)
- Binary encoder mode
- Alternative frequency-direction mode
- Timed direction and up or down counting modes
- Zero marker or push-button support
- Capture event timing in association with GP Timer
- Boundary comparison and boundary setting features
- M/N frequency scaling of the inputs CUD/CDG

GP Counter Functional Description

The *GP Counter Block Diagram* shows a block diagram of the GP counter. The `CNT_UD` and `CNT_DG` pins accept various forms of incremental inputs. The 32-bit counter processes the inputs. The GP counter uses the `CNT_ZM` pin to sense the pressing of a push button.

NOTE: When enabled, the GP counter requires 3 SCLK0_0 cycles of initialization before recognizing valid toggles on its input pins.

The three input pins can be filtered (debounced) before the GP counter evaluates them.

The GP counter also features a flexible boundary comparison. In all of the operating modes, the counter can be compared to an upper and lower limit. It takes various actions when reaching these limits.

The module can optionally generate an interrupt request to the system through its IRQ line. On many processors, a GP timer module can use an output to generate time stamps on certain events.

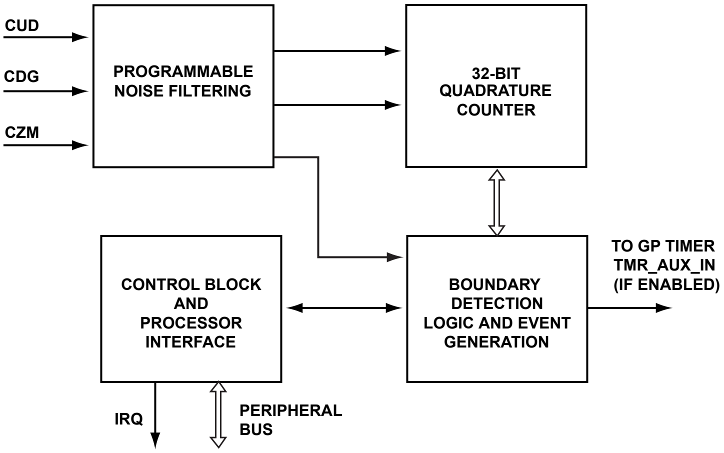

CUD
CDG
CZM
PROGRAMMABLE NOISE FILTERING
32-BIT QUADRATURE COUNTER
CONTROL BLOCK AND PROCESSOR INTERFACE
BOUNDARY DETECTION LOGIC AND EVENT GENERATION
TO GP TIMER TMR_AUX_IN (IF ENABLED)
IRQ
PERIPHERAL BUS

Figure 23-1: GP Counter Block Diagram

ADSP-SC58x CNT Register List

The GP Counter (CNT) provides support for manually controlled rotary controllers, such as the volume wheel on a radio device. This unit also supports industrial encoders.

The CNT converts pulses from incremental position encoders into data that is representative of the actual position. To complete this task, the CNT integrates (counts) pulses on one or two inputs. Because integration provides relative position, a zero position input (zero marker) is usually provided that establishes a reference point, verifying that the acquired position does not drift over time. The incremental position information may also be used to determine speed, if the relevant time intervals are measured. The CNT provides flexible ways to establish position information. When used in conjunction with the General-purpose Timer (TIMER), the CNT allows acquisition of coherent position/time stamp information, enabling speed calculation.

A set of registers govern CNT operations. For more information on CNT functionality, see the CNT register descriptions.

Table 23-1: ADSP-SC58x CNT Register List

Name	Description
`CNT_CFG`	Configuration Register
`CNT_CMD`	Command Register
`CNT_CNTR`	Counter Register
`CNT_DEBNCE`	Debounce Register
`CNT_IMSK`	Interrupt Mask Register
`CNT_MAX`	Maximum Count Register
`CNT_MIN`	Minimum Count Register
`CNT_STAT`	Status Register

ADSP-SC58x CNT Interrupt List

Table 23-2: ADSP-SC58x CNT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
126	CNT0_STAT	CNT0 Status	Level	

ADSP-SC58x CNT Trigger List

Table 23-3: ADSP-SC58x CNT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
20	`CNT0_STAT`	CNT0 Status	Level

Table 23-4: ADSP-SC58x CNT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

GP Counter Operating Modes

The GP counter has the following five modes of operation.

1. Quadrature Encoder
2. Binary Encoder
3. Up/Down Counter
4. Direction Counter

5. Timed Direction

With the exception of the timed direction mode, the GP counter can operate with the GP timer block to capture additional timing information (time stamps) associated with events detected by this block.

Quadrature Encoder Mode

In this mode, the `CNT_UD` and `CNT_DG` inputs expect a quadrature-encoded signal that is interpreted as a two-bit Gray code. The order of transitions of the `CNT_UD` and `CNT_DG` inputs determines whether the counter increments or decrements. The `CNT_CNTR` register contains the number of transitions that have occurred as shown in the ***Quadrature Events and Counting Mechanism*** table. Optionally, an interrupt is generated when both inputs change within one SCLK0_0 cycle. Gray coding prohibits such transitions. Therefore, the `CNT_CNTR` register remains unchanged, and an error condition is signaled.

Table 23-5: Quadrature Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG, CUD Inputs	00	01	11	10	00	01	11	10	00

It is possible to reverse the count direction of the Gray-coded signal by enabling the polarity inverter of either the `CNT_UD` pin or the `CNT_DG` pin. Inverting both pins does not alter the behavior. The GP counter can enable this feature with the `CNT_CFG.CDGINV` and `CNT_CFG.CUDINV` bits.

As an example, the `CNT_DG` and `CNT_UD` inputs are 00 and the next transition is to 01. These inputs normally change the counter in increments as shown in the table. If the `CNT_UD` polarity is inverted, this condition generates a received input of 01 followed by 00. The results is a decrement of the counter, altering the behavior of the connected hardware.

Binary Encoder Mode

This mode is almost identical to quadrature encoder mode, with the exception that the `CNT_UD`: `CNT_DG` inputs expect a binary-encoded signal. The order of transitions of the `CNT_UD` and `CNT_DG` inputs determines whether the counter increments or decrements. The `CNT_CNTR` register contains the number of transitions that have occurred as shown in the ***Binary Events and Counting Mechanism*** table. Optionally, an interrupt is generated when the detected code steps by more than 1 (in binary arithmetic) within one SCLK0_0 cycle. Such transitions are erroneous. Therefore, the `CNT_CNTR` register remains unchanged, and an error condition is signaled.

Table 23-6: Binary Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG:CUD Inputs	00	01	10	11	00	01	10	11	00

Reversing the `CNT_UD` and `CNT_DG` pin polarity has a different effect in binary encoder mode than for the quadrature encoder mode. Inverting the polarity of the `CNT_UD` pin only, or inverting both the `CNT_UD` and `CNT_DG` pins, results in reversing the count direction.

Up/Down Counter Mode

In this mode, the counter increments or decrements at every active edge of the input pins. The GP counter uses the `CNT_CFG.CUDINV` bit to select an active edge and has the following results.

- If the GP counter module detects an active edge at the `CNT_UD` input, the counter increments.
- If the GP counter module detects an active edge at the `CNT_DG` input, the counter decrements.
- If simultaneous edges occur on the `CNT_DG` and `CNT_UD` pins, the counter remains unchanged, and both up-count and down-count events are signaled in the `CNT_STAT` register.

Direction Counter Mode

In this mode, the counter is incremented or decremented at every active edge of the `CNT_DG` input pin. The state of the `CNT_UD` input determines whether the counter increments or decrements. The GP counter uses the `CNT_CFG.CUDINV` bit to select the polarity.

If the GP counter detects an active edge at the `CNT_DG` input, the counter value changes by one in the selected direction.

Timed Direction Mode

In this mode, the counter is incremented or decremented at each SCLK0_0 cycle. The state of the `CNT_UD` input determines whether the counter increments or decrements. The GP counter uses the `CNT_CFG.CUDINV` bit to select the polarity. The `CNT_DG` pin can be used to gate the clock. The GP counter uses the `CNT_CFG.CDGINV` bit to select the polarity.

GP Counter Programming Model

The following sections provide information for programming the interface.

GP Counter General Programming Flow

The following are general guidelines for configuring and enabling the GP counter.

1. Initialize (but do not enable) the GP counter for the desired mode and settings through the `CNT_CFG` register.
2. Usually, events of interest are processed using interrupts rather than by polling status bits. In this case, clear all status bits and activate the interrupt generation requests with the `CNT_IMSK` register.
3. Configure interrupts at the system level to insure desired interrupt signaling to the system.
4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupt requests and the GP Counter itself using the `CNT_IMSK` and `CNT_CFG` registers, respectively.

GP Counter Mode Configuration

The GP counter can use the `CNT_ZM` input pin to sense the zero marker output of a rotary device or to detect the pressing of a push button. There are four programming schemes, which are functional in all counter modes:

- Push-button mode
- Zero-marker-zeros-counter mode
- Zero-marker-error mode
- Zero-once mode

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set `CNT_IMSK.CZM` to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the `CNT_ZM` input sets the `CNT_IMSK.CZM` bit.

Configuring Zero-Marker-Zeros-Counter Mode

The following provides information on configuring zero-marker-zeros-counter mode for the GP counter.

1. Set `CNT_IMSK.CZMZ` to enable `CNT_CNTR`. The zero marker interrupt zeroes the counter.
2. Set `CNT_CFG.ZMZC` to enable ZMZC mode.
3. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
4. Proceed with any other desired configuration steps and enable the peripheral.

This configuration causes an active level at the `CNT_ZM` pin to clear the `CNT_CNTR` register and keep it cleared until the `CNT_ZM` pin is deactivated. In addition, the `CNT_STAT.CZMZ` bit is set.

Configuring Zero-Marker-Error Mode

The GP counter uses this mode to detect discrepancies between counter-value and the zero marker output of certain rotary encoder devices.

1. Set the `CNT_STAT.CZME` bit to enable this mode.
2. Select the active edge of the `CNT_ZM` pin through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

When the GP counter detects an active edge at the `CNT_ZM` input pin, it compares the four LSBs of the `CNT_CNTR` register to zero. If they are not zero, the GP counter uses `CNT_STAT.CZME` bit to signal a mismatch.

Configuring Zero-Once Mode

The GP counter uses this mode to perform an initial reset of the counter-value when it detects an active zero marker. After that, the zero marker is ignored (the counter is no longer reset).

1. Set the `CNT_CMD.W1ZMONCE` bit to enable this mode.
2. Select the active edge of the `CNT_ZM` pin through the `CNT_CFG.CZMINV` bit.
3. Ensure that at least one of the following bits is enabled: `CNT_IMSK.CZM`, `CNT_IMSK.CZME`, `CNT_IMSK.CZMZ`.
4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_CNTR` register and the `CNT_CMD.W1ZMONCE` bit are cleared on the next active edge of the `CNT_ZM` pin. Now the `CNT_CMD.W1ZMONCE` bit can be read to check whether the event has already occurred.

Configuring Boundary Auto-Extend Mode

In this mode, hardware modifies the boundary registers (`CNT_MIN` and `CNT_MAX`) whenever the `CNT_CNTR` value reaches either of them. The GP counter uses this mode to monitor the widest angle a thumb wheel even if the software did not generate interrupts.

1. Initialize `CNT_CNTR` with the desired value.
2. Set both `CNT_MIN` and `CNT_MAX` to this same value.
3. Configure the `CNT_CFG.BNDMODE` field for auto extend mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_MAX` register is loaded with the current `CNT_CNTR` value when the latter increments beyond the `CNT_MAX` value. Similarly, the `CNT_MIN` register is loaded with the `CNT_CNTR` value when the latter decrements below the `CNT_MIN` value. The `CNT_STAT.MAXC` and `CNT_STAT.MINC` status bits are set when the `CNT_CNTR` value matches the respective boundary register value.

Configuring Boundary Capture Mode

In this mode, the `CNT_CNTR` value is latched into the `CNT_MIN` register at one detected edge of the `CNT_ZM` input pin, and latched into the `CNT_MAX` boundary register at the opposite edge.

1. To capture the `CNT_ZM` pin rising edge into `CNT_MIN` and the falling edge into `CNT_MAX`, program `CNT_CFG.CZMINV` for active high polarity. Conversely, to capture the `CNT_ZM` pin falling edge into `CNT_MIN` and the rising edge into `CNT_MAX`, program `CNT_CFG.CZMINV` for active low polarity.
2. Program the `CNT_IMSK.MAXC` and `CNT_IMSK.MINC` interrupt mask bits according to interrupt generation requirements.
3. Configure the `CNT_CFG.BNDMODE` field for boundary capture mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The `CNT_STAT.MAXC` and `CNT_STAT.MINC` status bits report the capture event, depending on how interrupt masks are configured.

Configuring Boundary Compare and Boundary Zero Modes

In these modes, the two boundary registers (`CNT_MAX` and `CNT_MIN`) are compared to the value in the `CNT_CNTR` register.

1. Program `CNT_MAX` and `CNT_MIN` registers with appropriate upper and lower range values.
2. Program the `CNT_IMSK.MAXC` and `CNT_IMSK.MINC` interrupt mask bits according to interrupt generation requirements.
3. Configure the `CNT_CFG.BNDMODE` field for boundary compare mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

If after incrementing, `CNT_CNTR` = `CNT_MAX`, then the `CNT_STAT.MAXC` bit is set. Similarly if after decrementing, `CNT_CNTR` = `CNT_MIN`, then the `CNT_STAT.MINC` bit is set.

Additionally, for boundary zero mode, the counter-value in `CNT_CNTR` is set to zero. The `CNT_STAT.MAXC` and `CNT_STAT.MINC` bits are not set when software updates the `CNT_MAX` or `CNT_MIN` registers.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set `CNT_IMSK.CZM` to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the `CNT_CFG.CZMINV` bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the `CNT_ZM` input sets the `CNT_IMSK.CZM` bit.

GP Counter Programming Concepts

Using the features, operating modes, and event control for the GP counter to their greatest potential requires an understanding of some GP counter-related concepts. Some key aspects to consider are input noise filtering and capturing timing information.

CNT Input Noise Filtering

In all modes, the three input pins can be filtered to present clean signals to the GP counter logic. The GP counter uses the `CNT_CFG.DEBEN` bit to enable or disable this filtering. The ***Programmable Noise Filtering*** figure shows the filtering operation for the `CNT_UD` pin.

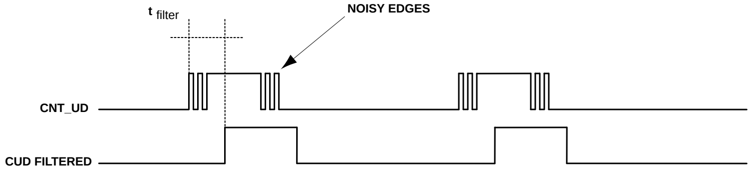


Figure 23-2: Programmable Noise Filtering

The CNT module implements the filtering mechanism using counters for each GP counter pin, where each counter is initialized from the `CNT_DEBNCE.DPRESCALE` field. When a transition is detected on a pin, the corresponding counter starts counting up to the programmed number of SCLK0_0 cycles. The state of the pin is latched after time t_{filter} and passed on to the GP counter logic.

The following formula determines the time t_{filter}, given SCLK0_0 and the `CNT_DEBNCE.DPRESCALE` value, where lower values of `CNT_DEBNCE.DPRESCALE` result in shorter debounce delays:

$t_{filter} = 128 \times (2^{DPRESCALE} \times SCLK0\_0)$

Capturing Counter Interval and CNT_CNTR Read Timing

When the count speed is low, it is often useful to capture the time elapsed since the last count event. Program the `TIMER_TMR[n]_CFG` register of the associated GP timer in a width capture mode with the following bit settings.

- `TIMER_TMR[n]_CFG.PULSEHI` = 0
- `TIMER_TMR[n]_CFG.TMODE` = b#1011
- `TIMER_TMR[n]_CFG.TINSEL` = 1

The *Capture Intervals* figure shows and the following list describe the operation of the GP counter and the GP timer in this mode.

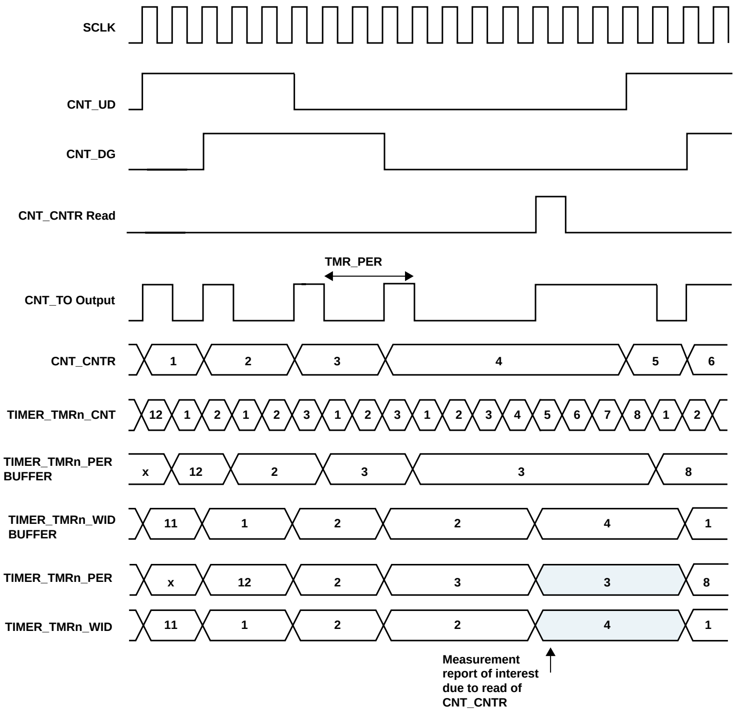

SCLK
CNT_UD
CNT_DG
CNT_CNTR Read
TMR_PER
CNT_TO Output
CNT_CNTR
1 2 3 4 5 6
TIMER_TMRn_CNT
12 1 2 1 2 3 1 2 3 1 2 3 4 5 6 7 8 1 2
TIMER_TMRn_PER BUFFER
x 12 2 3 3 8
TIMER_TMRn_WID BUFFER
11 1 2 2 4 1
TIMER_TMRn_PER
x 12 2 3 3 8
TIMER_TMRn_WID
11 1 2 2 4 1
Measurement report of interest due to read of CNT_CNTR

Figure 23-3: Capture Intervals

NOTE: SCLK in the ***Capture Intervals*** figure is SCLK0_0.

1. The CNT_TO signal generates a pulse every time a count event occurs. In addition, when the processor reads the CNT_CNTR register, the CNT_TO signal presents a pulse which is extended (high) until the next count event.
2. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from falling edge to falling edge, because TIMER_TMR[n]_CFG.PULSEHI = 0) of the CNT_TO signal.
3. The TIMER_TMR[n]_WID register is updated with the pulse width (the portion where CNT_TO is low, again because TIMER_TMR[n]_CFG.PULSEHI = 0).
4. Both registers are updated at every rising edge of the CNT_TO signal (because TIMER_TMR[n]_CFG.TMODE= b#011).

The TIMER_TMR[n]_PER register contains the period between the last two count events. The TIMER_TMR[n]_WID register contains the time since the last count event and the read of the CNT_CNTR register, both measured in SCLK0_0 cycles.

Read the CNT_CNTR register to latch the two time measurements, providing a coherent triplet of information to calculate speed and position.

NOTE: Speed restrictions apply to the use of the CNT_TO signal. Therefore, programs must not operate at count event rates that are high. For instance, if CNT_CNTR is incremented or decremented every SCLK0_0 cycle (timed direction mode), the CNT_TO signal is not valid.

Capturing Time Interval Between Successive Counter Events

When the required timing information is the interval between successive count events, program the associated timer in a width capture mode. Set the TIMER_TMR[n]_CFG bit of TIMER_TMR[n]_CFG.PULSEHI = 1, TIMER_TMR[n]_CFG.TMODE = b#1010 and TIMER_TMR[n]_CFG.TINSEL = 1. Typically, this information is sufficient if the speed of GP counter events does not to reach low values.

The ***Period Register Timing*** figure shows the operation of the GP counter and the GP timer in this mode.

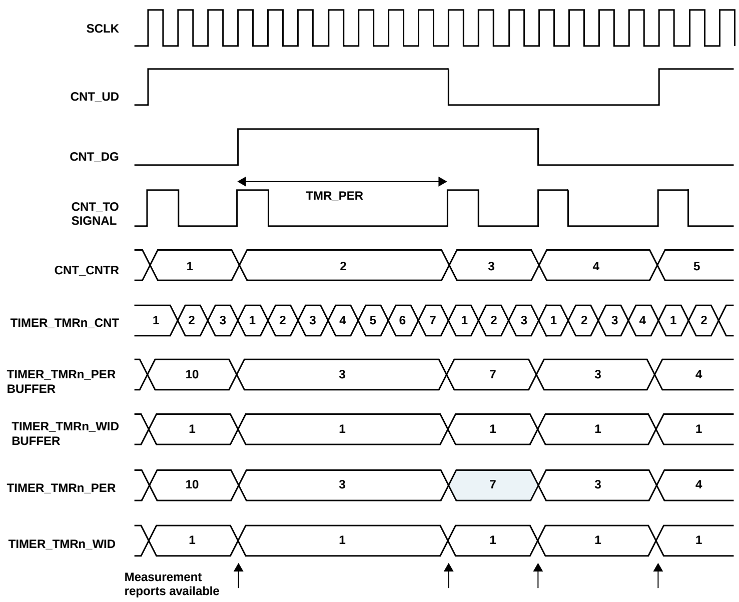

SCLK
CNT_UD
CNT_DG
TMR_PER
CNT_TO SIGNAL
CNT_CNTR
1
2
3
4
5
TIMER_TMRn_CNT
1 2 3 1 2 3 4 5 6 7 1 2 3 1 2 3 4 1 2
TIMER_TMRn_PER BUFFER
10
3
7
3
4
TIMER_TMRn_WID BUFFER
1
1
1
1
1
TIMER_TMRn_PER
10
3
7
3
4
TIMER_TMRn_WID
1
1
1
1
1
Measurement reports available

Figure 23-4: Period Register Timing

The CNT_TO signal generates a pulse every time a count event occurs. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from rising edge to rising edge) of the CNT_TO signal. The TIMER_TMR[n]_PER register is updated at every rising edge of the CNT_TO signal and contains the number of SCLK0_0 cycles that have elapsed since the previous rising edge.

Incidentally, the TIMER_TMR[n]_WID register is also updated at the same time, but is generally of no interest in this mode of operation. If no reads of the CNT_CNTR register occur between counter events, the TIMER_TMR[n]_WID register only contains the width of the CNT_TO pulse. If a read of CNT_CNTR has occurred between events, the TIMER_TMR[n]_WID register contains the time between the read of CNT_CNTR and the next event.

This mode can also be used with TIMER_TMR[n]_CFG.PULSEHI = 0. In this case, the period of CNT_TO is measured between falling edges. It results in the same values as in the previous case, only the latching occurs one SCLK0_0 cycle later.

GP Counter Event Control

Eleven events can be signaled to the processor using status information and optional interrupt requests. The GP counter uses the respective bits in the CNT_IMSK register to enable the interrupt requests. It uses dedicated bits in the CNT_STAT register to report events. When an interrupt request from the GP counter is serviced, the application software is responsible for correct interpretation of the events. It is recommended to logically AND the content of the CNT_IMSK and CNT_STAT registers to identify pending interrupt requests.

Perform a write-one-to-clear (W1C) operation to the CNT_STAT register to clear the interrupt requests. Hardware does not clear the status bits automatically, unless the counter module is disabled.

The following sections describe the events associated with the GP counter.

Illegal Gray and Binary Code Events

When illegal transitions occur in quadrature encoder or binary encoder modes, the CNT_STAT.IC bit is set. If enabled by the CNT_STAT.IC bit, the counter module generates an interrupt request. Set the CNT_STAT.IC bit only in the quadrature encoder or binary encoder modes.

Up/Down Count Events

The GP counter uses the CNT_STAT.UC bit to indicate whether the counter has been incremented. Similarly, the CNT_STAT.DC bit reports decrements. The two events are independent. For instance, if the counter increments by one and then decrements by two, both bits remain set, even though the resulting counter-value shows a decrement by one.

In up/down counter mode, hardware can detect simultaneous active edges on the CNT_UD and CNT_DG inputs. In that case, the CNT_CNTR remains unchanged, but both the CNT_STAT.UC and CNT_STAT.DC bits are set. Interrupt requests for these events can be enabled through the CNT_IMSK.UC and CNT_IMSK.DC bits. Use this feature carefully when the counter is clocked at high rates. This suggestion is especially critical when the counter operates in DIR_TMR mode, as interrupts are generated every SCLK0_0 cycle.

These events can also be used for more push buttons, when GP counter features are unnecessary. When up/down counter mode is enabled, the GP counter can use these count events to report interrupts from push buttons that connect to the CNT_UD and CNT_DG inputs.

Zero-Count Events

The CNT_STAT.CZERO status bit indicates that the CNT_CNTR has reached a value equal to 0x0000 0000 after an increment or decrement. This bit is not set when the counter value is set to zero by a write to CNT_CNTR or by setting the CNT_CMD.W1LCNTZERO bit. If enabled by the CNT_IMSK.CZERO bit, the GP counter module generates an interrupt request.

Overflow Events

There are two status bits that indicate whether the signed counter-register has overflowed from a positive to a negative value or conversely. The CNT_STAT.COV31 bit reports that the 32-bit CNT_CNTR register has either incremented from 0x7FFF FFFF to 0x8000 0000, or decremented from 0x8000 0000 to 0x7FFF FFFF.

If enabled by the CNT_IMSK.COV31 bit, an interrupt request is generated. Similarly, in applications where only the lower 16 bits of the counter are of interest, the CNT_STAT.COV15 status bit reports counter transitions from

0xXXXX 7FFF to 0xXXXX 8000, or from 0xXXXX 8000 to 0xXXXX 7FFF. If enabled by the `CNT_IMSK.COV15` bit, an interrupt request is generated.

Boundary Match Events

The `CNT_STAT.MINC` and `CNT_STAT.MAXC` status bits report boundary events as described in Configuring Boundary Capture Mode. These bits are not set if the software updates the `CNT_CNTR`, `CNT_MAX`, or `CNT_MIN` registers or writes to the `CNT_CMD` register. The `CNT_IMSK.MINC` and `CNT_IMSK.MAXC` bits enable interrupt request generation on boundary events.

Zero Marker Events

The `CNT_STAT.CZM`, `CNT_STAT.CZME`, and `CNT_STAT.CZMZ` bits are associated with zero marker events, as described in Configuring GP Counter Push-Button Operation. Each of these events can optionally generate an interrupt request, when enabled by the corresponding `CNT_IMSK.CZM`, `CNT_IMSK.CZME` and `CNT_IMSK.CZMZ` bits.

ADSP-SC58x CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 23-7: ADSP-SC58x CNT Register List

Name	Description
`CNT_CFG`	Configuration Register
`CNT_CMD`	Command Register
`CNT_CNTR`	Counter Register
`CNT_DEBNCE`	Debounce Register
`CNT_IMSK`	Interrupt Mask Register
`CNT_MAX`	Maximum Count Register
`CNT_MIN`	Minimum Count Register
`CNT_STAT`	Status Register

Configuration Register

The CNT_CFG register configures counter modes, configures input pins, and enables the CNT.

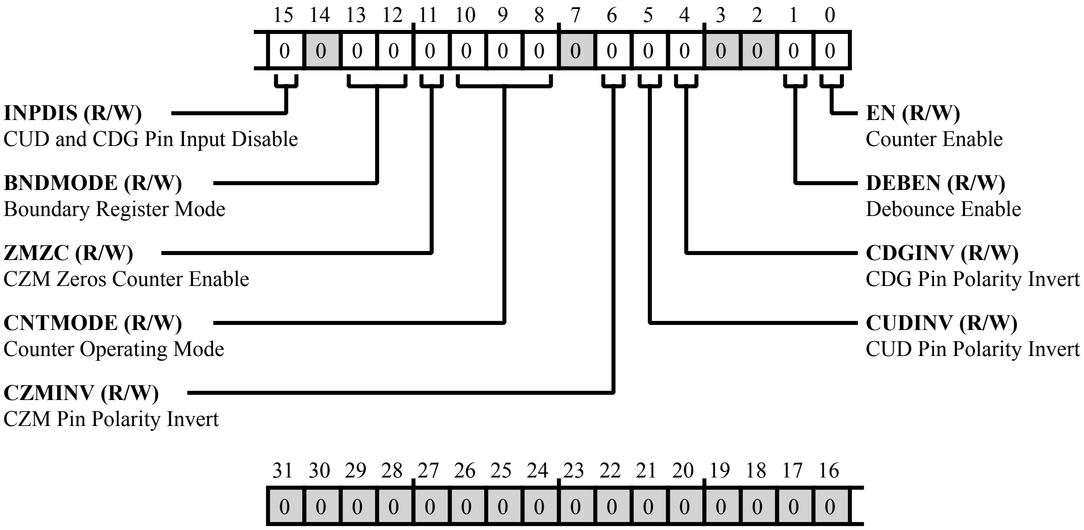

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INPDIS (R/W)
CUD and CDG Pin Input Disable
BNDMODE (R/W)
Boundary Register Mode
ZMZC (R/W)
CZM Zeros Counter Enable
CNTMODE (R/W)
Counter Operating Mode
CZMINV (R/W)
CZM Pin Polarity Invert
EN (R/W)
Counter Enable
DEBEN (R/W)
Debounce Enable
CDGINV (R/W)
CDG Pin Polarity Invert
CUDINV (R/W)
CUD Pin Polarity Invert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-5: CNT_CFG Register Diagram

Table 23-8: CNT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	INPDIS	CUD and CDG Pin Input Disable. The CNT_CFG.INPDIS disables or enables the CNT_UD input pin and the CNT_DG pin.	
		0	Enable
		1	Pin Input Disable
13:12 (R/W)	BNDMODE	Boundary Register Mode. The CNT_CFG.BNDMODE bit field selects the mode for the CNT_MIN and CNT_MAX boundary registers.	
		0	BND_COMP. Boundary Compare Mode
		1	BND_ZERO. Boundary Zero Mode
		2	BND_CAPT. Boundary Capture Mode
		3	BND_AEXT. Boundary Auto-extend Mode

Table 23-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ZMZC	CZM Zeros Counter Enable. The `CNT_CFG.ZMZC` bit enables or disables level sensitive - active `CNT_ZM` pin operation to zero the `CNT_CNTR` register.	
		0	Disable
		1	Enable
10:8 (R/W)	CNTMODE	Counter Operating Mode. The `CNT_CFG.CNTMODE` bit field selects the operating mode for the `CNT_UD` input pin and the `CNT_DG` pin.	
		0	QUAD_ENC. Quadrature Encoder Mode
		1	BIN_ENC. Binary Encoder Mode
		2	UD_CNT. Rotary Counter Mode
		4	DIR_CNT. Direction Counter Mode
		5	DIR_TMR. Direction Timer Mode
6 (R/W)	CZMINV	CZM Pin Polarity Invert. The `CNT_CFG.CZMINV` bit selects the polarity for the `CNT_ZM` pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
5 (R/W)	CUDINV	CUD Pin Polarity Invert. The `CNT_CFG.CUDINV` bit selects the polarity for the `CNT_UD` pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
4 (R/W)	CDGINV	CDG Pin Polarity Invert. The `CNT_CFG.CDGINV` bit selects the polarity for the `CNT_DG` pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge

Table 23-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DEBEN	Debounce Enable. The `CNT_CFG.DEBEN` bit enables or disables CNT input debounce filtering operation selected with the `CNT_DEBNCE` register.	
		0	Disable
		1	Enable
0 (R/W)	EN	Counter Enable. The `CNT_CFG.EN` bit enables or disables CNT operation.	
		0	Counter Disable
		1	Counter Enable

Command Register

The `CNT_CMD` register configures the CNT, enabling operations such as zeroing a counter register and copying or swapping boundary registers. These actions are taken by setting the appropriate bit.

Read operations from this register do not return meaningful values, with the exception of the `CNT_CMD.W1ZMONCE` bit, where a set bit indicates that the bit has been set by software before, but a zero marker event has not yet been detected on the `CNT_ZM` pin yet. For more information, see the CNT functional description.

The `CNT_CNTR`, `CNT_MIN`, and `CNT_MAX` registers can be initialized to zero by setting the `CNT_CMD.W1LCNTZERO`, `CNT_CMD.W1LMINZERO`, and `CNT_CMD.W1LMAXZERO` bits. In addition to clearing registers, the `CNT_CMD` register permits modifying the `CNT_MIN` and `CNT_MAX` boundary registers in a number of ways. The current counter value in the `CNT_CNTR` register can be captured and loaded into either of the two boundary registers to create new boundary limits. This operation is performed by setting the `CNT_CMD.W1LMAXCNT` and `CNT_CMD.W1LMINCNT` bits. Alternatively, the counter can be loaded from `CNT_MAX` or `CNT_MIN` using the `CNT_CMD.W1LCNTMAX` and `CNT_CMD.W1LCNTMIN` bits. It is also possible to transfer the current `CNT_MAX` value into `CNT_MIN` (or conversely) through the `CNT_CMD.W1LMINMAX` and `CNT_CMD.W1LMAXMIN` bits.

Another counter operation is the ability to only have the zero marker clear the `CNT_CNTR` register once. For more information, see the CNT functional description.

It is possible for multiple actions to be performed simultaneously by setting multiple bits in the `CNT_CMD` register. However, there are restrictions. The bits associated with each command have been grouped together such that all bits that involve a write to the `CNT_CNTR`, `CNT_MAX`, or `CNT_MIN` registers are located within bits 4-bit groups of the `CNT_CMD` register.

Note that a maximum of three commands can be issued at any one time, excluding the `CNT_CMD.W1ZMONCE` command. Also, note that `CNT_CMD.W1LCNTMIN`, `CNT_CMD.W1LCNTMAX`, and `CNT_CMD.W1LCNTZERO` bits have to be used exclusively. Never set more than one of them at the same time.

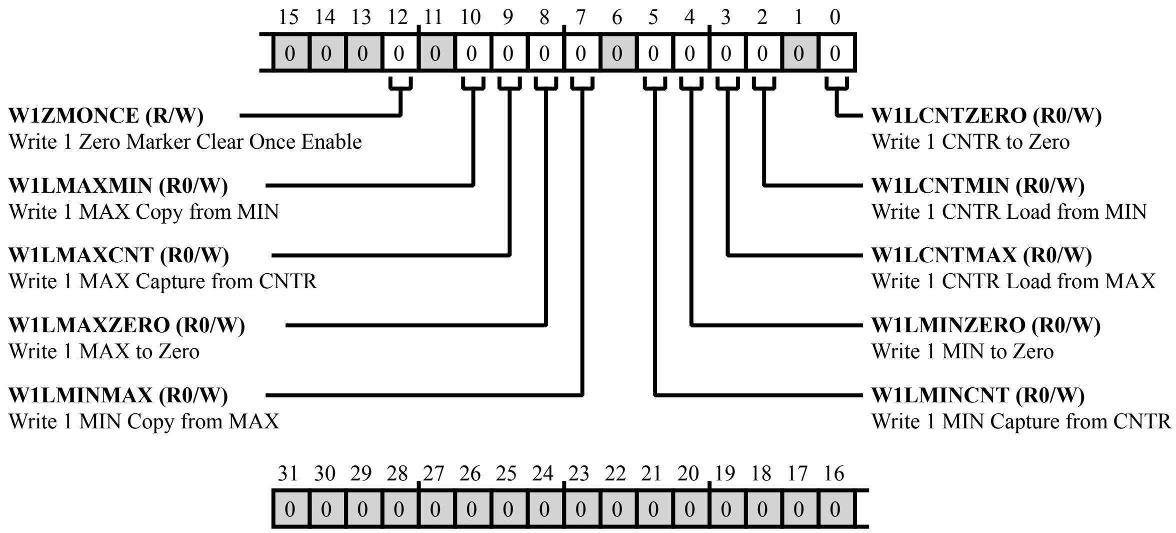

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
W1ZMONCE (R/W)
Write 1 Zero Marker Clear Once Enable
W1LMAXMIN (R0/W)
Write 1 MAX Copy from MIN
W1LMAXCNT (R0/W)
Write 1 MAX Capture from CNTR
W1LMAXZERO (R0/W)
Write 1 MAX to Zero
W1LMINMAX (R0/W)
Write 1 MIN Copy from MAX
W1LCNTZERO (R0/W)
Write 1 CNTR to Zero
W1LCNTMIN (R0/W)
Write 1 CNTR Load from MIN
W1LCNTMAX (R0/W)
Write 1 CNTR Load from MAX
W1LMINZERO (R0/W)
Write 1 MIN to Zero
W1LMINCNT (R0/W)
Write 1 MIN Capture from CNTR
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-6: CNT_CMD Register Diagram

Table 23-9: CNT_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	W1ZMONCE	Write 1 Zero Marker Clear Once Enable. The `CNT_CMD.W1ZMONCE` enables a single zero marker clear of the `CNT_CNTR` register. Reading a 1 in this bit indicates that the bit has been set by software before, but no zero marker event has been detected on the `CNT_ZM` pin yet.
10 (R0/W)	W1LMAXMIN	Write 1 MAX Copy from MIN. The `CNT_CMD.W1LMAXMIN` bit transfers the current `CNT_MIN` register value into `CNT_MAX` register.
9 (R0/W)	W1LMAXCNT	Write 1 MAX Capture from CNTR. The `CNT_CMD.W1LMAXCNT` bit loads the current value in the `CNT_CNTR` register into the `CNT_MAX` register to create a new boundary limit.
8 (R0/W)	W1LMAXZERO	Write 1 MAX to Zero. Writing a 1 to the `CNT_CMD.W1LMAXZERO` bit clears the `CNT_MAX` register.
7 (R0/W)	W1LMINMAX	Write 1 MIN Copy from MAX. The `CNT_CMD.W1LMINMAX` bit transfers the current `CNT_MAX` register value into `CNT_MIN` register.
5 (R0/W)	W1LMINCNT	Write 1 MIN Capture from CNTR. The `CNT_CMD.W1LMINCNT` bit loads the current value in the `CNT_CNTR` register into the `CNT_MIN` register to create a new boundary limit.
4 (R0/W)	W1LMINZERO	Write 1 MIN to Zero. Writing a 1 to the `CNT_CMD.W1LMINZERO` bit clears the `CNT_MIN` register.

Table 23-9: CNT_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R0/W)	W1LCNTMAX	Write 1 CNTR Load from MAX. The `CNT_CMD.W1LCNTMAX` bit loads the current value in the `CNT_MAX` register into the `CNT_CNTR` register to create a new boundary limit.
2 (R0/W)	W1LCNTMIN	Write 1 CNTR Load from MIN. The `CNT_CMD.W1LCNTMIN` bit loads the current value in the `CNT_MIN` register into the `CNT_CNTR` register to create a new boundary limit.
0 (R0/W)	W1LCNTZERO	Write 1 CNTR to Zero. Writing a 1 to the `CNT_CMD.W1LCNTZERO` bit clears the `CNT_CNTR` register.

Counter Register

The CNT_CNTR register holds the 32-bit, two's-complement count value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows use of the CNT as a 16-bit counter if sufficient for the application.

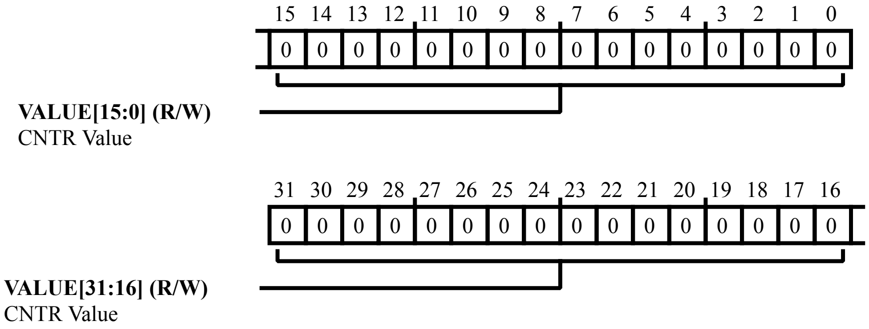

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
CNTR Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
CNTR Value

Figure 23-7: CNT_CNTR Register Diagram

Table 23-10: CNT_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CNTR Value. The `CNT_CNTR.VALUE` bit field holds the 32-bit, two's-complement count value.

Debounce Register

The `CNT_DEBNCE` register selects the noise filtering characteristic of the three input pins according to the formula:

$t_{filter} = 128 \times (2^{DPRESCALE} / SCLK0\_0)$

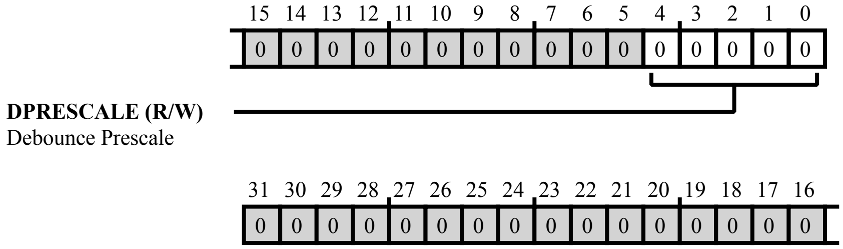

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DPRESCALE (R/W)
Debounce Prescale
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-8: CNT_DEBNCE Register Diagram

Table 23-11: CNT_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W)	DPRESCALE	Debounce Prescale. The `CNT_DEBNCE.DPRESCALE` selects the desired number of input filtering cycles (and resulting input debounce time) in multiples of SCLK0_0.	
		0	1x cycles = 128 SCLK0_0 cycles
		1	2x cycles
		2	4x cycles
		3	8x cycles
		4	16x cycles
		5	32x cycles
		6	64x cycles
		7	128x cycles
		8	256x cycles
		9	512x cycles
		10	1024x cycles
		11	2048x cycles
		12	4096x cycles
		13	8192x cycles
		14	16384x cycles
		15	32768x cycles
		16	65536x cycles

Table 23-11: CNT_DEBNCE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		17	131072x cycles
		18	Reserved from this value. The values 10010 - 11111 are reserved.
		31	Reserved until this value

Interrupt Mask Register

The `CNT_IMSK` register supports enabling (unmasking) interrupt request generation from each of the CNT events.

All bits in `CNT_IMSK` either disable/mask an interrupt request (if bit cleared) or enable/unmask an interrupt request (if bit set).

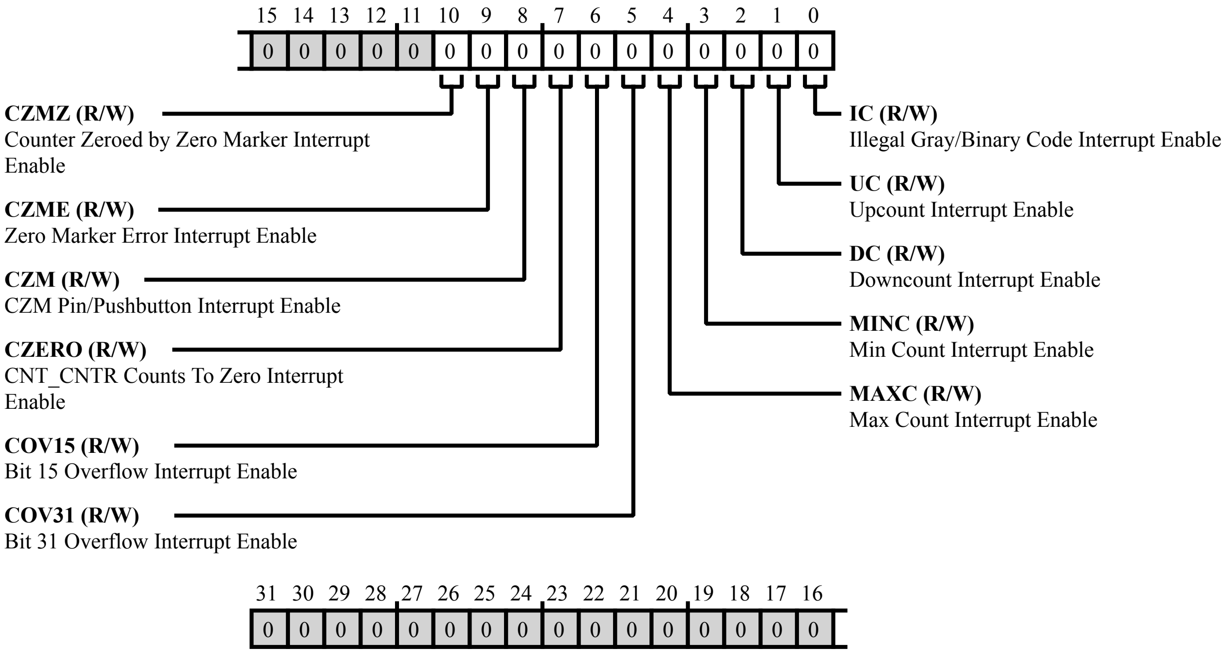

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CZMZ (R/W)
Counter Zeroed by Zero Marker Interrupt Enable
CZME (R/W)
Zero Marker Error Interrupt Enable
CZM (R/W)
CZM Pin/Pushbutton Interrupt Enable
CZERO (R/W)
CNT_CNTR Counts To Zero Interrupt Enable
COV15 (R/W)
Bit 15 Overflow Interrupt Enable
COV31 (R/W)
Bit 31 Overflow Interrupt Enable
IC (R/W)
Illegal Gray/Binary Code Interrupt Enable
UC (R/W)
Upcount Interrupt Enable
DC (R/W)
Downcount Interrupt Enable
MINC (R/W)
Min Count Interrupt Enable
MAXC (R/W)
Max Count Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-9: CNT_IMSK Register Diagram

Table 23-12: CNT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	CZMZ	Counter Zeroed by Zero Marker Interrupt Enable. The `CNT_IMSK.CZMZ` bit enables (unmasks) the counter zeroed by zero marker interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
9 (R/W)	CZME	Zero Marker Error Interrupt Enable. The `CNT_IMSK.CZME` bit enables (unmasks) the zero marker error interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 23-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	CZM	CZM Pin/Pushbutton Interrupt Enable. The `CNT_IMSK.CZM` bit enables (unmasks) the CZM pin/pushbutton interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
7 (R/W)	CZERO	CNT_CNTR Counts To Zero Interrupt Enable. The `CNT_IMSK.CZERO` bit enables (unmasks) the counts to zero interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
6 (R/W)	COV15	Bit 15 Overflow Interrupt Enable. The `CNT_IMSK.COV15` bit enables (unmasks) the bit 15 overflow interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable. The `CNT_IMSK.COV31` bit enables (unmasks) the bit 31 overflow interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
4 (R/W)	MAXC	Max Count Interrupt Enable. The `CNT_IMSK.MAXC` bit enables (unmasks) the max count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
3 (R/W)	MINC	Min Count Interrupt Enable. The `CNT_IMSK.MINC` bit enables (unmasks) the min count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt Enable. The `CNT_IMSK.DC` bit enables (unmasks) the down count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 23-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	UC	Upcount Interrupt Enable. The `CNT_IMSK.UC` bit enables (unmasks) the up count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
0 (R/W)	IC	Illegal Gray/Binary Code Interrupt Enable. The `CNT_IMSK.IC` bit enables (unmasks) the illegal Gray/Binary Code interrupt request and should only be used in these modes.	
		0	Mask Interrupt
		1	Unmask Interrupt

Maximum Count Register

The `CNT_MAX` register holds the 32-bit, two's-complement, higher boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

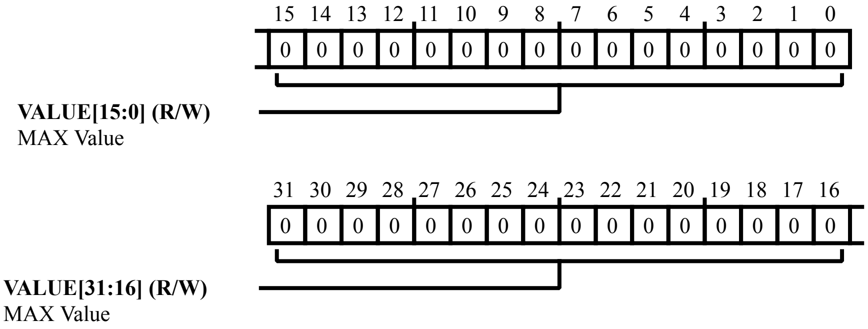

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MAX Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MAX Value

Figure 23-10: CNT_MAX Register Diagram

Table 23-13: CNT_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MAX Value. The `CNT_MAX.VALUE` bit field holds the 32-bit, two's-complement, higher boundary value.

Minimum Count Register

The `CNT_MIN` register holds the 32-bit, two's-complement, lower boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

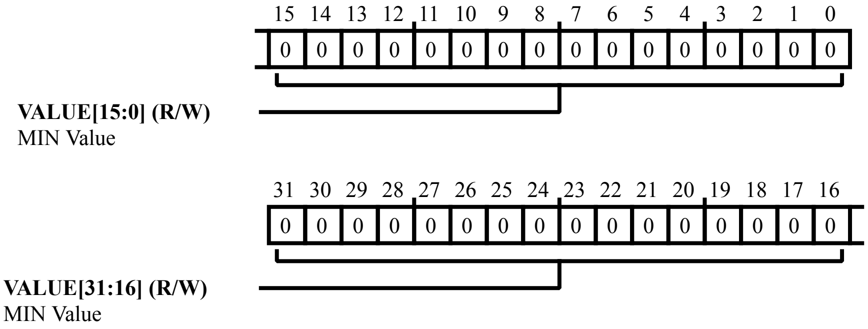

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MIN Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MIN Value

Figure 23-11: CNT_MIN Register Diagram

Table 23-14: CNT_MIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MIN Value. The `CNT_MIN.VALUE` bit field holds the 32-bit, two's-complement, lower boundary value.

Status Register

The CNT_STAT register provides status information for each of the CNT events as configured in the CNT_IMSK register. When a CNT event is detected, the corresponding bit in this register is set. It remains set until either software writes a 1 to the bit (write-1-to-clear) or the CNT is disabled.

All bits in the CNT_STAT register indicate either no interrupt request pending (if bit cleared) or an interrupt request pending (if bit set).

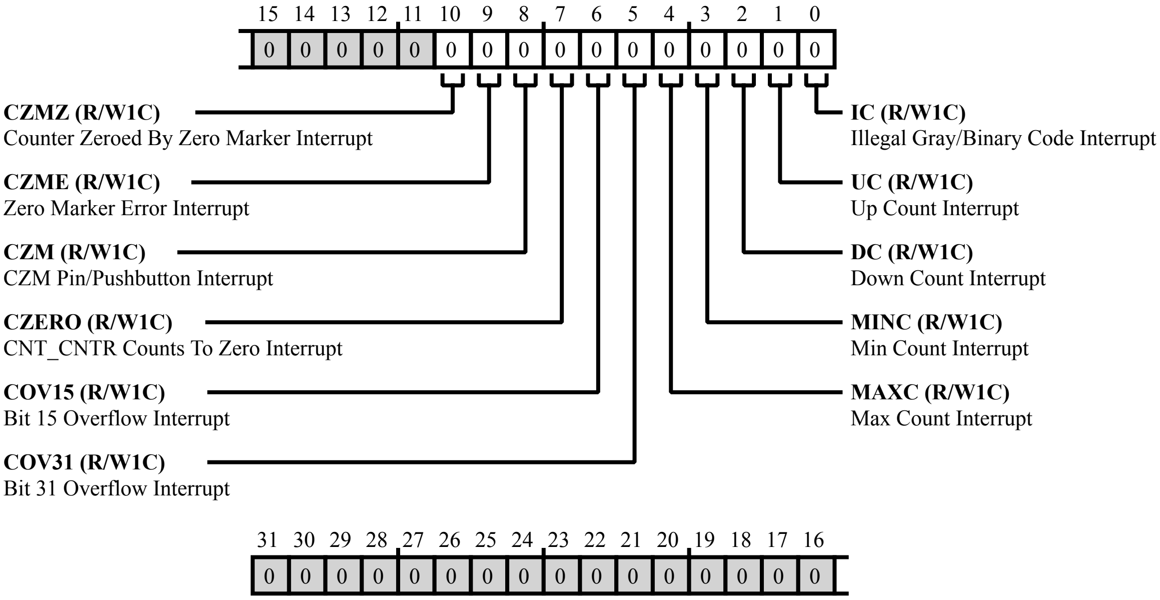

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CZMZ (R/W1C)
Counter Zeroed By Zero Marker Interrupt
CZME (R/W1C)
Zero Marker Error Interrupt
CZM (R/W1C)
CZM Pin/Pushbutton Interrupt
CZERO (R/W1C)
CNT_CNTR Counts To Zero Interrupt
COV15 (R/W1C)
Bit 15 Overflow Interrupt
COV31 (R/W1C)
Bit 31 Overflow Interrupt
IC (R/W1C)
Illegal Gray/Binary Code Interrupt
UC (R/W1C)
Up Count Interrupt
DC (R/W1C)
Down Count Interrupt
MINC (R/W1C)
Min Count Interrupt
MAXC (R/W1C)
Max Count Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-12: CNT_STAT Register Diagram

Table 23-15: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	CZMZ	Counter Zeroed By Zero Marker Interrupt. The CNT_STAT.CZMZ bit indicates a zero marker error. If the CNT_CFG.ZMZC bit =1, this interrupt request is generated when the CZMII latch reports a significant edge on the CZM input. Once cleared by software the CNT_STAT.CZM bit is not set again when the CZM input remains active without pulsing.	
		0	No error
		1	Error occurred

Table 23-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	CZME	Zero Marker Error Interrupt. The `CNT_STAT.CZME` bit behaves similarly to the `CNT_STAT.CZM` bit, with the exception that `CNT_STAT.CZME` is not set on the CZM edge when the lower four bits of the `CNT_CNTR` are not zero. In many applications this indicates an error condition, as the zero marker might be out of sync with the counter.	
		0	No error
		1	Error occurred
8 (R/W1C)	CZM	CZM Pin/Pushbutton Interrupt. The `CNT_STAT.CZM` bit indicates a CZM pin/pushbutton error. This interrupt request is generated when a significant edge is seen on the CZM pin, regardless what mode the counter is operating in. This is often used to sense push buttons (especially with the debouncing circuit enabled).	
		0	No error
		1	Error occurred
7 (R/W1C)	CZERO	CNT_CNTR Counts To Zero Interrupt. The `CNT_STAT.CZERO` bit indicates a counts to zero error. This error is generated when the `CNT_CNTR` register has incremented or decremented toward 0x0000.0000. The latch is not set when software writes to the `CNT_CNTR` register directly or when the counter is zeroed by writes to the `CNT_CMD` register.	
		0	No error
		1	Error occurred
6 (R/W1C)	COV15	Bit 15 Overflow Interrupt. The `CNT_STAT.COV15` bit indicates a bit 15 overflow error. This error is generated when the 16-bit twos-complement `CNT_CNTR` register has incremented from 0xxxxx.7FFF to 0xxxxx.8000 or decremented from 0xxxxx.8000 to 0xxxxx.7FFF.	
		0	No error
		1	Error occurred
5 (R/W1C)	COV31	Bit 31 Overflow Interrupt. The `CNT_STAT.COV31` bit indicates a bit 31 overflow error. This error is generated when the 32-bit twos-complement `CNT_CNTR` register has incremented from 0x7FFF.FFFF to 0x8000.0000 or decremented from 0x8000.0000 to 0x7FFF.FFFF.	
		0	No error
		1	Error occurred

Table 23-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	MAXC	Max Count Interrupt. The `CNT_STAT.MAXC` bit indicates a max count error. This interrupt is used in boundary compare (BND_COMP) mode. If after incrementing the `CNT_CNTR` register equals `CNT_MAX`, the `CNT_STAT.MAXC` bit is set.	
		0	No error
		1	Error occurred
3 (R/W1C)	MINC	Min Count Interrupt. The `CNT_STAT.MINC` bit indicates a minimum count error. This interrupt is used in boundary compare (BND_COMP) mode. If, after decrementing, the `CNT_CNTR` register equals `CNT_MIN`, the `CNT_STAT.MINC` bit is set.	
		0	No error
		1	Error occurred
2 (R/W1C)	DC	Down Count Interrupt. The `CNT_STAT.DC` bit indicates a down count error. This interrupt is generated when the `CNT_CNTR` register decrements.	
		0	No error
		1	Error occurred
1 (R/W1C)	UC	Up Count Interrupt. The `CNT_STAT.UC` bit indicates an up count interrupt. This interrupt is generated when the `CNT_CNTR` register increments.	
0 (R/W1C)	IC	Illegal Gray/Binary Code Interrupt. The `CNT_STAT.IC` bit indicates a illegal Gray/Binary Code interrupt and should only be used in these modes. In normal operation those codes can increment or decrement the `CNT_CNTR` register by one at a time. If the sensed inputs instruct the counter to increment or decrement by two, the `CNT_STAT.IC` bit is set. Hardware sets the `CNT_STAT.IC` bit in QUAD_ENC and BIN_ENC encoder modes only.	
		0	No error
		1	Error occurred

24 ADC Control Module (ACM)

The processor includes an ADC control module (ACM) that provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The processor initiates analog-to-digital conversions, based on either external or internal events.

Traditionally, ADC sampling uses processor interrupts and the interrupt service routine programming of the appropriate peripheral for initiating the ADC conversion process. Events initiate the interrupts. The interrupt service routine usually programs the SPORT or SPI peripherals. This traditional approach has some limiting factors:

- The ADC sampling instances are not precisely controlled due to interrupt latencies (which can vary) or due to variable instruction execution cycles
- Consumption of processor instruction cycles can be prohibitive, especially for high frequency of conversion-related events.
- If the ADC requires control signals with specific set-up, hold, or zero time for sampling time, it is difficult to provide the signals with GP flags in the application. For example, channel select pins, ADC mode select, ADC range pin.

The ADC control module (ACM) provides dedicated hardware to work around these limitations. The module samples the events and provides sampling signals and timing to the ADC in real time. The ADC permits flexible scheduling of sampling instants and provides precise sampling signals to the ADC. The ACM saves processor bandwidth and provides precise control for ADC sampling time. Furthermore, the processor can be interfaced directly to multiple ADCs without any glue logic required.

The ACM synchronizes the ADC conversion process (by providing the ADC clock, the ADC conversion start signal, and related ADC controls). However, other peripherals such as SPORT acquire the actual data from the ADC. The processor does not support ACM operation with the SPI. The ***ADC/SPORT Interface*** figure shows how an external ADC can be interfaced using the ACM and SPORT0 peripherals of the processor.

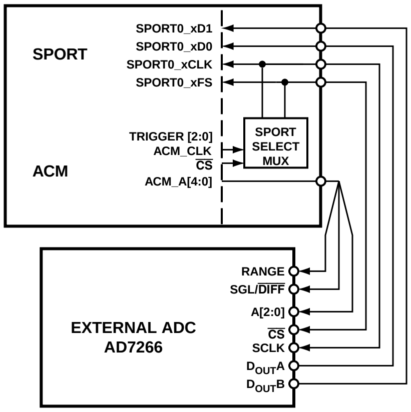


Figure 24-1: ADC/SPORT Interface

NOTE: The processor does not include an on-chip, internal ADC.

ACM Features

The ADC control module (ACM) offers the following features and capabilities:

- The ACM on the ADSP-SC58x/ADSP-2158x processors operate on the SCLK0_0 domain.
- The ACM can interface to the ADC at a maximum clock frequency of 30 MHz.
- It provides serial clock, chip select, and five general-purpose control lines capable of controlling the ADC operations. Internally routes serial clock and frame sync (chip select) signals to serial port 0.

  Only SPORT0 performs actual data acquisition from the ADC (no other SPORT instance). The `ACM_CLK` and `ACM_FS` signals are multiplexed with the SP0_CLK and SP0_FS signals. These signals are driven through the DAI0 and not through the GPIO port. The SPORT uses the `ACM_CTL.EN` bit to control the connection between SPORT0 and the ACM. By setting the `ACM_CTL.EN` bit, the ACM clock and FS are internally multiplexed to the SPORT0 SPORT0_CLK_O/SPORT0_FS_O outputs of the DAI0.
- The ACM can accept three trigger inputs (one from external on `ACM_T0` and two from the TRU) based on which it can precisely initiate the ADC sampling events. The trigger inputs can be internally generated or externally supplied. The polarity of trigger inputs is configurable.
- The ACM can handle 16 ADC sampling events per valid trigger received. Each event can be independently programmed to specify when to initiate ADC sampling for a trigger input.
- Two independent 32-bit ACM timers that can be used to divide 16 events into two groups of 8 events.
- Automatically stops the ACM timer after completion of associated events, saving power.
- Four-deep pending FIFO to queue the active events when the ACM is busy.
- The ACM can internally generate serial clock up to SCLK ÷ 2 rate. Improved granularity for internal clock generation, allowing both odd and even SCLK:ACLK ratios.

- The ACM clock can be gated (active only during enabled events) to interface it with SPI-compatible ADCs.
- When initiating an ADC sampling cycle, the width of chip select signal can be configured from 1 ACLK to 256 ACLKs. ADC can also use the width of chip select signal as start of conversion. Further the polarity of this signal can be configured as active-high or active-low signal.
- Auto ACLK adjustment at the time of $\overline{\text{CS}}$ assertion. After assertion of the $\overline{\text{CS}}$ signal, the first edge of ACLK can be configured to be either rising edge or falling edge.
- The ACM provides the five general-purpose control lines that can be programmed for required set-up and hold time based on the ADC sampling cycle. Additionally, zero time can be inserted between two successive sampling cycles.
- The ACM provides 16 event order registers (one per each event) which indicate the order in which events are handled. Optionally, the trigger input of the ACM can clear these registers automatically.
- The ACM hardware flags the appropriate event completion status bit on completion of an event. If an event is missed, the appropriate event missed status bit is flagged. Each event has separate bits. Optionally, the event completion interrupt and event missed interrupt can be triggered on these respective conditions.
- Predictable latency between the internal occurrence of an event and the assertion of a sampling event.
- The ACM can operate as trigger master to provide signal to TRU upon completion of events.

ACM Functional Description

The ADC control module uses internal ACM timers and the event time register to create events. Enable one of the timers (or both timers) for the ACM operation. Program the appropriate event control register and event time register values. After receiving a valid trigger on the selected trigger input, the timer starts counting. When the timer count matches the time specified in the event time register (`ACM_EVTIME[n]`), the comparators generate an active event signal to the timing generation unit. The signal starts the ADC access. The event must be enabled and the event must be associated with the timer. The counter continues counting, and for each match with enabled event time, the ACM gives an event signal to the timing generation unit.

ADSP-SC58x ACM Register List

The ADC control module (ACM) provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The analog-to-digital conversions are initiated by the processor, based on external or internal events. A set of registers govern ACM operations. For more information on ACM functionality, see the ACM register descriptions.

Table 24-1: ADSP-SC58x ACM Register List

Name	Description
`ACM_CTL`	Control Register
`ACM_EVCTL[n]`	Event N Control Register

Table 24-1: ADSP-SC58x ACM Register List (Continued)

Name	Description
`ACM_EVMSK`	Event Complete Interrupt Mask Register
`ACM_EVORD[n]`	Event N Order Register
`ACM_EVSTAT`	Event Complete Status Register
`ACM_EVTIME[n]`	Event N Time Register
`ACM_MEVMSK`	Missed Event Interrupt Mask Register
`ACM_MEVSTAT`	Missed Event Status Register
`ACM_STAT`	Status Register
`ACM_TC0`	Timing Configuration 0 Register
`ACM_TC1`	Timing Configuration 1 Register
`ACM_TMR0`	Timer 0 Register
`ACM_TMR1`	Timer 1 Register

ADSP-SC58x ACM Interrupt List

Table 24-2: ADSP-SC58x ACM Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
36	ACM0_EVT_MISS	ACM0 Event Miss		
37	ACM0_EVT_COMPLETE	ACM0 Event Complete		

ADSP-SC58x ACM Trigger List

Table 24-3: ADSP-SC58x ACM Trigger List Masters

Trigger ID	Name	Description	Sensitivity
132	`ACM0_EVT_COMPLETE`	ACM0 Event Complete	

Table 24-4: ADSP-SC58x ACM Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
135	`ACM0_TRIG2`	ACM0 Trigger Input 2	Pulse
136	`ACM0_TRIG3`	ACM0 Trigger Input 3	Pulse

ACM Event Handling Latency

The ACM ensures a predictable latency between the internal occurrence of an event and the assertion of a sampling event by the timing generation unit (for example, the assertion of $\overline{\text{CS}}$ and other ACM signals). The internal occurrence of an event is when the event time value matches the ACM timer count value.

Latency between occurrence of an event to $\overline{\text{CS}}$ assertion = (t_S + t_{ED}) *SCLK* cycles, where:

- t_S= ADC control setup cycles programmed in the `ACM_TC0` register
- t_{ED} = 1 SCLK0_0 cycle latency

This predictable latency is applicable only when events are generated when the timing generation unit is idle. If this unit was processing a prior sampling event, the new event is held in the pending event FIFO. The duration that the new event is held in the pending event FIFO increases the latency.

If an external trigger input is selected as a trigger input, then synchronization to this signal leads to a 3 SCLK0_0 cycle fixed delay and 1 SCLK0_0 cycle variability. This result is due to delays in latching asynchronous external triggers. When the external trigger is synchronous to SCLK0_0, the 1 SCLK0_0 cycle variability is eliminated. The latency from the external trigger to the start of the count of an ACM timer becomes fixed at 3 SCLK0_0 cycles. This latency is denoted as t_{TRIG}.

As a result, the total latency between an external trigger and the assertion of an ADC sampling event is:

Total latency = t_{TRIG} + t_{ED} + t_{PD} + t_S

The latency calculation assumes that the sampling event is not queued in the pending event FIFO. The ***Latency of External Triggers to Assertion of ADC Sampling Events*** figure shows latency details from the occurrence of external triggers to the assertion of ADC sampling events.

Observe the following timing definitions:

- t_{TRIG} = trigger to timer start delay (3–4 SCLK0_0 cycles)
- t_{PD} = event time (programmed in the `ACM_EVTIME[n]` register of the event)
- t_{ED} = internal event delay (1 SCLK0_0 cycle)
- t_S = setup time (programmed in the `ACM_TC0` register)
- t_{CSW} = $\overline{\text{CS}}$ width (programmed in the `ACM_TC1.CSW` bit field)
- t_H = hold time (programmed in the `ACM_TC1` register)

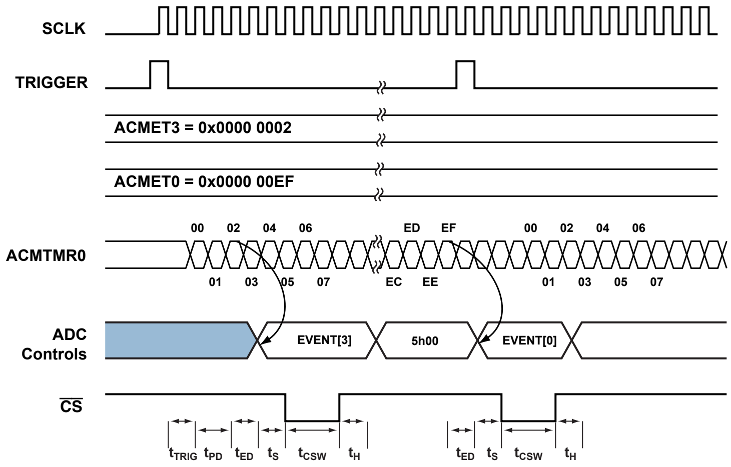


Figure 24-2: Latency of External Triggers to Assertion of ADC Sampling Events

ACM Timing Specifications

The AC timing of the ACM signals is specified in the product-specific data sheet. When trigger sources external to the processor are used for triggering the ACM, the minimum pulse width for the sources must be greater than one SCLK0_0 period. This pulse width is the minimum needed for the ACM trigger logic to detect the source as a valid trigger (for example, external signals on the GPIO, timer, or PWM sync pins).

- When the processor uses the ACM with the SPORTs, the requirements for the SPORT data signals vary. The setup and hold time requirements for the SPORT data signals for the `SPORT_ACLK` signal differ from those requirements for an internally-generated or externally-supplied SPORT clock. Consult the product-specific data sheet for information on these timing requirements.

When using gated clock mode (`ACM_CTL.CLKMOD` =1), configure the interfaced serial mode in gated clock mode (`SPORT_CTL_A.GCLKEN`, `SPORT_CTL_B.GCLKEN` =1). In this case, some conditions must be satisfied to set up the serial port in gated clock mode. These conditions are:

- The serial port needs at least 7 serial clock cycles between enabling the SPORT and first frame sync. If this requirement is not met, the SPORT may drop the first data. (For subsequent data, this requirement is not applicable.)
- Set the frame sync to the inactive (deasserted) state when the SPORT is enabled. Otherwise, one extra cycle (in addition to the cycle mentioned) is needed before the frame sync can be applied. If this requirement is not met, the SPORT may drop the first data.

ACM External Pin Timing

The ACM clock (`ACM_CLK`) is derived internally from SCLK0_0 using the `ACM_TC0.CKDIV` divider. The other output signals, such as the ADC control pins (`ACM_A0` through `ACM_A4`) and the chip select are driven on the rising edge of SCLK0_0. As a result, these signals cannot be synchronous to the `ACM_CLK` signal.

The ACM uses timing configuration registers (`ACM_TC0` and `ACM_TC1`) to configure setup, hold, and other timing parameters of the ADC control signals, the width of the chip select signal, and the frequency of `ACM_CLK`. The polarity of $\overline{\text{CS}}$ and the `ACM_CLK` signals can be configured in the ACM control register (`ACM_CTL`). The timing parameters of the ADC control pins cannot be individually specified.

The ***Timing Reference*** figure shows the inactive period of $\overline{\text{CS}}$ as t_{CSIW}. The inactive period of $\overline{\text{CS}}$ is the sum of the three timing parameters; Setup time (t_s), Zero time (t_z) and the Hold time (t_H):

$t_{CSIW} = t_S + t_Z + t_H$.

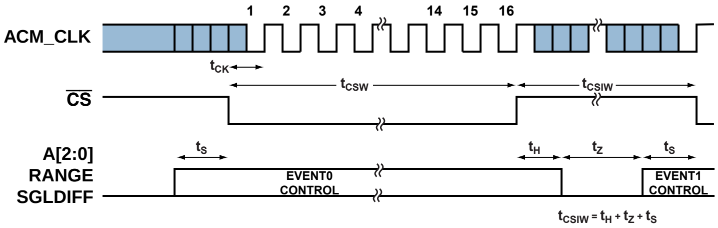


Figure 24-3: Timing Reference

Proper specification of the values of these three parameters yields the desired inactive period of $\overline{\text{CS}}$.

The ADC provides a predictable latency from the occurrence of an internal event to the assertion of an external ADC sampling event. The ADC controls and drives the $\overline{\text{CS}}$ signal on the rising edge of SCLK0_0. Therefore, the setup time (T_s) of these signals is specified in terms of SCLK0_0. However, the hold-time and zero-time are specified in terms of `ACM_CLK` cycles.

To achieve an accurate timing relationship between the $\overline{\text{CS}}$ and `ACM_CLK` (which is normally a free running clock) signals, the `ACM_CLK` signal is realigned with the active edge of $\overline{\text{CS}}$. This realignment of the `ACM_CLK` signal ensures that the setup time of the first active edge of `ACM_CLK`, relative to the active edge of $\overline{\text{CS}}$, is at least 1 `ACM_CLK` cycle.

The following set of figures in the cases show various scenarios of `ACM_CLK` realignment. All of these figures assume an *ACM_CLK:SCLK* ratio of 1:4.

- Case 1—Chip select asserted during the high phase of `ACM_CLK`
- Case 2—Chip select asserted during the low phase of `ACM_CLK`
- Case 3—Chip select asserted right before the falling edge of `ACM_CLK`
- Case 4—Chip select asserted right before the rising edge of `ACM_CLK`
- Case 5—`ACM_CLK` polarity set to 1 (`ACM_CTL.CLKPOL` =1)

Realignment of the ACM clock causes the suppression or extention of clock edges, leading to duty cycle variation. Ensure that systems interfacing with the ACM can tolerate such duty cycle variation.

The set of figures shows both the ACM-generated $\overline{\text{CS}}$ signal, which is output externally onto the appropriate `ACM_FS` pin, and the serial port receive frame sync signal. This internal signal is routed to the frame sync input of the appropriate SPORT.

The ACM clock polarity can be configured using the `ACM_CTL.CLKPOL` bit. After the $\overline{\text{CS}}$ signal is asserted, the first edge of `ACM_CLK` can be configured to be either rising edge or falling edge. Also, by default the clock is free running; it is possible to gate the ACM clock during an inactive $\overline{\text{CS}}$ period using the `ACM_CTL.CLKMOD` bit.

Case 1—Chip Select Asserted During the High Phase of `ACM_CLK` (`ACM_CTL.CLKPOL` =0)

The ***Chip Select Asserted During the High Phase of*** `ACM_CLK` figure shows the realignment of `ACM_CLK` when $\overline{\text{CS}}$ is asserted during the high phase of `ACM_CLK`. The first edge of `ACM_CLK`, after the assertion of $\overline{\text{CS}}$, is the falling edge.

The two reference clock signals (REF ACLK1 and REF ACLK2) illustrate how the `ACM_CLK` signal can be generated from a free running clock (REF ACLK1). This setup meets the timing requirements between the `ACM_CLK` and $\overline{\text{CS}}$ signals. The REF ACLK2 signal is based on the free running clock REF ACLK1. However, the REF ACLK2 signal is adjusted such that its period is immediately reset upon the assertion of $\overline{\text{CS}}$. In the resulting `ACM_CLK` signal, the time from the active edge of $\overline{\text{CS}}$ to the falling edge of `ACM_CLK` is constant at a period of 1 `ACM_CLK` cycle. See `ACM_CLK` in the ***Chip Select Asserted During the High Phase of*** `ACM_CLK` figure.

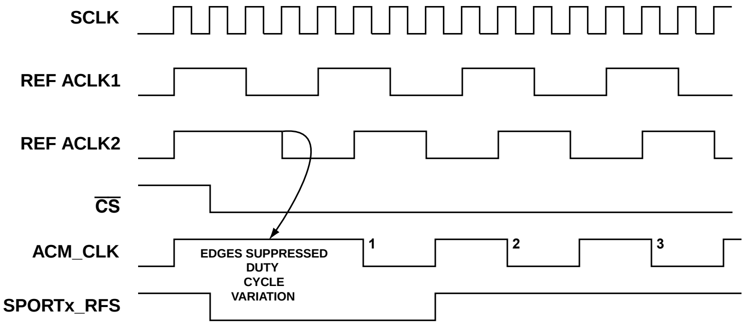


Figure 24-4: Chip Select Asserted During the High Phase of `ACM_CLK`

Case 2—Chip Select Asserted During the Low Phase of `ACM_CLK` (`ACM_CTL.CLKPOL` =0)

Refer to the ***Chip Select Asserted During the Low Phase of*** `ACM_CLK` figure. When the $\overline{\text{CS}}$ signal is asserted during the low phase of `ACM_CLK`, `ACM_CLK` is immediately pulled high which causes a duty cycle variation. In this case, (similar to Case 1), the time from the active edge of $\overline{\text{CS}}$ to the falling edge of `ACM_CLK` is 1 `ACM_CLK` period.

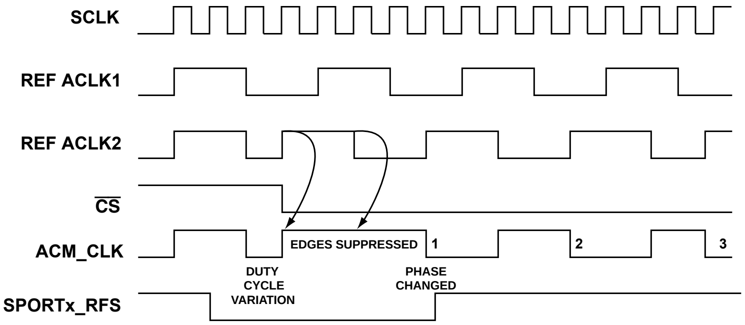


Figure 24-5: Chip Select Asserted During the Low Phase of `ACM_CLK`

Case 3—Chip Select Asserted Right Before the Falling Edge of `ACM_CLK` (`ACM_CTL.CLKPOL` =1)

Refer to the ***Chip Select Asserted Right Before the Falling Edge of*** `ACM_CLK` figure. When $\overline{\text{CS}}$ is asserted right before the falling edge of `ACM_CLK`, the falling edge of `ACM_CLK` is suppressed. This functionality ensures that the time from the active edge of $\overline{\text{CS}}$ to the falling edge of `ACM_CLK` is constant at a period of 1 `ACM_CLK` cycle.

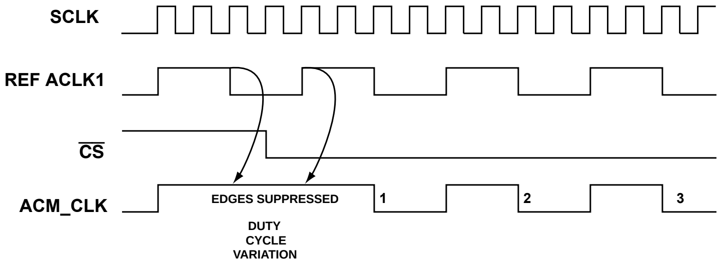


Figure 24-6: Chip Select Asserted Right Before the Falling Edge of `ACM_CLK`

Case 4—Chip Select Asserted Right Before the Rising Edge of `ACM_CLK` (`ACM_CTL.CLKPOL` =0)

Refer to the ***Chip Select Asserted Right Before the Rising Edge of*** `ACM_CLK` figure. When $\overline{\text{CS}}$ is asserted right before the rising edge of `ACM_CLK`, the high phase of `ACM_CLK` is extended. This extension ensures that the time from the active edge of $\overline{\text{CS}}$ to the falling edge of `ACM_CLK` is constant at a period of 1 `ACM_CLK` cycle.

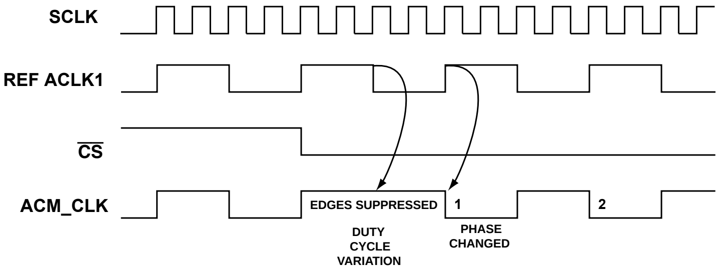


Figure 24-7: Chip Select Asserted Right Before the Rising Edge of `ACM_CLK`

Case 5— `ACM_CLK` Polarity Set to 1 (`ACM_CTL.CLKPOL` =1)

When the `ACM_CLK` polarity is set to 1 (the `ACM_CTL.CLKPOL` bit is set to 1), the first `ACM_CLK` edge after the assertion of $\overline{\text{CS}}$ is the rising edge. The ACM ensures that the time from the active edge of $\overline{\text{CS}}$ to the rising edge of `ACM_CLK` has a constant duration of 1 `ACM_CLK` cycle. The ***Polarity Set to 1*** figure shows an example diagram of the case where `ACM_CTL.CLKPOL` =1.

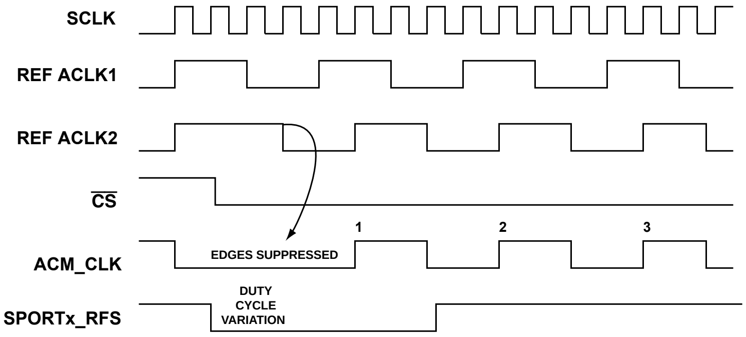


Figure 24-8: Polarity Set to 1

ACM Architectural Concepts

The following sections provide information on the architecture of the ACM module.

Clocking

The ACM on the ADSP-SC58x/ADSP-2158x processors operate on the SCLK0_0 domain. The term SCLK in this chapter is a generic reference to SCLK0_0. For more details on SCLK0_0 programming refer to CDU Functional Description.

Block Diagram

The ADC Control Module (ACM) consists of two independent 32-bit ACM timers, 16 event register pairs, 16 event comparators and a timing generation unit.

The ACM can accept four trigger inputs (internal as well as external signals). On receiving a valid trigger on selected trigger input, the ACM timer/s start counting (based on the mode of the ACM). The trigger input can be independently selected for each timer.

Two sets of eight event register pairs (a total of 16 event register pairs) determine the ADC controls for each ADC sample. The registers also determine when the sample occurs. The event register pair consists of the event control register (`ACM_EVCTL[n]`) and the event time register (`ACM_EVTIME[n]`). The event comparators unit compares the timer of the ACM count with the event time of associated enabled events. When the count matches, the event comparators unit signals to the timing generation unit. The unit starts handling the events by driving the $\overline{CS}$ and `ACM_A4` to `ACM_A0` signals accordingly.

The ***ACM Block Diagram*** shows the structure of the ACM. The following sections discuss these blocks in detail.

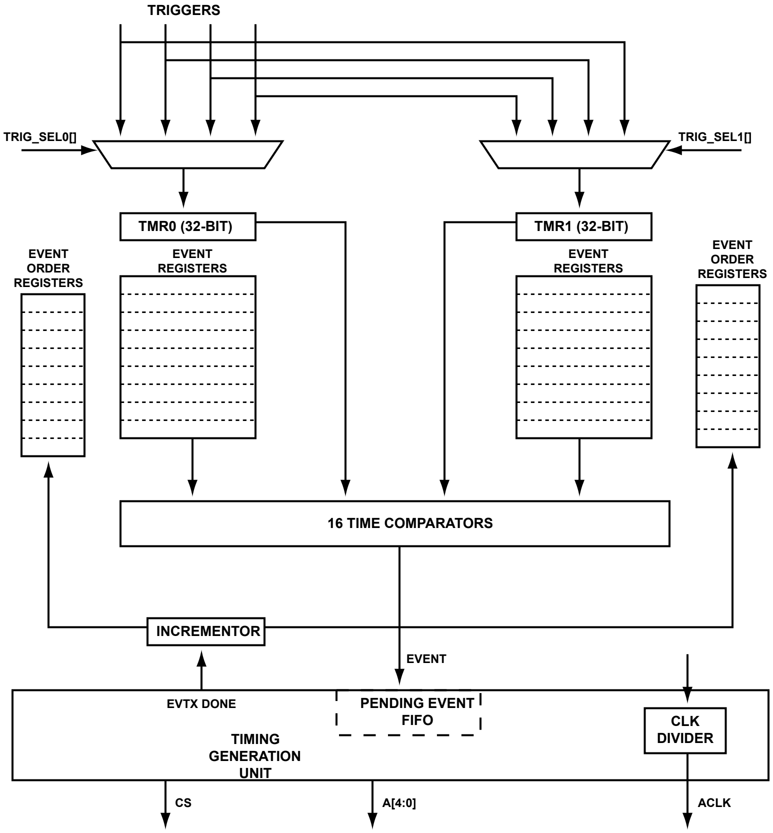

TRIGGERS
TRIG_SEL0[]
TRIG_SEL1[]
TMR0 (32-BIT)
TMR1 (32-BIT)
EVENT ORDER REGISTERS
EVENT REGISTERS
EVENT REGISTERS
EVENT ORDER REGISTERS
16 TIME COMPARATORS
INCREMENTOR
EVENT
EVTX DONE
PENDING EVENT FIFO
TIMING GENERATION UNIT
CLK DIVIDER
CS
A[4:0]
ACLK

Figure 24-9: ACM Block Diagram

Trigger Inputs

The ACM can accept four trigger inputs, based on which the ACM timers start running at the SCLK0_0 rate. The ACM contains three 32-bit internal timers (timer 0, timer 2, and timer 3). Each timer can be independently configured to use one of these trigger inputs. The ACM uses `ACM_CTL.TRGSEL0` and `ACM_CTL.TRGSEL1` bit fields to select the trigger input for Timer0 and Timer1 respectively.

Out of the 4 trigger inputs to the ACM design:

- `ACM_T0` is driven from the chip I/O
- `ACM_T2` and `ACM_T3` are driven from the TRU

The ACM uses two trigger inputs when both ACM timers are enabled for different trigger inputs. However, it uses only one trigger if both ACM timers are enabled for same trigger input or if a single ACM timer is enabled. The non-selected trigger inputs are *do-not-care* for the ACM. At most two, and at least one selected trigger input must be active in the system for the ACM to start operation.

The following list briefly describes the possible trigger inputs:

- Trigger input 0 (`ACM_T0`) – PE8:

Trigger input 0 is sourced from the PE8 pin of the GPIO port E (sometimes also referred as GPIO[72]). When the ACM is enabled, the input tap on the PE8 pin is enabled. The ACM trigger input can be from any source (internal or external) depending on the PE8 pin configuration.

When the PE8 pin is configured in GPIO mode (`PORT_FER` =0), the source of the GPIO signal can be either internal or external. The source depends on the GPIO direction, configured in the E `PORT_DIR` register. When the PE8 pin is configured in function mode (`PORT_FER` =1), the trigger 0 input is sourced in from peripheral signals. The source is based on the `PORT_MUX` register setting for the PE8 pin. For example, the ACM can source this trigger input from the `PWM_SYNC` signal if the multiplex bits for the PE8 pin are set to b#00. The PWM unit internally generates the `PWM_SYNC` signal or it can be provided externally. If the multiplex bits for PE8 are configured to b#01, the trigger input is sourced from the `EPPI_FS1` signal. The EPPI0 unit internally generates this signal or it can be generated externally.

Consider the source of the trigger input when programming the `PORT_FER.PX8` and `PORT_MUX.MUX8` bits for the PE8 pin. Enabling the input tap ensures that the ACM operation does not interfere with the module driving the PE8 pin.

- Trigger input 2 (`ACM_T2`) – (PG5) – ACM Slave TriggerID 135:

  The trigger routing unit (TRU) of the processor provides system-level sequence control without core intervention. When this trigger input mode is selected, the ACM acts as a trigger slave and accepts a trigger through the trigger slave ID-135. The slave-select (`TRU_SSR[n].SSR`) field of the SSR 135 register can be configured to receive triggers from a specific trigger master. In this way, `ACM_T2` trigger input can accept triggers asserted by that particular trigger master or through software. The software writes the ID of that trigger master to one of the four fields in the `TRU_MTR` register. The trigger response from selected master is internally routed to ACM trigger input. This way, ACM slave trigger ID 135 can receive any one of the 86 internal triggers available.

- Trigger input 3 (`ACM_T3`) – ACM Slave TriggerID 136:

  Similar to the `ACM_T2` input, the TRU of the processor provides the ACM trigger input 3 (`ACM_T3`) internally. When this trigger input mode is selected, the ACM acts as trigger slave and accepts a trigger through trigger slave ID 136. The slave-select (`TRU_SSR[n].SSR`) field of the SSR 136 register can be configured to receive triggers from a specific trigger master. In this way, the ACM slave trigger ID 136 can accept triggers asserted by that particular trigger master or through software. The software writes the ID of that trigger master to one of the four fields in the `TRU_MTR` register. The trigger response from selected master is internally routed to the ACM trigger input. This way, ACM slave trigger ID 136 can receive any one of the 86 internal triggers available.

  See the Trigger Routing Unit (TRU) chapter for more details about trigger slaves and trigger masters.

For all trigger input signals, the active edge of the trigger is programmable in the ACM control register as either rising edge or falling edge trigger.

The ***Detailed ACM Trigger Generation Logic*** figure shows the detailed ACM trigger generation logic.

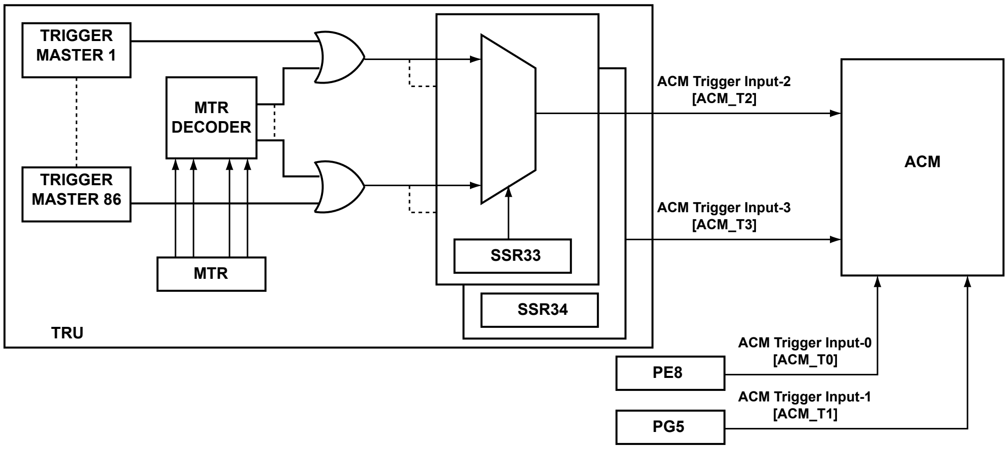

TRIGGER MASTER 1
MTR DECODER
TRIGGER MASTER 86
MTR
SSR33
SSR34
TRU
ACM Trigger Input-2 [ACM_T2]
ACM Trigger Input-3 [ACM_T3]
ACM
ACM Trigger Input-0 [ACM_T0]
PE8
ACM Trigger Input-1 [ACM_T1]
PG5

Figure 24-10: Detailed ACM Trigger Generation Logic

When the processor uses external trigger sources external for triggering the ACM timers, the minimum pulse width for these sources must be greater than one SCLK0_0 period. (For example, `ACM_T1`:`ACM_T0`).

NOTE: A latency of no more than four SCLK0_0 cycles exists between external trigger and ACM timer starts counting. Refer to the ACM Event Handling Latency section for further details.

Timers

The ACM module has two independent 32-bit timers (ACMTMR0 and ACMTMR1) which start counting at the system clock (SCLK0_0) rate when a valid trigger is detected on a selected trigger input. The timers are independently enabled using the timer enable bits (`ACM_CTL.TMR0EN`, `ACM_CTL.TMR1EN`). The timers can be independently configured for one of the four trigger inputs with configurable polarity of the signal. Enable at least one ACM timer for proper ACM operation.

By default, each ACM timer has 8 events associated with it. If both timers are enabled, Event[7:0] are associated with ACMTMR0 while Event[15:8] are associated with ACMTMR1. However, if only one timer is enabled, all of the event registers are associated with that particular timer. For example, if only ACMTMR1 is enabled (if `ACM_CTL.TMR0EN` =0 and `ACM_CTL.TMR1EN` =1), ACMTMR1 handles all 16 events, Event[15:0].

The timers start counting when a trigger input occurs that is selected for that particular timer. The timer only stops counting under one of the following conditions:

1. A timer rollover occurs.
2. All the events associated with the trigger have completed.

A timer rollover cannot occur unless the event time register of an event is programmed at some point after the trigger occurs. This practice is contrary to ACM programming guidelines.

In the second case, the exact time at which the timer stops counting depends on the FIFO state when the last event occurred internally.

If a trigger occurs while the timer is counting, the time resets and starts counting again. In this case, some of the events can miss which results in flagging the appropriate status bit and optionally an event missed interrupt.

When an ACM timer is disabled or the ACM itself is disabled, the timer resets to zero.

Event Register Pairs

An event for the ACM is a point in time where ADC sampling occurs on a particular channel of the ADC with the specified control settings of the ADC.

The ACM can handle a total of 16 events which are grouped into two sets of 8 events. The 8 events can be assigned to each of the timers (if both timers are enabled) or all 16 events can be assigned to one particular timer. (Assignment happens when only one timer is enabled). See the Timers sections for details.

All events can be independently configured and enabled. The enabled events determine the ADC controls and timing for each ADC sampling interval. Each event consists of a register pair with an event control register (`ACM_EVCTL[n]`) and an event time register (`ACM_EVTIME[n]`). The `ACM_EVCTL[n]` register enables a particular event and determines settings for the ADC control lines (`ACM_A[4:0]`) for that particular ADC conversion. The ACM uses the `ACM_EVTIME[n]` register to determine the time offset from the corresponding ACM timer trigger input to the start of that particular event. (The time offset is when the event occurs based on trigger input.) This time offset can be specified in terms of the system clock of the processor.

Enable at least one event associated with an enabled ACM timer, for ACM to execute ADC sampling.

Event Comparators Unit

The event comparators block consists of 16 event time comparators which determine when an enabled event could happen. After detecting valid trigger on selected trigger input, ACM timer starts running at *SCLK* rate. The comparators compare the ACM timer count with the event time specified in the `ACM_EVTIME[n]` register of the enabled event. If the time value matches, the comparators indicate an active event signal to the timing generation unit.

If an event happens when another event is ongoing, the occurred event is stored in the pending event FIFO of timing generation unit. If more than one event associated with an ACM timer is active during the same SCLK0_0 cycle, only the highest priority event is processed. All other events are missed (even if there was space in the pending event FIFO). However, if both timers are enabled and if multiple events associated with both ACM timers are active at the same SCLK0_0 cycle, then two events are signaled. One highest priority active event for each timer is signaled.

The priority of events is fixed; the event with the lowest event ID has higher priority compared to other events.

When both ACM timers are enabled, Event0 has the highest priority and Event7 has the lowest priority in the Event[0:7] group associated with ACMTIMER0. Similarly, Event8 has the highest priority and Event15 has the lowest priority in the Event[8:15] group associated with ACMTIMER1. So if Event1 and Event5 occur simultaneously, then Event5 is missed, even if space exists in the pending FIFO. But between the Event[0:7] and Event[8:15] group, simultaneous events can be written into the FIFO.

For example, if Event0 and Event9 occur together, then both are written into the FIFO. The Event[0:7] group has higher priority. Therefore, the order of events in the FIFO is Event0 first and then Event9. If Event1, Event5, Event9, Event15 occurred together, then Event5, and Event15 are missed. Event1 and Event9 are put into the FIFO (Event1 first followed by Event9). When both timers trigger events simultaneously, the event triggered by ACMTMR0 is given higher priority.

When only a single ACM timer is enabled, then all 16 events, Event[0:15], are assigned to that timer. Event0 has highest priority; while Event15 has lowest priority. In this case, if Event1, Event5, Event9, Event15 happened together, then only Event1 is placed in forwarded. Event5, Event9, and Event15 are missed, even if space exists in the pending FIFO.

If an event is missed, the `ACM_STAT.EMISS` bit and the corresponding bit in the ACM event missed status register (`ACM_MEVSTAT`) are set.

Timing Generation Unit

After event signaling from event comparators, the timing generation unit initiates an ADC sampling interval as per settings of that particular event. It generates ADC control signals based on the `ACM_EVCTL[n].EPF` bit field setting. The ACM uses timing registers (`ACM_TC0`, `ACM_TC1`) to determine the timing of output signals (`ACM_CLK`, $\overline{\text{CS}}$, `A[4:0]`). These registers contains the fields for set-up time, hold time and zero time for the output signals and ACM clock divider.

The pending FIFO is part of the timing generation unit. If an event occurs when the ACM is busy with another event, the occurred event is stored in the pending event FIFO. This pending event is serviced (for example, the ACM starts an ADC conversion for the event that occurred), after completion of an ongoing event.

The pending event FIFO has a depth of 4, so it can hold up to four pending events. If an event occurs when the pending event FIFO is full, that event is missed. If an event is missed, the `ACM_STAT.EMISS` bit is set and the corresponding bit in the `ACM_MEVSTAT` register also is set.

When the ACM is disabled, all pending entries in the pending FIFO are flushed.

Status Flags and Interrupts

The ACM provides a read-only status register (`ACM_STAT`) to check the module activities. Activities include identifying which event is being serviced, whether any event has been missed, or whether all the events have been serviced for the current trigger.

In addition to this register, the ACM also provides two general-purpose status registers, the ACM event completion status register (`ACM_EVSTAT`) and the ACM missed event status register (`ACM_MEVSTAT`). The `ACM_EVSTAT` register specifies how to service a completed enabled event for a particular trigger cycle. The `ACM_MEVSTAT` register specifies which enabled event has been missed for that particular trigger cycle. This information is provided for all 16 events through individual bits.

Based on these status bits, the ACM can generate two interrupts, event-completed or event-missed, for each event. These interrupts can be selectively enabled for particular ACM events through the ACM completed event interrupt mask register (`ACM_EVMSK`) and the ACM missed event interrupt mask register (`ACM_MEVMSK`).

The event completion interrupt is generated only after the entire event completes externally. (For example, when the $\overline{\text{CS}}$ signal goes inactive, and the hold time (`TH`) and zero time (`TZ`) periods are completed for that particular event). The ACM uses the `ACM_EVSTAT` register to provide the status of each event indicating which event has caused the interrupt. It is also possible to generate this interrupt when all the events associated with an ACM timer are completed for an ACM trigger cycle. Write the relevant W1C (write 1 to clear) bit in the `ACM_EVSTAT` register to clear this interrupt.

The event missed interrupt is generated when an enabled event is missed for a trigger cycle and the corresponding mask bit in the `ACM_MEVMSK` register is set. The event comparators unit can miss the event when more than one event, related to the same timer, are active during the same SCLK0_0 cycle. The timing generation unit can miss when an event occurred and the event pending FIFO was full. The ACM uses the `ACM_MEVSTAT` register to provide the status of each missed event indicating which event-miss caused the interrupt. Write the relevant W1C bit in the `ACM_MEVSTAT` register to clear this interrupt.

NOTE: A status bit set either in the `ACM_EVSTAT` or `ACM_MEVSTAT` registers triggers an interrupt only if the corresponding bit in the `ACM_EVMSK` or `ACM_MEVMSK` registers is enabled.

The ACM provides an event completion interrupt upon completion of particular event or all events. The ACM can also provide a trigger output to the trigger routing unit (TRU) of the processor. Trigger slaves can use this trigger output for their operations without requiring core intervention.

Event Order Registers

For debugging purpose, ACM hardware includes 16 event order registers, one per each event. These registers indicate the order in which the events complete externally.

These registers are denoted as `ACM_EVORD[n]`, where n stands for event ID, 0–15. The ACM module uses the 8-bit `ACM_EVORD[n].ORD` field of this register to indicate the order the ADC data has been captured for the event. This field accumulates the order count every trigger cycle, unless it is cleared in the software. At each trigger cycle, the values of the register are updated. The ACM must read the register at the end of each trigger cycle (as it writes the new order value of the event in the next trigger cycle). Thus, the 8-bit field can store the order of 256 data captures at a stretch, after which it starts the order count from zero again.

All the event registers can be reset in software by setting the `ACM_CTL.ORST` bit. When set, it clears all the register values to zero. This bit is auto-cleared to 0 after all `ACM_EVORD[n]` registers are cleared. The ACM can select trigger inputs of the timers to automatically clear these registers, when the `ACM_CTL.AOREN` bit is set. The ACM uses the `ACM_CTL.OTSEL` bit to determine which trigger input to select for this auto-clearing.

The event order functionality is demonstrated with an example where the ACM has only three events enabled (Event1, Event7, and Event13). The ***Event Order Timing*** figure shows the event order register value of these events at different stages.

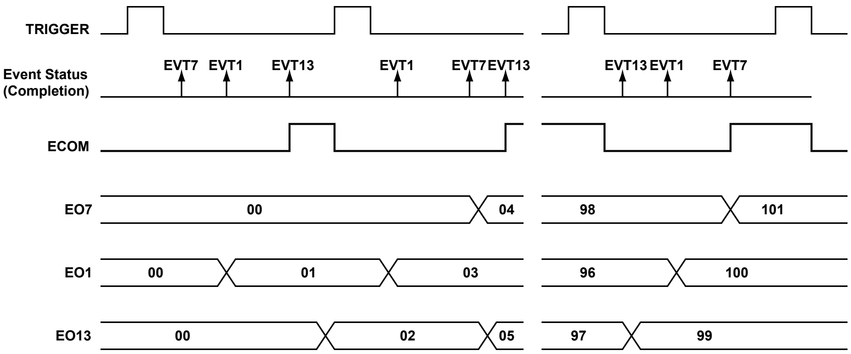


Figure 24-11: Event Order Timing

ACM Operation

The ***ACM Operation, Two Events*** figure shows the ACM operation where only two events (Event0 and Event3) are enabled. The line labeled "ADC Controls" depicts the timing of the ADC control signals: `ACM_A4` through `ACM_A0`.

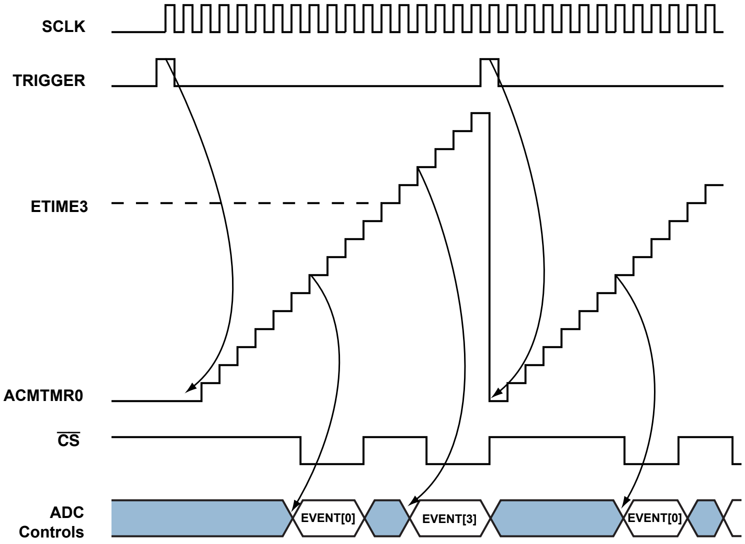


Figure 24-12: ACM Operation, Two Events

The ***ACM Operation, Two Events*** figure depicts a usage case. The `ACM_EVTIME[n]` 3 time register is programmed with a count value that is greater than the value programmed in Event0. So, Event3 occurs after Event0. There are, however, no restrictions on the order of the different events. Event0 time can be greater than, less than or equal to Event3 time.

If the value in Event0 time register is less than the value in Event3 time, Event3 occurs after Event0. If the value in Event0 is greater than the value in Event3, then Event3 occurs before Event0. If Event0 time is equal to the Event3 time, the events are processed according to their priorities. Only the highest priority event is processed. Other lower-priority events are missed (even if there is a space in pending event FIFO), as the same ACM timer handles both events. Event0 has the highest priority. So in this case, the timing generation unit processes Event0, while Event3 is missed. In this case, the Event3 missed status bit (`ACM_MEVSTAT.EV3`) is set and the event missed bit (`ACM_STAT.EMISS`) is set, indicating that an event has been missed.

If event times are not sufficiently spaced apart, an event could occur while a previous event is underway (while the chip select of the previous event is asserted). In this situation, the second event is queued in the pending event FIFO. If the pending event FIFO is full, the event is not queued and is missed. This situation happens most commonly when enabling both ACM timers with different trigger inputs and the sources of these triggers are not synchronized together. In this case, it is possible that the events controlled by the two timers overlap. It is therefore important to consider the possibility of events occurring either simultaneously or being missed when enabling events on two asynchronously triggered timers. It is the programs responsibility to ensure that the values in the event time registers do not lead to event misses. The bits corresponding to the missed event in the `ACM_MEVSTAT` and `ACM_STAT` registers are flagged on a missed event.

When both ACM timers are enabled and they triggered the events simultaneously, the event triggered by `ACM_TMR0` has the higher priority. For example, when an `ACM_TMR0` event (one of events 0 through 7) and `ACM_TMR1` event (one of events 8 through 15) occur simultaneously:

- The timing generation unit processes the `ACM_TMR0` event, or
- The `ACM_TMR0` event is queued in the pending event FIFO before the processing or the queuing of the `ACM_TMR1` event.

When all the events enabled for a given ACM timer are processed, the ACM timer stops incrementing. (This timer action is not reflected in the ***ACM Operation, Two Events*** figure). Also, the corresponding event completion bit (`ACM_STAT.ECOM1`, `ACM_STAT.ECOM0`) is flagged. The same bit is also reflected in the status register for event compilation (`ACM_EVSTAT`). This register can optionally generate the event completion interrupt when the corresponding bit in the event completion interrupt mask register (`ACM_EVMSK`) is set. Two separate bits are available, one for each ACM timer.

The processor can use the ACM to generate various sequences of ADC sampling events through appropriate programming of event time registers, event control registers, and triggers. For more information, see the use cases described in Emulation Mode Use Case.

ACM Programming Concepts

Since the ACM module is used with the SPORT, PWM, GP timer, and GPIO, the programming must comply with the following guidelines for reliable operation of the ACM.

- The ACM is a control module and provides clock and chip select and control signals with required timing. But for capturing the data from ADC, the ACM uses one of the halves of SPORT1.
- Enable the ACM before enabling the SPORT. The SPORT can be configured before enabling the ACM. Configure the SPORT in slave mode [external clock (`SPORT_CTL_A.ICLK` =0), external frame sync (`SPORT_CTL_A.IFS` =0)] as receiver.

  The timings of external ADC decide the settings of `SPORT_CTL_A.LFS`, `SPORT_CTL_A.LAFS`, `SPORT_CTL_A.CKRE` bits. Generally, the SPORT is configured in DSP serial mode to receive the ADC samples, but other operating modes (such as multichannel) are possible.

  If the ACM is programmed in gated clock mode (`ACM_CTL.CLKMOD` =1), set the serial port also in gated clock mode (`SPORT_CTL_A.GCLKEN` =1).
- DMA mode of SPORT operation is preferred, as it saves the processor MIPS when receiving chunks of data. However, receiving ADC samples in core mode is also possible. When using DMA mode, configure the DMA registers of the selected SPORT appropriately and enable DMA before enabling the SPORT. When using the primary and secondary channels of a SPORT to receive data from two ADC channels, use the 2D feature of DMA to de-interleave the data from two channels. When using core mode of SPORT operation, register the core handler to handle the data read requests from the SPORT receiver.

- In addition to SPORT register settings, first set the DAI registers to enable: SPORT data pins, the ACM clock, the $\overline{\text{CS}}$, and the data pins. Then, configure the PORT registers for the trigger and control pins. When using either of the `ACM_T1`/`ACM_T0` trigger inputs for ACM timers, configure the `PORT_FER` and `PORT_MUX` bits of the corresponding pins (PE8 or PG5) according to source of trigger input.
- When using ACM_T[2:3] trigger inputs for ACM timers, the ACM can configure and enable the trigger routing unit (TRU) at this step. Program the corresponding slave trigger ID using the `TRU_SSR[n]` register to select the required master trigger. When using these trigger inputs, do not configure the `TRU_SSR[n]` register when the ACM is enabled. The default value of this register is zero and the master trigger ID 0 is system reserved.
- Before enabling the ACM (by setting the `ACM_CTL.EN` bit), program all the control bits of ACM control register. These control bits include ACM trigger selects (`ACM_CTL.TRGSEL1`/`ACM_CTL.TRGSEL0`), trigger input polarities (`ACM_CTL.TRGPOL1`/`ACM_CTL.TRGPOL0`), $\overline{\text{CS}}$ signal polarity (`ACM_CTL.CSPOL`), ACM clock polarity (`ACM_CTL.CLKPOL`), ACM clock mode (`ACM_CTL.CLKMOD`), and serial port unit selection and the `ACM_EVORD[n]` register settings.
- Configure the ACM timing control registers to define the ACM clock frequency and the setup, hold, and zero time of the ACM control signals.
- Program the timer enabled bits (`ACM_CTL.TMR0EN`/`ACM_CTL.TMR1EN`) together only after the ACM is enabled. Once the bits are programmed, do not change them. Modifying these enable bits in the ACM control register is not recommended while the ACM is in operation. Doing so can cause events to change dependency from one timer to the other and can cause the values in the ACM status registers (`ACM_EVSTAT` and `ACM_STAT`) to be inaccurate. If both timers are required for use, enable them together after the ACM is enabled. If one timer is already enabled, disable and then reenable the ACM and then program both timer enable bits together. Similarly, when both timers are running, disable them together.
- Once the peripherals have been configured, enable the ACM first and then the SPORT DMA and finally the SPORT module itself. Ideally, a trigger should not be active when enabling the ACM.
- After enabling the ACM, configure and enable the event register pairs (event control and event time registers) to create the required events.
- ACLK is an external clock relative to the SPORT peripheral. Observe any SPORT requirements around a minimum number of stable external clock cycles before the assertion of the first SPORT frame sync. The SPORT requires a minimum of 3 clock cycles before it is able to recognize a frame sync. When the SPORT is configured in gated clock mode, this requirement becomes a minimum of 7 SPORT clock cycles. The required number of ACLK cycles should elapse before first assertion of $\overline{\text{CS}}$. Use any of the following methods:
    - Ensure that ACM triggers are generated at least 3 ACLK cycles after the ACM is enabled.
    - Ensure that the event time value (`ACM_EVTIME[n]`) of the first active event is such that 3 ACLK cycles elapse before the event is processed.

- When the minimum number of ACLK cycles before the assertion of $\overline{\text{CS}}$ is not observed, the SPORT can miss the data of the first ADC sampling event. There is a software workaround for fulfilling this requirement. Program the `ACM_TC0.CKDIV` value after enabling the SPORT (subsequently after enabling the ACM) but before the trigger is applied. Since the default value of `ACM_TC0.CKDIV` is 1, the ACLK frequency is higher. Therefore, the SPORT can receive the required clock cycles within a short period (before the frame sync arrives).
- When using the `ACM_T2`/`ACM_T3` trigger inputs, the master can be enabled as a last step (if it is configured only to provide triggers to the ACM) to generate the triggers.
- While disabling the ACM system, disable the SPORT first, then the DMA. Finally, disable the ACM.

Emulation Mode Use Case

This section describes the usage modes of the ACM by illustrating how to implement various sequencing ADC sampling modes.

Single-Shot Sequencing Mode Emulation

In single-shot sequencing mode, all enabled events are sequentially issued one after the other on the occurrence of an ACM trigger. The sequence of events is fixed, starting with Event0 and ending with Event15.

The ***Single Shot Sequencing*** figure shows an example of single-shot sequencing mode where only Event0 and Event1 are enabled. The value ETIME0 is written into the ACM_ET0 register, and ACMTMR0 is enabled in this mode.

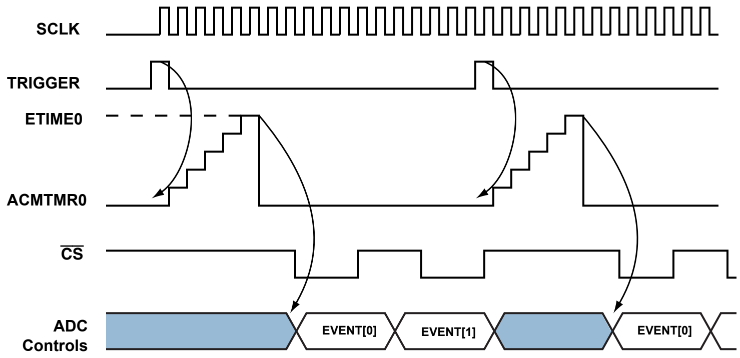


Figure 24-13: Single Shot Sequencing

To emulate this mode of operation using the ACM:

- Configure the appropriate trigger source for initiating ACM activity. Refer to ADC Control Module (ACM) for information on signals that can trigger the ACM counters.
- Enable only one ACM timer (ACMTMR0)
- Enable events and program the event time values as: Event0 time = X, Event1 time = X + Y, Event2 time = X + 2Y where:

  X = `ETIME0`, the initial time offset from trigger (if needed)

$Y = t_H + t_{CSW} + t_S + t_Z$, where t_H is the hold time, t_Z is the zero time, and t_S is the setup time for ACM control lines, as specified in ACM timing registers.

For more information, refer to the ACM External Pin Timing section.

NOTE: Y has to be slightly less than the calculated value to ensure that the next event occurs before the first event completes. Then, the next event is in the pending FIFO and enables the transitions between events without a break.

Continuous Sequencing Mode Emulation

Continuous sequencing mode is similar to single-shot sequencing mode, except in continuous sequencing the event sequencing continuously repeats. As in single-shot mode, the time offset is programmable in continuous mode. The trigger in continuous mode is relevant only for the first time. Therefore, any subsequent triggers after the first active edge of the trigger are neglected.

The ***Continuous Sequencing*** figure shows an example of continuous sequencing mode with only two events – Event0, Event1 enabled. To emulate continuous sequencing mode using ACM:

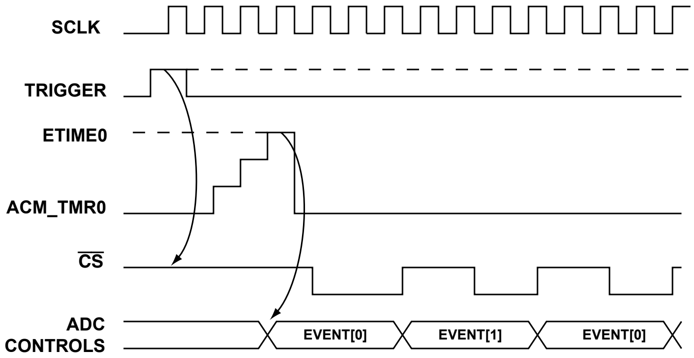


Figure 24-14: Continuous Sequencing

- Enable only one ACM timer (for example ACMTIMER0). Select a timer trigger input from the TRU (either of the `ACM_T[n]`[3:2].

  If `ACM_T2` is selected, program ACM slave trigger ID 33 to select ACM event completion (whose master trigger ID is 19) as trigger input. If `ACM_T3` is selected as trigger input, configure ACM slave trigger ID 34.

- Configure rest of the settings required by programming ACM control and timing registers.
- Enable ACM events with required ACM control lines settings.

  Program the event time registers as: Event0 time = X, Event1 time = X + Y, Event 2 time = X + 2Y where:

  X and Y values, in terms of SCLK, are as described in the single-shot case. (Y must be slightly less than $t_H + t_{CSW} + t_S + t_Z$ to avoid any break between events.)

- Configure and enable system event controller (SEC).

Also configure, map, and enable the ACM event completion interrupt. This interrupt occurs on the completion of all enabled ACM events for the current trigger. That means that programs must only set the `ACM_EVMSK.IECOM0` bit (or the `ACM_EVMSK.IECOM1` bit, if using ACMTIMER1) register.

- Enable ACM, SPORT, and SPORT DMA as per the guidelines in the ACM Programming Concepts section.
- Since the ACM trigger input is configured as ACM event completion trigger output, the first trigger is necessary to start the ACM operation. We can provide this dummy trigger by writing master trigger ID into master trigger register (`TRU_MTR`).

  Write the ACM event completion trigger ID (19) into the `TRU_MTR` register. This action triggers the ACM timers and the ACM starts handling the events.
- After completing all events, the ACM provides an event completion trigger and the corresponding interrupt is generated. The ACM trigger output is provided to the ACM timers. The timers reset their counter and start running from zero. The ACM rehandles all the enabled ACM events. This sequence continues.

  However, in order to provide the trigger outputs properly, clear the interrupt latch in the ISR by clearing the `ACM_EVSTAT.ECOM0S` (or `ACM_EVSTAT.ECOM1S`, if using ACMTIMER1) bit.

ADSP-SC58x ACM Register Descriptions

ADC Control Module (ACM) contains the following registers.

Table 24-5: ADSP-SC58x ACM Register List

Name	Description
`ACM_CTL`	Control Register
`ACM_EVCTL[n]`	Event N Control Register
`ACM_EVMSK`	Event Complete Interrupt Mask Register
`ACM_EVORD[n]`	Event N Order Register
`ACM_EVSTAT`	Event Complete Status Register
`ACM_EVTIME[n]`	Event N Time Register
`ACM_MEVMSK`	Missed Event Interrupt Mask Register
`ACM_MEVSTAT`	Missed Event Status Register
`ACM_STAT`	Status Register
`ACM_TC0`	Timing Configuration 0 Register
`ACM_TC1`	Timing Configuration 1 Register
`ACM_TMR0`	Timer 0 Register
`ACM_TMR1`	Timer 1 Register

Control Register

The `ACM_CTL` register enables and selects the various modes of operation of the ACM.

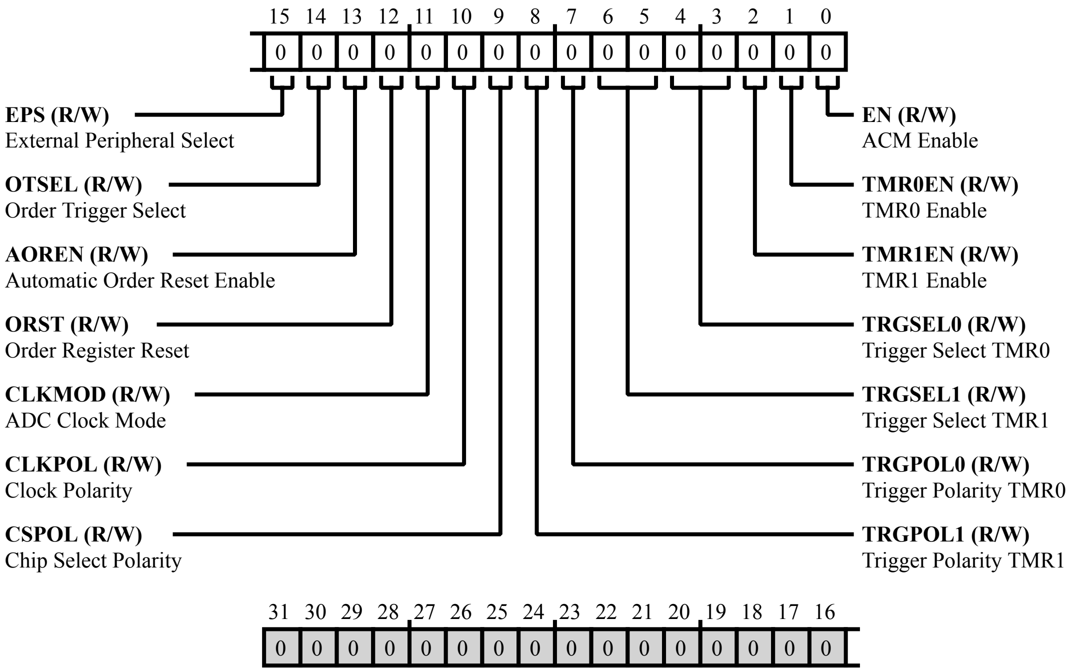

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EPS (R/W)
External Peripheral Select
OTSEL (R/W)
Order Trigger Select
AOREN (R/W)
Automatic Order Reset Enable
ORST (R/W)
Order Register Reset
CLKMOD (R/W)
ADC Clock Mode
CLKPOL (R/W)
Clock Polarity
CSPOL (R/W)
Chip Select Polarity
EN (R/W)
ACM Enable
TMR0EN (R/W)
TMR0 Enable
TMR1EN (R/W)
TMR1 Enable
TRGSEL0 (R/W)
Trigger Select TMR0
TRGSEL1 (R/W)
Trigger Select TMR1
TRGPOL0 (R/W)
Trigger Polarity TMR0
TRGPOL1 (R/W)
Trigger Polarity TMR1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 24-15: ACM_CTL Register Diagram

Table 24-6: ACM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EPS	External Peripheral Select. The `ACM_CTL.EPS` bit selects whether the ACM interfaces to half SPORT A or half SPORT B.	
		0	Half SPORT A Interfaces to ACM
		1	Half SPORT B Interfaces to ACM
14 (R/W)	OTSEL	Order Trigger Select. The `ACM_CTL.OTSEL` bit selects whether TMR0 or TMR1 triggers a reset of the event order (`ACM_EVORD[n]`) registers. This bit is applicable only if the `ACM_CTL.AOREN` bit is set.	
		0	ACM TMR0 Triggers Reset of Order Registers
		1	ACM TMR1 Triggers Reset of Order Registers

Table 24-6: ACM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	AOREN	Automatic Order Reset Enable. The `ACM_CTL.AOREN` bit enables automatic reset of the event order (`ACM_EVORD[n]`) registers, based on the selected timer trigger. The `ACM_CTL.OTSEL` bit selects the trigger.	
		0	Disable Automatic Order Reset
		1	Enable Automatic Order Reset
12 (R/W)	ORST	Order Register Reset. The `ACM_CTL.ORST` bit resets the event order (`ACM_EVORD[n]`) register value to 0. This bit auto-clears to 0 after the `ACM_EVORD[n]` registers clear.	
11 (R/W)	CLKMOD	ADC Clock Mode. The `ACM_CTL.CLKMOD` bit selects gated clock mode (`ACM_CLK` is gated when the ADC CS is inactive) or continuous (ACM generates continuous `ACM_CLK`).	
		0	Continuous Clock Mode
		1	Gated Clock Mode
10 (R/W)	CLKPOL	Clock Polarity. The `ACM_CTL.CLKPOL` bit selects whether the rising or falling edge of `ACM_CLK` comes after ADC CS becomes active.	
		0	Falling Edge of Clock After CS
		1	Rising Edge of Clock After CS
9 (R/W)	CSPOL	Chip Select Polarity. The `ACM_CTL.CSPOL` bit selects whether ADC CS is active high or low.	
		0	Active Low CS
		1	Active High CS
8 (R/W)	TRGPOL1	Trigger Polarity TMR1. The `ACM_CTL.TRGPOL1` bit selects whether the trigger polarity for ACM TMR1 occurs on the falling or rising edge.	
		0	Rising Edge Trigger
		1	Falling Edge Trigger
7 (R/W)	TRGPOL0	Trigger Polarity TMR0. The `ACM_CTL.TRGPOL0` bit selects whether the trigger polarity for ACM TMR0 occurs on the falling or rising edge.	
		0	Rising Edge Trigger
		1	Falling Edge Trigger

Table 24-6: ACM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:5 (R/W)	TRGSEL1	Trigger Select TMR1. The `ACM_CTL.TRGSEL1` bits select the external trigger for ACM TMR1.	
		0	Trigger 0 (ACM_T0 Pin)
		1	Trigger 1 (ACM_T1 Pin)
		2	Trigger 2 (Trigger Input 2 - TRU Slave)
		3	Trigger 3 (Trigger Input 3 - TRU Slave)
4:3 (R/W)	TRGSEL0	Trigger Select TMR0. The `ACM_CTL.TRGSEL0` bits select the external trigger for ACM TMR0.	
		0	Trigger 0 (ACM_T0 Pin)
		1	Trigger 1 (ACM_T1 Pin)
		2	Trigger 2 (Trigger Input 2 - TRU Slave)
		3	Trigger 3 (Trigger Input 3 - TRU Slave)
2 (R/W)	TMR1EN	TMR1 Enable. The `ACM_CTL.TMR1EN` bit enables ACM TMR1.	
		0	Disable ACM TMR1
		1	Enable ACM TMR1
1 (R/W)	TMR0EN	TMR0 Enable. The `ACM_CTL.TMR0EN` bit enables ACM TMR0.	
		0	Disable ACM TMR0
		1	Enable ACM TMR0
0 (R/W)	EN	ACM Enable. The `ACM_CTL.EN` bit enables ACM operation.	
		0	Disable ACM
		1	Enable ACM

Event N Control Register

The `ACM_EVCTL[n]` registers each hold the ADC control value corresponding to the event related to the register. These control registers each have an event enable bit, that permits a selective enable of a particular event.

Do not program the `ACM_EVCTL[n]` register when an event is active. Program this register before setting trigger and re-program this register after all the events are complete (`ACM_STAT.ECOM1` or `ACM_STAT.ECOM0` bit is set). If no events are enabled in this register (for example, all `ACM_EVCTL[n].ENAEV` bits =0, and the `ACM_STAT.ECOM0` or `ACM_STAT.ECOM1` bits are set) and an interrupt generates (if unmasked) a trigger is applied with the Timer enabled.

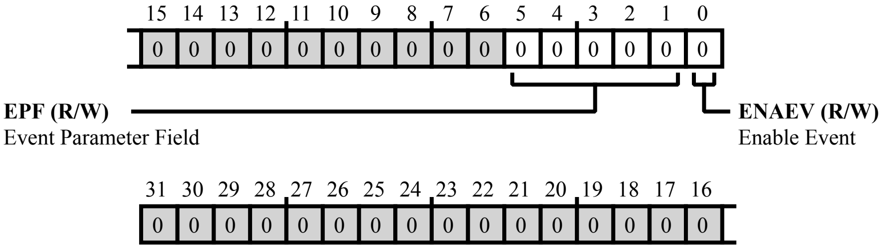


Figure 24-16: ACM_EVCTL[n] Register Diagram

Table 24-7: ACM_EVCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5:1 (R/W)	EPF	Event Parameter Field. The `ACM_EVCTL[n].EPF` bits select values for the ADC control pins (`ACM_A[n]`. These values are output when the enabled event occurs. Selection of `ACM_EVCTL[n].EPF` values are based on the type of ADC, usage mode, and other items. For more information, see the operating modes section. All `ACM_EVCTL[n].EPF` bits have the same external pin timing.	
0 (R/W)	ENAEV	Enable Event. The `ACM_EVCTL[n].ENAEV` bit causes a sampling event to occur based on the controls selected by the `ACM_EVCTL[n].EPF` bit field when an event (time comparison match or other external trigger) occurs. If disabled, the corresponding event has no significance, and the control values are not used.	
		0	Disable Event
		1	Enable Event

Event Complete Interrupt Mask Register

The ACM_EVMSK register enables interrupts corresponding to status bits in the ACM_EVSTAT register. When a bit in the ACM_EVMSK register is set (=1), an interrupt generates when the corresponding event complete bit in the ACM_EVSTAT register is set.

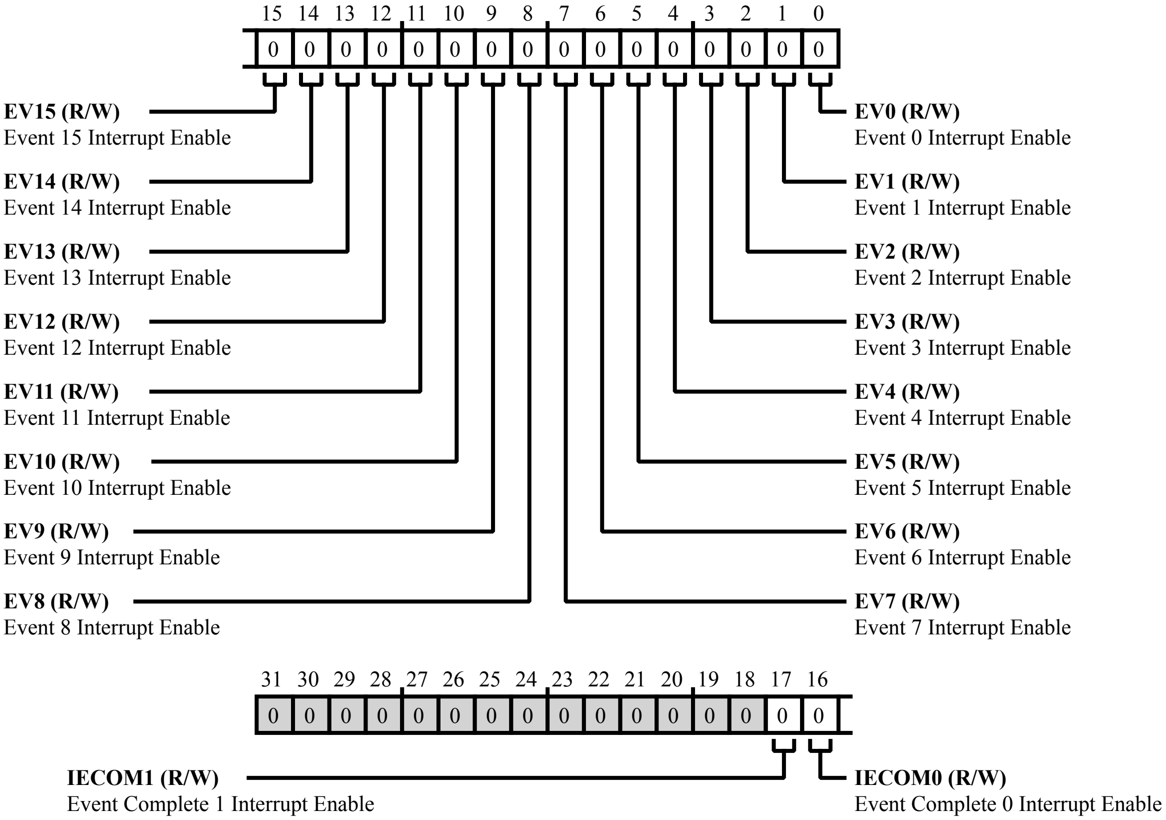

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EV15 (R/W)
Event 15 Interrupt Enable
EV14 (R/W)
Event 14 Interrupt Enable
EV13 (R/W)
Event 13 Interrupt Enable
EV12 (R/W)
Event 12 Interrupt Enable
EV11 (R/W)
Event 11 Interrupt Enable
EV10 (R/W)
Event 10 Interrupt Enable
EV9 (R/W)
Event 9 Interrupt Enable
EV8 (R/W)
Event 8 Interrupt Enable
EV0 (R/W)
Event 0 Interrupt Enable
EV1 (R/W)
Event 1 Interrupt Enable
EV2 (R/W)
Event 2 Interrupt Enable
EV3 (R/W)
Event 3 Interrupt Enable
EV4 (R/W)
Event 4 Interrupt Enable
EV5 (R/W)
Event 5 Interrupt Enable
EV6 (R/W)
Event 6 Interrupt Enable
EV7 (R/W)
Event 7 Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IECOM1 (R/W)
Event Complete 1 Interrupt Enable
IECOM0 (R/W)
Event Complete 0 Interrupt Enable

Figure 24-17: ACM_EVMSK Register Diagram

Table 24-8: ACM_EVMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	IECOM1	Event Complete 1 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
16 (R/W)	IECOM0	Event Complete 0 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-8: ACM_EVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EV15	Event 15 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	EV14	Event 14 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	EV13	Event 13 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	EV12	Event 12 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	EV11	Event 11 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	EV10	Event 10 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	EV9	Event 9 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	EV8	Event 8 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	EV7	Event 7 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	EV6	Event 6 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-8: ACM_EVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	EV5	Event 5 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	EV4	Event 4 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
3 (R/W)	EV3	Event 3 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
2 (R/W)	EV2	Event 2 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	EV1	Event 1 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	EV0	Event 0 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Event N Order Register

The `ACM_EVORD[n]` registers hold the ADC data capture event order. These registers can store the order of 256 data captures at a stretch. The `ACM_EVORD[n]` registers also have status bits indicating whether an event misses or completes in the trigger cycle.

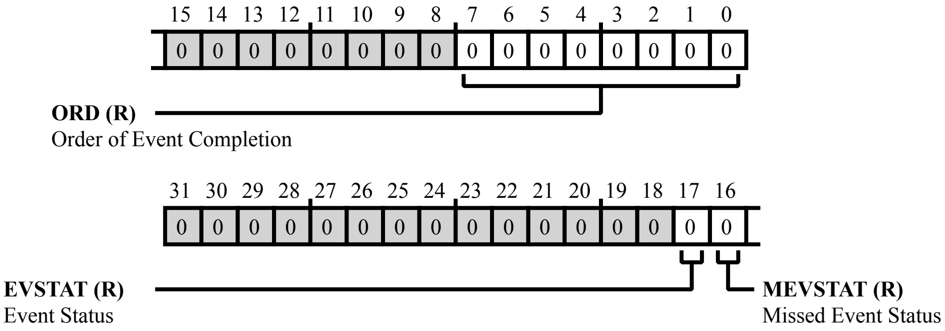

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ORD (R)
Order of Event Completion
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EVSTAT (R)
Event Status
MEVSTAT (R)
Missed Event Status

Figure 24-18: ACM_EVORD[n] Register Diagram

Table 24-9: ACM_EVORD[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	EVSTAT	Event Status. The `ACM_EVORD[n].EVSTAT` bit reflects the state of the corresponding event bit in the `ACM_EVSTAT` register.	
16 (R/NW)	MEVSTAT	Missed Event Status. The `ACM_EVORD[n].MEVSTAT` bit reflects the state of the corresponding event bit in the `ACM_MEVSTAT` register.	
7:0 (R/NW)	ORD	Order of Event Completion. The `ACM_EVORD[n].ORD` bits indicate the order of event completion. Zero indicates the first event completed (after the ACM is enabled or after the `ACM_CTL.ORST` bit is set) and 255 indicates the 256th event completed.	
		0	1st Event Completed
		1	2nd Event Completed
		255	256th Event Completed

Event Complete Status Register

The ACM_EVSTAT register identifies which enabled event has occurred for a particular trigger cycle. When an ACM_EVSTAT bit is cleared (=0), this status indicates that the ACM has not begun or completed conversion for the corresponding event (conversion not done). When an ACM_EVSTAT bit is set (=1), this status indicates that the ACM has completed conversion for the corresponding event (conversion done).

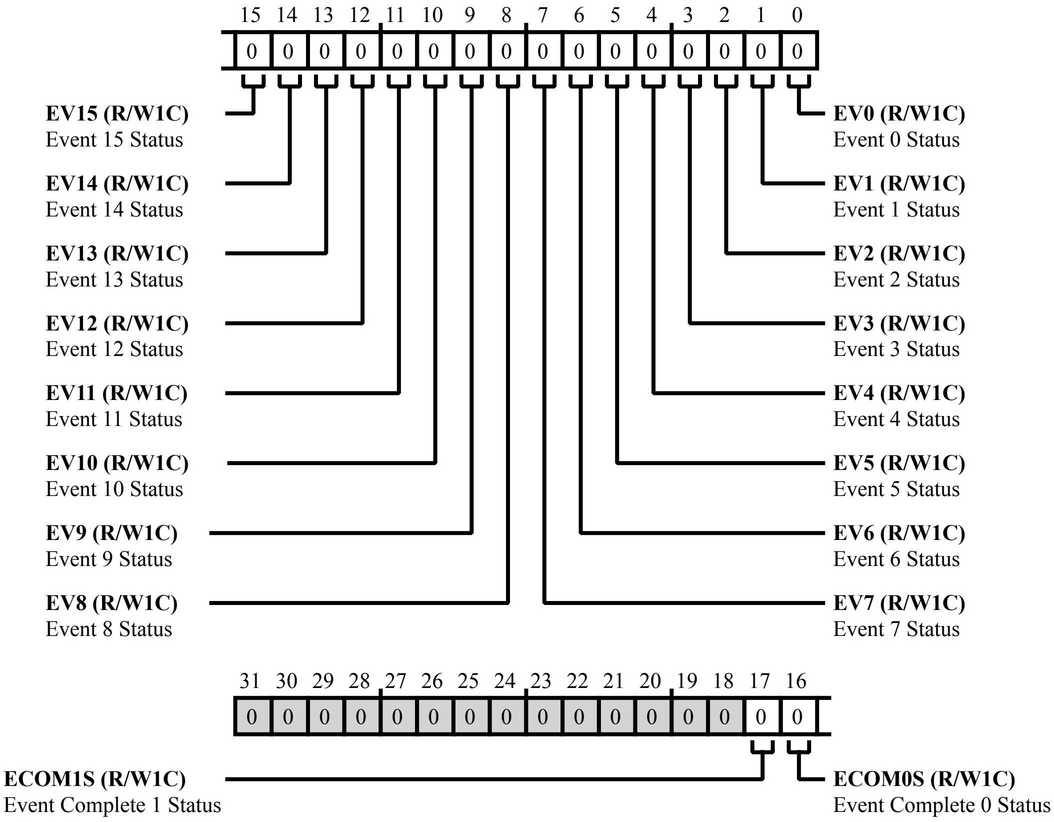

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EV15 (R/W1C)
Event 15 Status
EV14 (R/W1C)
Event 14 Status
EV13 (R/W1C)
Event 13 Status
EV12 (R/W1C)
Event 12 Status
EV11 (R/W1C)
Event 11 Status
EV10 (R/W1C)
Event 10 Status
EV9 (R/W1C)
Event 9 Status
EV8 (R/W1C)
Event 8 Status
EV0 (R/W1C)
Event 0 Status
EV1 (R/W1C)
Event 1 Status
EV2 (R/W1C)
Event 2 Status
EV3 (R/W1C)
Event 3 Status
EV4 (R/W1C)
Event 4 Status
EV5 (R/W1C)
Event 5 Status
EV6 (R/W1C)
Event 6 Status
EV7 (R/W1C)
Event 7 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ECOM1S (R/W1C)
Event Complete 1 Status
ECOM0S (R/W1C)
Event Complete 0 Status

Figure 24-19: ACM_EVSTAT Register Diagram

Table 24-10: ACM_EVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	ECOM1S	Event Complete 1 Status. The ACM_EVSTAT.ECOM1S bit indicates the state of the ACM_STAT.ECOM1 bit. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C and is not cleared by a trigger.	
		0	No Status
		1	ACM_STAT.ECOM1 =1 Occurred

Table 24-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ECOM0S	Event Complete 0 Status. The `ACM_EVSTAT.ECOM0S` bit indicates the state of the `ACM_STAT.ECOM0` bit. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C and is not cleared by a trigger.	
		0	No Status
		1	ACM_STAT.ECOM0 =1 Occurred
15 (R/W1C)	EV15	Event 15 Status. The `ACM_EVSTAT.EV15` bit indicates when the ACM has completed the conversion for event 15. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 15 Conversion
		1	Event 15 Conversion Done
14 (R/W1C)	EV14	Event 14 Status. The `ACM_EVSTAT.EV14` bit indicates when the ACM has completed the conversion for event 14. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 14 Conversion
		1	Event 14 Conversion Done
13 (R/W1C)	EV13	Event 13 Status. The `ACM_EVSTAT.EV13` bit indicates when the ACM has completed the conversion for event 13. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 13 Conversion
		1	Event 13 Conversion Done
12 (R/W1C)	EV12	Event 12 Status. The `ACM_EVSTAT.EV12` bit indicates when the ACM has completed the conversion for event 12. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 12 Conversion
		1	Event 12 Conversion Done

Table 24-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	EV11	Event 11 Status. The `ACM_EVSTAT.EV11` bit indicates when the ACM has completed the conversion for event 11. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 11 Conversion
		1	Event 11 Conversion Done
10 (R/W1C)	EV10	Event 10 Status. The `ACM_EVSTAT.EV10` bit indicates when the ACM has completed the conversion for event 10. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 10 Conversion
		1	Event 10 Conversion Done
9 (R/W1C)	EV9	Event 9 Status. The `ACM_EVSTAT.EV9` bit indicates when the ACM has completed the conversion for event 9. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 9 Conversion
		1	Event 9 Conversion Done
8 (R/W1C)	EV8	Event 8 Status. The `ACM_EVSTAT.EV8` bit indicates that the ACM has completed the conversion for event 8. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 8 Conversion
		1	Event 8 Conversion Done
7 (R/W1C)	EV7	Event 7 Status. The `ACM_EVSTAT.EV7` bit indicates when the ACM has completed the conversion for event 7. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 7 Conversion
		1	Event 7 Conversion Done

Table 24-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	EV6	Event 6 Status. The `ACM_EVSTAT.EV6` bit indicates when the ACM has completed the conversion for event 6. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 6 Conversion
		1	Event 6 Conversion Done
5 (R/W1C)	EV5	Event 5 Status. The `ACM_EVSTAT.EV5` bit indicates when the ACM has completed the conversion for event 5. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 5 Conversion
		1	Event 5 Conversion Done
4 (R/W1C)	EV4	Event 4 Status. The `ACM_EVSTAT.EV4` bit indicates when the ACM has completed the conversion for event 4. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 4 Conversion
		1	Event 4 Conversion Done
3 (R/W1C)	EV3	Event 3 Status. The `ACM_EVSTAT.EV3` bit indicates when the ACM has completed the conversion for event 3. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 3 Conversion
		1	Event 3 Conversion Done
2 (R/W1C)	EV2	Event 2 Status. The `ACM_EVSTAT.EV2` bit indicates when the ACM has completed the conversion for event 2. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 2 Conversion
		1	Event 2 Conversion Done

Table 24-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	EV1	Event 1 Status. The `ACM_EVSTAT.EV1` bit indicates when the ACM has completed the conversion for event 1. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 1 Conversion
		1	Event 1 Conversion Done
0 (R/W1C)	EV0	Event 0 Status. The `ACM_EVSTAT.EV0` bit indicates when the ACM has completed the conversion for event 0. If set and the corresponding bit in `ACM_EVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 0 Conversion
		1	Event 0 Conversion Done

Event N Time Register

The `ACM_EVTIME[n]` registers each hold a 32-bit event time value. There are 16 event time registers, 8 for each ACM timer (when both timers are enabled). If only one timer is enabled, all 16 of the `ACM_EVTIME[n]` registers are assigned to the enabled timer.

Do not program the `ACM_EVTIME[n]` register when an event is active. Program this register before setting a trigger and re-program the register after all events are complete (`ACM_STAT.ECOM1` or `ACM_STAT.ECOM0` bits are set). If no events are enabled in this register (for example, the `ACM_EVCTL[n].ENAEV` bits =0 and the `ACM_STAT.ECOM0` or `ACM_STAT.ECOM1` bits =1) and an interrupt generates (if unmasked), a trigger is applied with the Timer enabled.

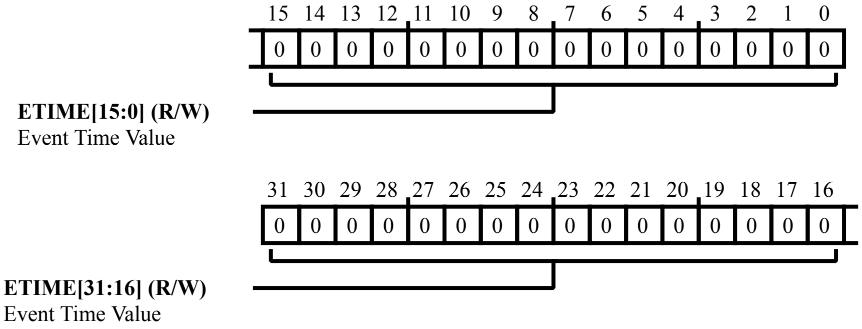

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ETIME[15:0] (R/W)
Event Time Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ETIME[31:16] (R/W)
Event Time Value

Figure 24-20: ACM_EVTIME[n] Register Diagram

Table 24-11: ACM_EVTIME[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ETIME	Event Time Value. The `ACM_EVTIME[n].ETIME` bit field contains a 32-bit event time value.

Missed Event Interrupt Mask Register

The ACM_MEVMSK register enables interrupts corresponding to status bits in the ACM_MEVSTAT register. When a bit is set in the ACM_MEVMSK register, an interrupt generates when the corresponding event missed bit in the ACM_MEVSTAT register is set.

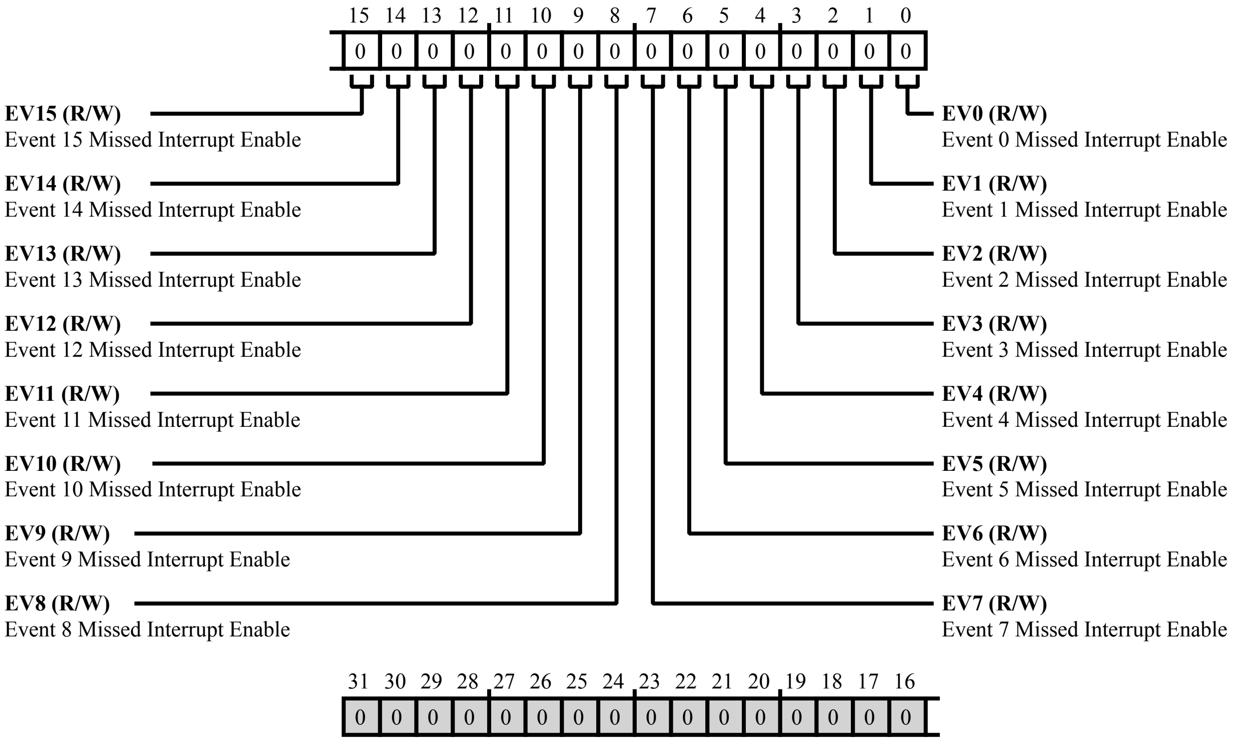

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EV15 (R/W)
Event 15 Missed Interrupt Enable
EV14 (R/W)
Event 14 Missed Interrupt Enable
EV13 (R/W)
Event 13 Missed Interrupt Enable
EV12 (R/W)
Event 12 Missed Interrupt Enable
EV11 (R/W)
Event 11 Missed Interrupt Enable
EV10 (R/W)
Event 10 Missed Interrupt Enable
EV9 (R/W)
Event 9 Missed Interrupt Enable
EV8 (R/W)
Event 8 Missed Interrupt Enable
EV0 (R/W)
Event 0 Missed Interrupt Enable
EV1 (R/W)
Event 1 Missed Interrupt Enable
EV2 (R/W)
Event 2 Missed Interrupt Enable
EV3 (R/W)
Event 3 Missed Interrupt Enable
EV4 (R/W)
Event 4 Missed Interrupt Enable
EV5 (R/W)
Event 5 Missed Interrupt Enable
EV6 (R/W)
Event 6 Missed Interrupt Enable
EV7 (R/W)
Event 7 Missed Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 24-21: ACM_MEVMSK Register Diagram

Table 24-12: ACM_MEVMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EV15	Event 15 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	EV14	Event 14 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	EV13	Event 13 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-12: ACM_MEVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	EV12	Event 12 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	EV11	Event 11 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	EV10	Event 10 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	EV9	Event 9 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	EV8	Event 8 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	EV7	Event 7 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	EV6	Event 6 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
5 (R/W)	EV5	Event 5 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	EV4	Event 4 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
3 (R/W)	EV3	Event 3 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-12: ACM_MEVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	EV2	Event 2 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	EV1	Event 1 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	EV0	Event 0 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Missed Event Status Register

The ACM_MEVSTAT register indicates which enabled event is missed for a particular trigger cycle. When a ACM_MEVSTAT register bit is set (=1), this status indicates a miss of the corresponding event. This status generates an interrupt if the corresponding bit in the ACM_MEVMSK register is set.

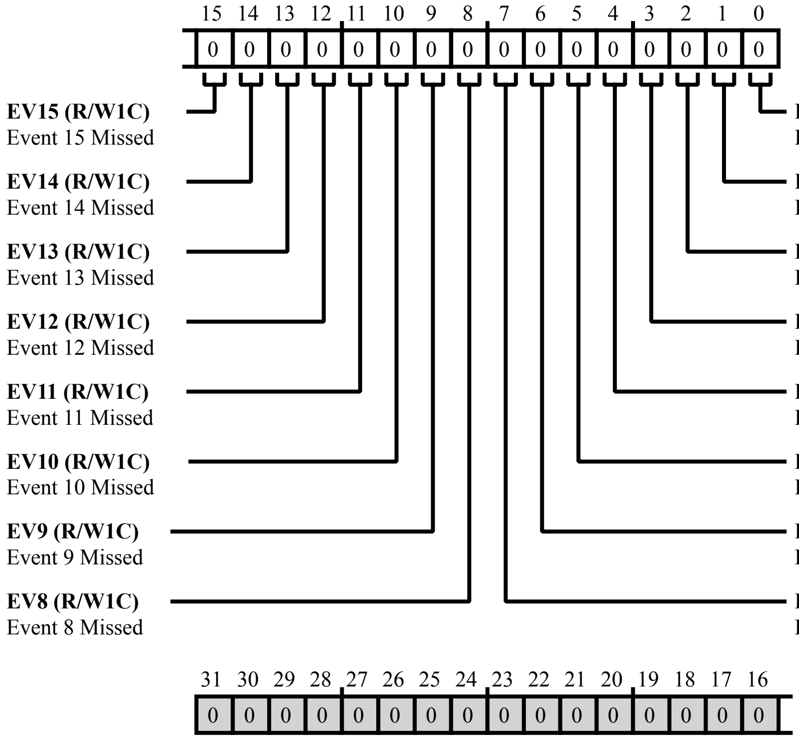

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EV15 (R/W1C)
Event 15 Missed
EV14 (R/W1C)
Event 14 Missed
EV13 (R/W1C)
Event 13 Missed
EV12 (R/W1C)
Event 12 Missed
EV11 (R/W1C)
Event 11 Missed
EV10 (R/W1C)
Event 10 Missed
EV9 (R/W1C)
Event 9 Missed
EV8 (R/W1C)
Event 8 Missed
EV0 (R/W1C)
Event 0 Missed
EV1 (R/W1C)
Event 1 Missed
EV2 (R/W1C)
Event 2 Missed
EV3 (R/W1C)
Event 3 Missed
EV4 (R/W1C)
Event 4 Missed
EV5 (R/W1C)
Event 5 Missed
EV6 (R/W1C)
Event 6 Missed
EV7 (R/W1C)
Event 7 Missed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 24-22: ACM_MEVSTAT Register Diagram

Table 24-13: ACM_MEVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	EV15	Event 15 Missed. The ACM_MEVSTAT.EV15 bit indicates a miss of event 15 since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 15 Missed Status
		1	Event 15 Missed

Table 24-13: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	EV14	Event 14 Missed. The `ACM_MEVSTAT.EV14` bit indicates a miss of event 14 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 14 Missed Status
		1	Event 14 Missed
13 (R/W1C)	EV13	Event 13 Missed. The `ACM_MEVSTAT.EV13` bit indicates a miss of event 13 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 13 Missed Status
		1	Event 13 Missed
12 (R/W1C)	EV12	Event 12 Missed. The `ACM_MEVSTAT.EV12` bit indicates a miss of event 12 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 12 Missed Status
		1	Event 12 Missed
11 (R/W1C)	EV11	Event 11 Missed. The `ACM_MEVSTAT.EV11` bit indicates a miss of event 11 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 11 Missed Status
		1	Event 11 Missed
10 (R/W1C)	EV10	Event 10 Missed. The `ACM_MEVSTAT.EV10` bit indicates a miss of event 10 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 10 Missed Status
		1	Event 10 Missed

Table 24-13: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	EV9	Event 9 Missed. The `ACM_MEVSTAT.EV9` bit indicates when a miss of event 9 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 9 Missed Status
		1	Event 9 Missed
8 (R/W1C)	EV8	Event 8 Missed. The `ACM_MEVSTAT.EV8` bit indicates a miss instance of event 8 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 8 Missed Status
		1	Event 8 Missed
7 (R/W1C)	EV7	Event 7 Missed. The `ACM_MEVSTAT.EV7` bit indicates a miss of event 7 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 7 Missed Status
		1	Event 7 Missed
6 (R/W1C)	EV6	Event 6 Missed. The `ACM_MEVSTAT.EV6` bit indicates a miss of event 6 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 6 Missed Status
		1	Event 6 Missed
5 (R/W1C)	EV5	Event 5 Missed. The `ACM_MEVSTAT.EV5` bit indicates a miss of event 5 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 5 Missed Status
		1	Event 5 Missed

Table 24-13: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	EV4	Event 4 Missed. The `ACM_MEVSTAT.EV4` bit indicates a miss of event 4 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 4 Missed Status
		1	Event 4 Missed
3 (R/W1C)	EV3	Event 3 Missed. The `ACM_MEVSTAT.EV3` bit indicates a miss of event 3 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 3 Missed Status
		1	Event 3 Missed
2 (R/W1C)	EV2	Event 2 Missed. The `ACM_MEVSTAT.EV2` bit indicates a miss of event 2 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 2 Missed Status
		1	Event 2 Missed
1 (R/W1C)	EV1	Event 1 Missed. The `ACM_MEVSTAT.EV1` bit indicates a miss of event 1 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 1 Missed Status
		1	Event 1 Missed
0 (R/W1C)	EV0	Event 0 Missed. The `ACM_MEVSTAT.EV0` bit indicates a miss of event 0 since the last trigger. If set and the corresponding bit in `ACM_MEVMSK` is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 0 Missed Status
		1	Event 0 Missed

Status Register

The `ACM_STAT` register indicates the ACM event that is currently being serviced, pending events, missed events, and missed triggers.

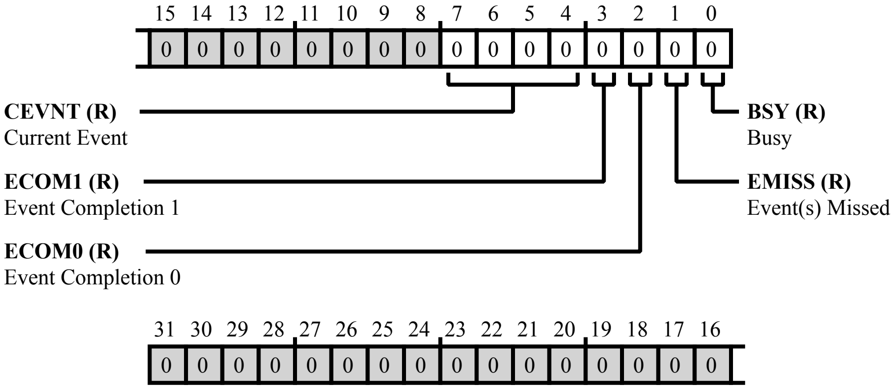

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CEVNT (R)
Current Event
ECOM1 (R)
Event Completion 1
ECOM0 (R)
Event Completion 0
BSY (R)
Busy
EMISS (R)
Event(s) Missed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 24-23: ACM_STAT Register Diagram

Table 24-14: ACM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/NW)	CEVNT	Current Event. The `ACM_STAT.CEVNT` bits indicate to which event (0 through 15) the ongoing access (current event, if any) corresponds.	
		0	Current Event Correspond to Event 0
		1	Current Event Correspond to Event 1
		15	Current Event Correspond to Event 15
3 (R/NW)	ECOM1	Event Completion 1. The `ACM_STAT.ECOM1` bit indicates TMR1 event completion for all enabled ACM TMR1 events and the current trigger. The ACM clears this bit with each trigger.	
		0	No Status
		1	ACM TMR1 Events Complete
2 (R/NW)	ECOM0	Event Completion 0. The `ACM_STAT.ECOM0` bit indicates TMR0 event completion for all enabled ACM TMR0 events and the current trigger. The ACM clears this bit with each trigger.	
		0	No Status
		1	ACM TMR0 Events Complete

Table 24-14: ACM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	EMISS	Event(s) Missed. The `ACM_STAT.EMISS` bit indicates a missed event time (any bits in `ACM_MEVSTAT` set). This bit is cleared by writing into the `ACM_MEVSTAT` register.	
		0	No Missed Event(s)
		1	Missed Event(s)
0 (R/NW)	BSY	Busy. The `ACM_STAT.BSY` bit indicates when the ACM is busy (an external sampling event in progress; CS is active or about to go active).	
		0	Idle
		1	Busy

Timing Configuration 0 Register

The ACM_TC0 register determines the frequency of ACM_CLK (using the ACM_TC0.CKDIV field) and the setup cycles (using the ACM_TC0.SC field) for the ADC controls. Setup cycles are specified in terms of SCLK0_0.

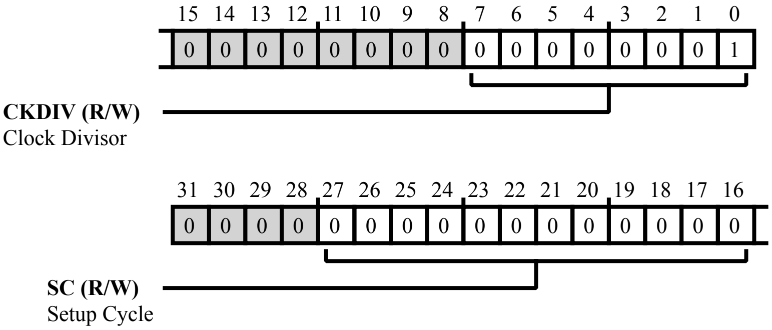

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
CKDIV (R/W)
Clock Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SC (R/W)
Setup Cycle

Figure 24-24: ACM_TC0 Register Diagram

Table 24-15: ACM_TC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27:16 (R/W)	SC	Setup Cycle. The ACM_TC0.SC bits select the ADC control pins and the setup time in SCLK0_0 cycles with respect to the ADC chip select active edge. The setup time may be calculated from: Setup Time = ACM_TC0.SC + 1. The maximum setup cycle time is 4096 * SCLK0_0, and the minimum setup cycle time is 1 SCLK0_0.	
		0	1 SCLK0_0 Cycle Setup Time
		1	2 SCLK0_0 Cycles Setup Time
		4095	4096 SCLK0_0 Cycles Setup Time
7:0 (R/W)	CKDIV	Clock Divisor. The ACM_TC0.CKDIV bits select the frequency of ACM_CLK as a function of the system clock frequency (SCLK0_0) and the value of the CKDIV field according to the formula: ACM_CLK frequency = (SCLK0_0 frequency)/(ACM_TC0.CKDIV + 1) The maximum ACM_CLK frequency is SCLK0_0/2, and the minimum ACM_CLK frequency is SCLK0_0/256. For example, for a 100 MHz SCLK0_0, the ACM_CLK frequency range is from 390 KHz to 50 MHz. The value ACM_TC0.CKDIV =0 is reserved.	

Timing Configuration 1 Register

The ACM_TC1 register provides programmability for the active duration of the following ADC controls: chip select (T_{CSW}), Hold Cycles (T_H), and Zero Cycles (T_Z).

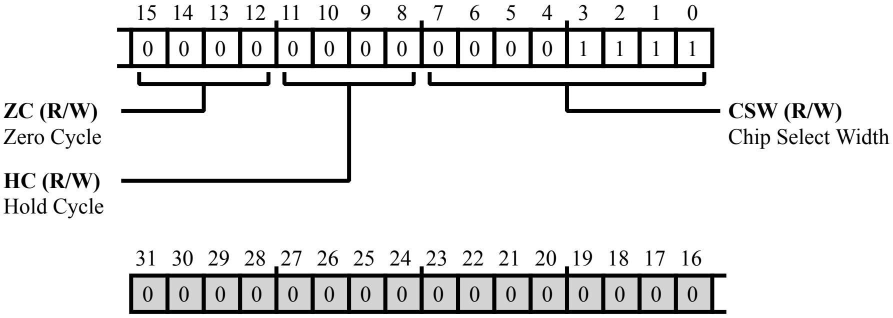

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
ZC (R/W)
Zero Cycle
HC (R/W)
Hold Cycle
CSW (R/W)
Chip Select Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 24-25: ACM_TC1 Register Diagram

Table 24-16: ACM_TC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	ZC	Zero Cycle. The ACM_TC1.ZC bits select the ADC control zero duration. All ADC controls drive low for ACM_TC1.ZC ACM_CLK cycles.	
		0	0 Zero Cycles
		1	1 Zero Cycle
		15	15 Zero Cycles
11:8 (R/W)	HC	Hold Cycle. The ACM_TC1.HC bits select the ADC control hold duration. All ADC controls are held after the inactive edge of CS for ACM_TC1.HC ACM_CLK cycles.	
		0	0 Hold Cycles
		1	1 Hold Cycle
		15	15 Hold Cycles
7:0 (R/W)	CSW	Chip Select Width. The ACM_TC1.CSW bits select the active duration of CS. The CS is active for ACM_TC1.CSW ACM_CLK +1 cycles.	
		0	1 Active CS Cycle
		1	2 Active CS Cycles
		15	16 Active CS Cycles
		255	256 Active CS Cycles

Timer 0 Register

The `ACM_TMR0` register contains the active count value for ACM timer 0. Access this read-only value at any time.

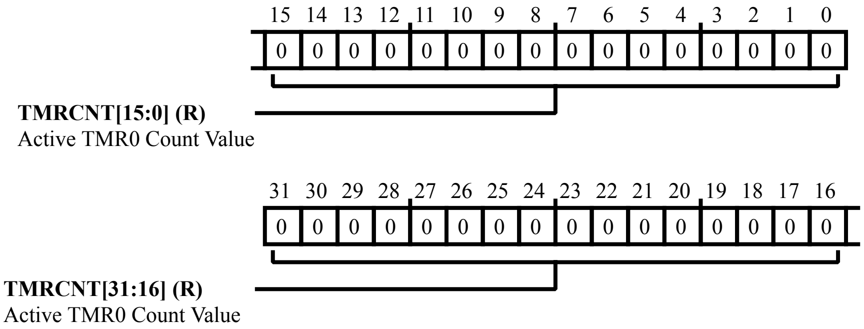

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRCNT[15:0] (R)
Active TMR0 Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRCNT[31:16] (R)
Active TMR0 Count Value

Figure 24-26: ACM_TMR0 Register Diagram

Table 24-17: ACM_TMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TMRCNT	Active TMR0 Count Value. The `ACM_TMR0.TMRCNT` bit field contains the active count value for ACM timer 0.

Timer 1 Register

The `ACM_TMR1` register contains the active count value for ACM timer 1. Programs can access this read-only value at any time.

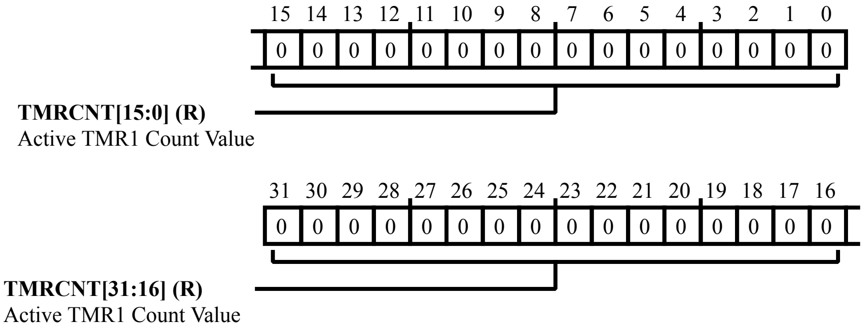

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRCNT[15:0] (R)
Active TMR1 Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRCNT[31:16] (R)
Active TMR1 Count Value

Figure 24-27: ACM_TMR1 Register Diagram

Table 24-18: ACM_TMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TMRCNT	Active TMR1 Count Value. The `ACM_TMR1.TMRCNT` bit field contains the active count value for ACM timer 1.

25 Controller Area Network (CAN)

The processor contains a Controller Area Network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is compatible with the control applications. It can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

NOTE: This document assumes familiarity with the CAN standard. For more information, refer to Version 2.0 of the CAN specification from Robert Bosch GmbH.

CAN Features

Key features of the CAN module include:

- Conformity to the CAN 2.0B (active) standard
- Dedicated acceptance mask for each mailbox
- Support for data rates of up to 1M bit/s
- Support for standard (11-bit) and extended (29-bit) identifiers
- 32 mailboxes (8 transmit, 8 receive, 16 configurable)
- Data filtering (first 2 bytes) for acceptance filtering (DeviceNet^{TM} mode)
- Error status and warning registers
- Universal counter-module
- Readable receive and transmit pin values
- Support for remote frames
- Active or passive network support
- Interrupts, including transmit or receive complete, error, and global
- Clock derived from CDU0_CLKO4 through a programmable divider, eliminating the need for an extra crystal

CAN Functional Description

The following sections provide information on the functional operation of the CAN module. This section also provides listings of the CAN registers and interrupts.

ADSP-SC58x CAN Register List

The controller area network (CAN) module implements the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. A set of registers govern CAN operations. For more information on CAN functionality, see the CAN register descriptions.

Table 25-1: ADSP-SC58x CAN Register List

Name	Description
CAN_AA1	Abort Acknowledge 1 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register

Table 25-1: ADSP-SC58x CAN Register List (Continued)

Name	Description
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Time Stamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_MD2	Mailbox Direction 2 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_STAT	Status Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_TIMING	Timing Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register

ADSP-SC58x CAN Interrupt List

Table 25-2: ADSP-SC58x CAN Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
84	CAN0_RX	CAN0 Receive	Level	
85	CAN0_TX	CAN0 Transmit	Level	
86	CAN0_STAT	CAN0 Status	Level	
87	CAN1_RX	CAN1 Receive	Level	
88	CAN1_TX	CAN1 Transmit	Level	
89	CAN1_STAT	CAN1 Status	Level	

External Interface

The interface to the CAN bus is a simple two-wire line. The ***Representation of CAN Transceiver Interconnection*** shows a symbolic representation of the CAN transceiver interconnection. Typically, the `CAN_TX` output and `CAN_RX` input pins of the processor connect to an external CAN `CAN_TX` and `CAN_RX` pins (respectively) of the tranceiver. The `CAN_TX` and `CAN_RX` pins operate with TTL levels and are appropriate for operation with CAN bus transceivers according to ISO/DIS 11898.

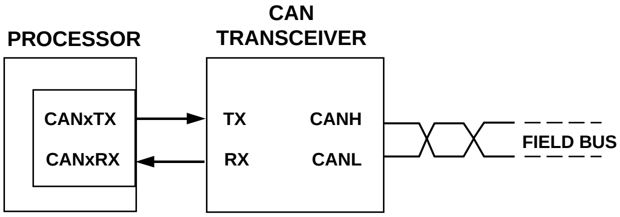


Figure 25-1: Representation of CAN Transceiver Interconnection

CAN data is either dominant (logic 0) or recessive (logic 1). The default state of the `CAN_TX` output is recessive.

Architectural Concepts

The full CAN controller features 32 message buffers called mailboxes. Eight mailboxes are dedicated for message transmission, eight are for reception, and 16 are programmable in direction.

The CAN module architecture is based around a 32-entry mailbox RAM. The CAN serial interface or the processor core accesses the mailbox sequentially. Each mailbox consists of eight 16-bit control and data registers and two optional 16-bit acceptance mask registers. Configure all of these registers before enabling the mailbox.

Since the mailbox area is implemented as RAM, the reset values of these registers are undefined. The ***CAN Mailbox Area*** figure shows the mailbox area. The data is divided into fields, which include a message identifier, a time stamp, a byte count, up to 8 bytes of data, and several control bits.

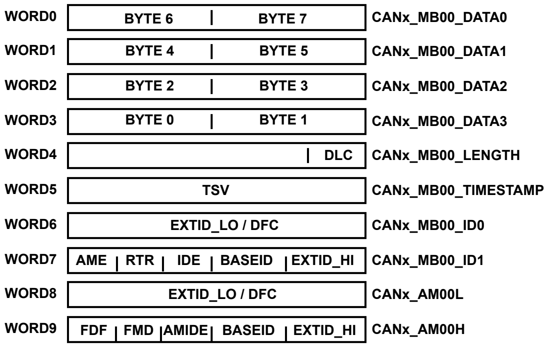

WORD0
BYTE 6
BYTE 7
CANx_MB00_DATA0
WORD1
BYTE 4
BYTE 5
CANx_MB00_DATA1
WORD2
BYTE 2
BYTE 3
CANx_MB00_DATA2
WORD3
BYTE 0
BYTE 1
CANx_MB00_DATA3
WORD4
DLC
CANx_MB00_LENGTH
WORD5
TSV
CANx_MB00_TIMESTAMP
WORD6
EXTID_LO / DFC
CANx_MB00_ID0
WORD7
AME
RTR
IDE
BASEID
EXTID_HI
CANx_MB00_ID1
WORD8
EXTID_LO / DFC
CANx_AM00L
WORD9
FDF
FMD
AMIDE
BASEID
EXTID_HI
CANx_AM00H

Figure 25-2: CAN Mailbox Area

The CAN mailbox identification register pair (CAN_MB[nn]_ID0/1) includes:

- The 29-bit identifier (base part CAN_AM[nn]H.BASEID plus the extended part CAN_AM[nn]L.EXTID/ CAN_AM[nn]H.EXTID)
- The acceptance mask enable bit (CAN_MB[nn]_ID1.AME)
- The remote transmission request bit (CAN_MB[nn]_ID1.RTR)
- The identifier extension bit (CAN_MB[nn]_ID1.IDE)

NOTE: Do not write to the identifier of a message object while the mailbox is enabled for the CAN module (the corresponding bit in CAN_MC1 is set).

The other mailbox area registers and bits are:

- The data length code bit (CAN_MB[nn]_LENGTH.DLC). The upper 12 bits of this register for each mailbox are marked as reserved. Always, set these 12 bits to zero.
- The mailbox word registers (CAN_MB[nn]_DATA0/1/2/3) supply up to 8 bytes for the data field. The data is sent MSB first based on the number of bytes defined in the CAN_MB[nn]_LENGTH.DLC bit. For example, if only one byte is transmitted or received (CAN_MB[nn]_LENGTH.DLC=1), then it is stored in the most significant byte of the CAN_MB[nn]_DATA3 register.
- The time stamp value bits (CAN_MB[nn]_TIMESTAMP.TSV)

The final registers in the mailbox area are the acceptance mask registers (CAN_AM[nn]H and CAN_AM[nn]L). The acceptance mask is enabled when the CAN_MB[nn]_ID1.AME bit is set.

Setting the CAN_CTL.DNM and CAN_AM[nn]H.FDF bits enables the *filtering on data field* option. When enabled, the CAN_MB[nn]_ID0.EXTID[15:0] bits are reused as acceptance code (DFC) for the data field filtering.

Block Diagram

The ***CAN Controller Block Diagram*** figure shows a block diagram of the CAN module.

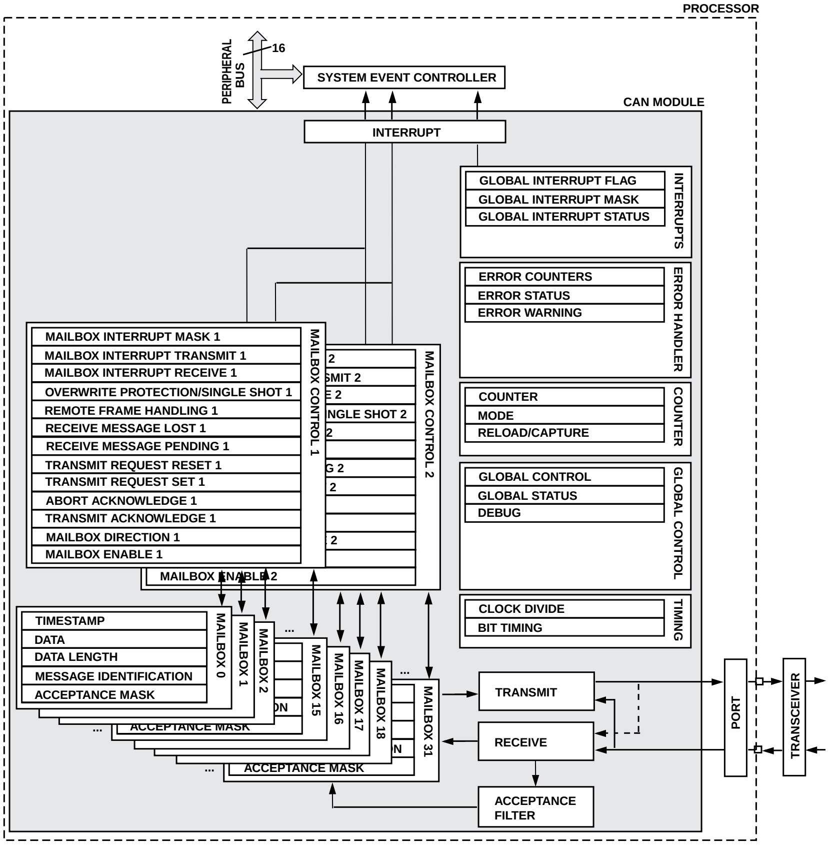
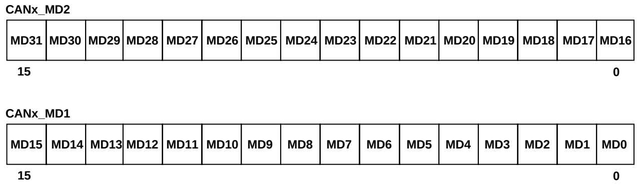

PROCESSOR
PERIPHERAL BUS
16
SYSTEM EVENT CONTROLLER
CAN MODULE
INTERRUPT
GLOBAL INTERRUPT FLAG
GLOBAL INTERRUPT MASK
GLOBAL INTERRUPT STATUS
INTERRUPTS
ERROR COUNTERS
ERROR STATUS
ERROR WARNING
ERROR HANDLER
COUNTER
MODE
RELOAD/CAPTURE
COUNTER
GLOBAL CONTROL
GLOBAL STATUS
DEBUG
GLOBAL CONTROL
CLOCK DIVIDE
BIT TIMING
TIMING
MAILBOX INTERRUPT MASK 1
MAILBOX INTERRUPT TRANSMIT 1
MAILBOX INTERRUPT RECEIVE 1
OVERWRITE PROTECTION/SINGLE SHOT 1
REMOTE FRAME HANDLING 1
RECEIVE MESSAGE LOST 1
RECEIVE MESSAGE PENDING 1
TRANSMIT REQUEST RESET 1
TRANSMIT REQUEST SET 1
ABORT ACKNOWLEDGE 1
TRANSMIT ACKNOWLEDGE 1
MAILBOX DIRECTION 1
MAILBOX ENABLE 1
MAILBOX CONTROL 1
MAILBOX ENABLE 2
MAILBOX CONTROL 2
TIMESTAMP
DATA
DATA LENGTH
MESSAGE IDENTIFICATION
ACCEPTANCE MASK
MAILBOX 0
MAILBOX 1
MAILBOX 2
MAILBOX 15
MAILBOX 16
MAILBOX 17
MAILBOX 18
MAILBOX 31
...
TRANSMIT
RECEIVE
ACCEPTANCE FILTER
PORT
TRANSCEIVER

Figure 25-3: CAN Controller Block Diagram

Mailbox Control

Mailbox control memory-mapped registers (MMRs) function as control and status registers for the 32 mailboxes. Each bit in these registers represents one specific mailbox. Since CAN MMRs are all 16 bits wide, pairs of registers manage certain functionality for all 32 individual mailboxes. Mailboxes 0–15 are configured or monitored in registers with a suffix of 1. Similarly, mailboxes 16–31 use the same named register with a suffix of 2. For example, the CAN mailbox direction registers (`CAN_MD1` / `CAN_MD2`) control mailboxes. See the ***CAN Mailbox Register Pair*** figure. The ***CAN Register List*** table shows the mailbox control registers.


CANx_MD2

15															0
MD31	MD30	MD29	MD28	MD27	MD26	MD25	MD24	MD23	MD22	MD21	MD20	MD19	MD18	MD17	MD16

CANx_MD1

15															0
MD15	MD14	MD13	MD12	MD11	MD10	MD9	MD8	MD7	MD6	MD5	MD4	MD3	MD2	MD1	MD0

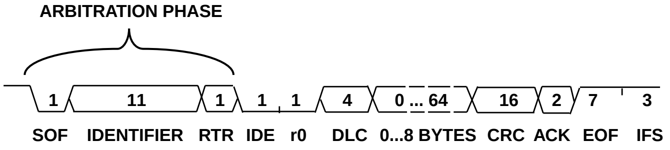
Figure 25-4: CAN Mailbox Register Pair

Mailboxes 24–31 support transmit operation only and mailboxes 0–7 are receive-only mailboxes. Therefore, the lower 8 bits in the 1 registers and the upper 8 bits in the 2 registers are sometimes reserved or are restricted in their use.

Protocol Fundamentals

Although the `CAN_RX` and `CAN_TX` pins are TTL-compliant signals, the CAN signals beyond the transceiver have asymmetric drivers. A low state on the `CAN_TX` pin activates strong drivers while a high state activates weak drivers. So, the active low state is the *dominant* state and the active high state is the *recessive* state. If the CAN module is passive, the `CAN_TX` pin is always high. If two CAN nodes transmit at the same time, dominant bits overwrite recessive bits.

The CAN protocol specifies that all nodes trying to send a message on the CAN bus attempt to send a frame once the bus is available. The ***Standard CAN Frame*** figure shows the frame. The Start of Frame (SOF) indicator signals the beginning of a new frame. Each CAN node then begins transmitting its message starting with the message ID.



SOF - START OF FRAME (SINGLE BIT = 0)
RTR - REMOTE TRANSMISSION REQUEST (REMOTE FRAME = 1)
IDE - IDENTIFIER EXTENSION (EXTENDED ID FRAME = 1)
r0 - RESERVED FOR FUTURE EXPANSION
DLC - DATA LENGTH CONTROL (NUMBER OF DATA BYTES IN FRAME)
CRC - CYCLIC REDUNDANCY CHECK (ERROR BITS IN FRAME)
ACK - ACKNOWLEDGE (RECEIVER DRIVES ONE DOMINANT BIT TO ACK)
EOF - END OF FRAME (SERIES OF 7 RECESSIVE BITS = b#1111111)
IFS - INTERFRAME SPACE (3 RECESSIVE BITS = b#111)

Figure 25-5: Standard CAN Frame

While transmitting, the CAN controller samples the `CAN_RX` pin to verify that the driven logic level is the value it placed on the `CAN_TX` pin. The names for the logic levels apply here. When a transmitting node places a recessive 1 on the `CAN_TX` pin and detects a dominant 0 on the `CAN_RX` pin, another node has placed a dominant bit on the bus. In this case, the dominant bit from from the other node has a higher priority.

Therefore, if the value sensed on the `CAN_RX` pin is the value driven on the `CAN_TX` pin, transmission continues. Otherwise, the CAN controller senses that it has lost arbitration. Module configuration determines the next course of action.

The ***Standard CAN Frame*** figure shows a basic 11-bit identifier frame. The `CAN_MB[nn]_ID1.RTR` bit follows the SOF and identifier. The `CAN_MB[nn]_ID1.RTR` bit indicates whether the frame contains data (data frame) or is a request for data associated with the message identifier in the frame sent (remote frame).

> **NOTE:** In the CAN protocol, a dominant bit in the `CAN_MB[nn]_ID1.RTR` field wins arbitration against a remote frame request (`CAN_MB[nn]_ID1.RTR`=1) for the same message ID. This functionality allows a remote request to be a lower priority than a data frame.

The next field of interest in the frame is the `CAN_MB[nn]_ID1.IDE` bit. When set, it indicates that the message is an extended frame with a 29-bit identifier instead of an 11-bit identifier. In an extended frame, the first part of the message resembles the ***Extended CAN Frame*** figure.

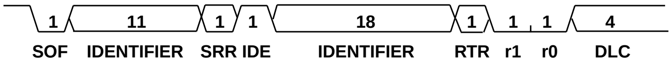

1
11
1
1
18
1
1
1
4
SOF
IDENTIFIER
SRR
IDE
IDENTIFIER
RTR
r1
r0
DLC

Figure 25-6: Extended CAN Frame

For the `CAN_MB[nn]_ID1.RTR` field, a dominant bit in the `CAN_MB[nn]_ID1.IDE` field wins arbitration against an extended frame with the same lower 11 bits. Standard frames have a higher priority than extended frames.

The internal logic automatically generates the Substitute Remote Request (SRR), the reserved bits r0 and r1, and the checksum (CRC). (The SRR is always sent as recessive; reserved bits r0 and r1 are always sent as dominant).

CAN Operating Modes

The CAN controller is in configuration mode when coming out of processor reset. Hardware behavior can be altered only when CAN is in configuration mode. Before initializing the mailboxes, configure the CAN bit timing to work on the CAN bus. The controller connect to the CAN bus.

Data Transfer Modes

The following sections provide information on the data transfer modes supported by the CAN controller.

Transmit Operations

The ***CAN Transmit Operation Flowchart*** shows the CAN transmit operation. Mailboxes 24–31 are dedicated transmitters. Configure mailboxes 8–23 as transmitters by writing 0 to the corresponding bit in the `CAN_MD1` or `CAN_MD2` registers. Enable mailbox n (`CAN_MC1.MB`=1). After writing the data and the identifier into the mailbox area, the message is sent. Then, the corresponding transmit request bit is set (`CAN_TRS1.MB`=1).

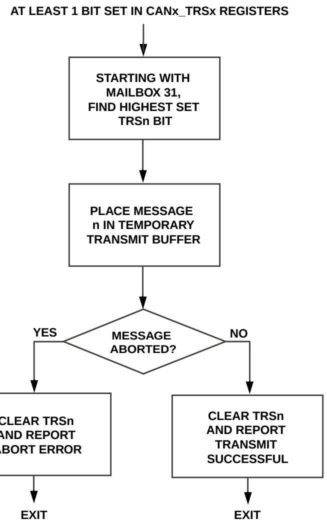

AT LEAST 1 BIT SET IN CANx_TRSx REGISTERS
STARTING WITH MAILBOX 31, FIND HIGHEST SET TRSn BIT
PLACE MESSAGE n IN TEMPORARY TRANSMIT BUFFER
MESSAGE ABORTED?
YES
NO
CLEAR TRSn AND REPORT ABORT ERROR
CLEAR TRSn AND REPORT TRANSMIT SUCCESSFUL
EXIT
EXIT

Figure 25-7: CAN Transmit Operation Flowchart

When a transmission completes, the corresponding bits in the `CAN_TRS1` or `CAN_TRS2` and `CAN_TRR1` or `CAN_TRR2` registers are cleared. If the transmission is successful, the corresponding bit in the `CAN_TA1`/ `CAN_TA2` register is set. If the transmission aborts due to lost arbitration or a CAN error, the corresponding bit in the CAN_AA1/CAN_AA2 register is set. A requested transmission can also be manually aborted by setting the corresponding bit in the CAN_TRR1/CAN_TRR2 register.

Software sets multiple `CAN_TRS1.MB` bits simultaneously. These bits are reset after either a successful or an aborted transmission.

The CAN hardware sets these bits in the following cases:

- When using the auto-transmit mode of the universal counter
- When a message loses arbitration and the single-shot `CAN_OPSS1.MB` bit is not set
- When a remote frame request occurs (only possible for receive or transmit mailboxes if the feature for automatic remote frame handling is enabled (`CAN_RFH1.MB`=1)).

NOTE: Manage the mailbox area when a CAN_TRS1 or CAN_TRS2 bit is set. Write access to the mailbox is permissible with a bit set. But, changing data in such a mailbox can lead to unexpected data during transmission.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 or CAN_TRS2 bit associated with a disabled mailbox can result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 or CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

Retransmission

Normally, the current message object is resent after the loss of arbitration or error frame detection on the CAN bus line. If there is more than one transmit message object pending, the message object with the highest mailbox transmits first. See the ***Transmit Flow*** figure. The currently aborted transmission restarts after any messages with higher priority are sent.

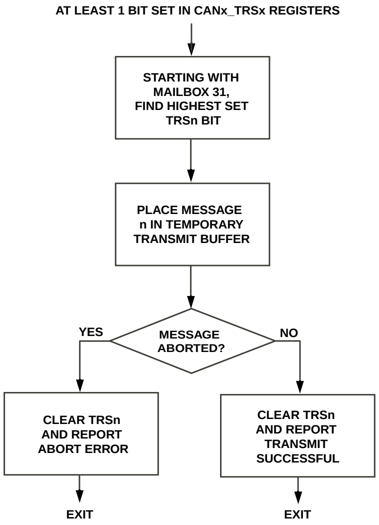

AT LEAST 1 BIT SET IN CANx_TRSx REGISTERS
STARTING WITH MAILBOX 31, FIND HIGHEST SET TRSn BIT
PLACE MESSAGE n IN TEMPORARY TRANSMIT BUFFER
MESSAGE ABORTED?
YES
NO
CLEAR TRSn AND REPORT ABORT ERROR
CLEAR TRSn AND REPORT TRANSMIT SUCCESSFUL
EXIT
EXIT

Figure 25-8: Transmit Flow

A message written into the mailbox does not replace a message under preparation. The message under preparation is copied into the temporary transmit buffer when the internal transmit request for the CAN core module is set. The message in the buffer is not replaced until:

- The message is sent successfully
- The arbitration on the CAN bus line is lost
- There is an error frame on the CAN bus line

Single-Shot Transmission

When using the single-shot transmission feature (`CAN_OPSS1.MB`=1), the corresponding `CAN_TRS1` bit is cleared after the message is successfully sent. The bit is cleared even if the transmission aborts due to a lost arbitration or an error frame on the CAN bus line. Therefore, there is no further attempt to transmit the message again when the initial try failed, and the abort error is reported (`CAN_AA1.MB`=1).

Auto-Transmission

In auto-transmit mode, the message in mailbox 11 (MB11) can be sent periodically using the universal counter. This mode often broadcasts heartbeats to all CAN nodes. So, messages sent this way usually have a high priority.

The period value is written to the `CAN_UCRC` register. Auto-transmission mode is enabled by setting the `CAN_UCCNF.UCCNF` field to 0x03. When enabled, the counter `CAN_UCCNT` is loaded with the value in the `CAN_UCRC` register. The counter decrements to 0 at the CAN bit clock rate and is then reloaded from `CAN_UCRC`. Each time the counter reaches a value of 0, internal logic automatically sends the `CAN_TRS1.MB` bit. The corresponding message from mailbox 11 transfers.

For proper auto-transmit operation, configure mailbox 11 as a transmit mailbox. The mailbox must contain valid data (identifier, control bits, and data) before the counter expires and after this mode is enabled.

Receive Operation

The CAN hardware autonomously receives messages and discards invalid messages. Once a valid message is successfully received, the receive logic interrogates all enabled receive mailboxes. The logic interrogates sequentially, from mailbox 23 down to mailbox 0, whether the message is of interest to the local node or not.

Each incoming data frame is compared to all identifiers stored in the active receive and transmit mailboxes with the feature for remote frame handling enabled (=1). The active receive mailboxes indices of `CAN_MD1` and `CAN_MC1` registers are set to 1. The message identifier of the received message, along with the identifier extension (`CAN_MB[nn]_ID1.IDE`) and remote transmission request (`CAN_MB[nn]_ID1.RTR`) bits, are compared with the register settings of each mailbox. In standard mode, the message is compared with the content of the `CAN_MB[nn]_ID1` register. In extended mode, the content of the `CAN_MB[nn]_ID0` register must also match.

If the acceptance mask enable `CAN_MB[nn]_ID1.AME` bit is not set, a match is signaled only if `CAN_MB[nn]_ID1.IDE`, `CAN_MB[nn]_ID1.RTR`, and all (11 or 29) identifier bits are exact. If, however, the `CAN_MB[nn]_ID1.AME` bit is set, the acceptance mask registers (`CAN_AM[nn]H`/L) determine which of the `CAN_MB[nn]_ID1.IDE` and `CAN_MB[nn]_ID1.RTR` bits must match.

The following logic applies:

[(Received Message ID) XNOR (`CAN_MB[nn]_ID0`/1)] OR [(`CAN_MB[nn]_ID1.AME`) AND (`CAN_AM[nn]H` /L)].

This logic appears graphically in the *CAN Message Receive Logic* figure.

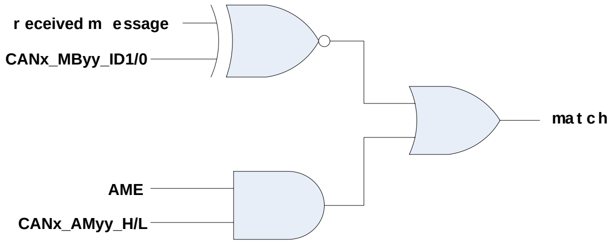


Figure 25-9: CAN Message Receive Logic

A one (1) at the respective bit position in the `CAN_AM[nn]H`/`CAN_AM[nn]L` mask registers means that the bit does not need to match when `CAN_MB[nn]_ID1.AME`=1. This way, a mailbox can accept a group of messages.

Table 25-3: Mailbox Used for Acceptance Filtering

MCn	MDn	RFHn	Mailbox n	Comment
0	X	X	Ignored	Mailbox n disabled
1	0	0	Ignored	Mailbox n enabled; Mailbox n configured for transmit; Remote frame handling disabled
1	0	1	Used	Mailbox n enabled; Mailbox n configured for transmit; Remote frame handling enabled
1	1	X	Used	Mailbox n enabled; Mailbox n configured for receive

If the acceptance filter finds a matching identifier, the content of the received data frame is stored in that mailbox. A received message is stored only once, even if multiple receive mailboxes match its identifier. If the current identifier does not match any mailbox, the message is not stored.

The *CAN Receive Operation Flowchart* illustrates the decision tree of the receive logic when processing the individual mailboxes.

If a message is received for a mailbox and that mailbox still contains unread data (`CAN_RMP1.MB`), then the program decides whether to overwrite the old message. If the `CAN_OPSS1.MB` bit is cleared, the corresponding `CAN_RML1.MB` bit is set, and the stored message is overwritten. The receive message lost interrupt request occurs (`CAN_GIS.RMLIS` is set). If, however, the `CAN_OPSS1.MB` bit is set, the next mailboxes are checked for another matching identifier. If no match is found, the message is discarded, and the next message is checked.

NOTE: If a receive mailbox is disabled, an ongoing receive message for that mailbox is lost even if a second mailbox is configured to receive the same identifier.

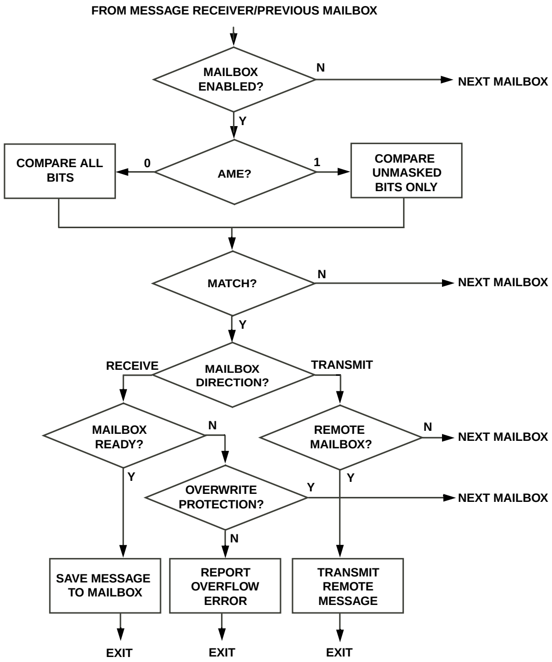

FROM MESSAGE RECEIVER/PREVIOUS MAILBOX
MAILBOX ENABLED?
N
NEXT MAILBOX
Y
COMPARE ALL BITS
0
AME?
1
COMPARE UNMASKED BITS ONLY
MATCH?
N
NEXT MAILBOX
Y
RECEIVE
MAILBOX DIRECTION?
TRANSMIT
MAILBOX READY?
N
REMOTE MAILBOX?
N
NEXT MAILBOX
Y
OVERWRITE PROTECTION?
Y
Y
NEXT MAILBOX
N
SAVE MESSAGE TO MAILBOX
REPORT OVERFLOW ERROR
TRANSMIT REMOTE MESSAGE
EXIT
EXIT
EXIT

Figure 25-10: CAN Receive Operation Flowchart

Data Acceptance Filtering

If Device Net mode is enabled (`CAN_CTL.DNM` = 1) and the mailbox is set up for filtering on data field, the filtering occurs on the standard ID of the message and data fields. The data field filtering can be programmed for either the first byte only or the first 2 bytes, as shown the ***Data Field Filtering*** table.

If the `CAN_AM[nn]H.FDF` bit is set, the corresponding `CAN_AM[nn]L` register holds the data field mask (DFM bits 15:0]). If the `CAN_AM[nn]H.FDF` bit is cleared, the corresponding `CAN_AM[nn]L` register holds the extended identifier mask (`CAN_AM[nn]H.EXTID` bits 15:0).

Table 25-4: Data Field Filtering

FDF (Filter on Data Field)	FMD (Full Mask Data Field)	Description
0	0	Do not allow filtering on the data field
0	1	Not allowed. FMF must be 0 when FDF is 0
1	0	Filter on first data byte only
1	1	Filter on first two data bytes

Watchdog Mode

Watchdog mode ensures that messages are received periodically. It also observes whether a certain node on the network is alive and functioning properly. Watchdog mode detects and manages the failure cases, as needed.

Enable this mode by programming the universal counter to watchdog mode by setting the `CAN_UCCNF.UCCNF` to 0x2. Once enabled, the `CAN_UCCNT` register is loaded with the predefined value contained in `CAN_UCRC`. This counter decrements at the CAN bit rate.

If the `CAN_UCCNF.UCCT` and `CAN_UCCNF.UCRC` bits are set and a message is received in mailbox 4 before the counter counts down to 0, the counter is reloaded with the `CAN_UCRC` contents. If the counter has counted down to 0 without receiving a message in mailbox 4, then the `CAN_GIS.UCEIS` bit is set. The counter reloads automatically with the contents of the `CAN_UCRC` register. If an interrupt request is desired for this event, set the `CAN_GIM.UCEIM` bit. With the mask bit set, when a watchdog interrupt request occurs, the `CAN_GIF.UCEIF` bit is also set.

Write to the `CAN_UCCNF` register to reload the counter with the contents of `CAN_UCRC` or to disable the register.

The `CAN_UCRC` register controls the time period it takes for the watchdog interrupt request to occur.

Time Stamps

To get an indication of the time of the receive or transmit time for each message, program the CAN universal counter to time stamp mode. Enable this mode by setting the `CAN_UCCNF.UCCNF` field to 0x01.

If enabled, the value of the 16-bit free-running counter (`CAN_UCCNT`) is written into the `CAN_MB[nn]_TIMESTAMP` register of the corresponding mailbox. The operation occurs when a received message is stored or a message is transmitted.

The time stamp value is captured at the sample point of the Start of Frame (SOF) bit of each incoming or outgoing message. Afterwards, this time stamp value is copied to the `CAN_MB[nn]_TIMESTAMP` register of the corresponding mailbox.

If the mailbox is configured for automatic remote frame handling (`CAN_RFH1.MB` = 1), the time stamp value is written for transmission of a data frame or the reception of the requested data frame. The mailbox is configured for transmit or receive.

Clear the counter by setting the `CAN_UCCNF.UCRC` bit to 1. Or, disable the counter by clearing the `CAN_UCCNF.UCE` bit. Write to the `CAN_UCCNT` register to load the counter with a value.

It is also possible to clear the counter (`CAN_UCCNT`) by the reception of a message in mailbox number 4 (synchronization of all time stamp counters in the system). This operation is accomplished by setting the `CAN_UCCNF.UCCT` bit.

The `CAN_GIS.UCEIS` bit is set when the counter overflows. A global CAN interrupt request can optionally occur by unmasking the `CAN_GIM.UCEIM` bit. If the interrupt source is unmasked, the `CAN_GIF.UCEIF` bit is also set.

Remote Frame Handling

Automatic handling of remote frames for a transmit mailbox is enabled by setting the corresponding `CAN_RFH1.MB` bit.

Remote frames are data frames that have no data field and the `CAN_MB[nn]_ID1.RTR` bit is set. The data length code (DLC) of the requesting remote frame overrules the DLC of the responding data frame. A DLC can be programmed with values in the range of 0–15, but DLC values greater than 8 are considered as 8.

A remote frame contains:

- The identifier bits
- The control field `CAN_MB[nn]_LENGTH.DLC` (data length count)
- The remote transmission request (`CAN_MB[nn]_ID1.RTR`) bit

Only configurable mailboxes MB8–MB23 can process remote frames, but all mailboxes can receive and transmit remote frame requests. The `CAN_OPSS1` register has no effect when configured for automatic remote frame handling. All content of a mailbox is always overwritten by an incoming message.

NOTE: If a remote frame is received, the DLC of the corresponding mailbox is overwritten with the received value.

Erroneous behavior can result when the `CAN_RFH1.MB` bit is changed while the corresponding mailbox is processing. To avoid the risk of inconsistent messages, programs must temporarily disable the mailbox while its data registers are updating.

Temporarily Disabling CAN Mailbox

If a mailbox is enabled and configured to transmit, monitor the write accesses to the data field to avoid transmitting inconsistent messages. Be careful if the mailbox is transmitting (or attempting to transmit) repeatedly. Also, if this mailbox is used for automatic remote frame handling, the data field must be updated without losing an incoming remote request frame and without sending inconsistent data. Therefore, the CAN controller allows for temporarily disabling the mailbox using the mailbox temporary disable register (`CAN_MBTD`).

The pointer to the requested mailbox to the `CAN_MBTD.TDPTR` field is written, and the `CAN_MBTD.TDR` bit is set. Internal logic then sets the corresponding `CAN_MBTD.TDA` flag.

If a mailbox is configured as transmit (`CAN_MD1` = 0) and the `CAN_MBTD.TDA` bit is set, the content of the data field of that mailbox can be updated. If there is an incoming remote request frame while the mailbox is temporarily disabled,

- Internal logic sets the corresponding transmit request bit (`CAN_TRS1.MB`), and
- The data length code (DLC) of the incoming message is written to the corresponding mailbox.

However, the requested message is not sent until the `CAN_MBTD.TDR` bit is cleared. Similarly, all transmit requests for temporarily disabled mailboxes are ignored until the `CAN_MBTD.TDR` bit is cleared. Additionally, transmission of a message immediately aborts when the mailbox is temporarily disabled and the corresponding transmission request reset (`CAN_TRR1.MB`) bit for this mailbox is set.

If a mailbox is configured to receive (`CAN_MD1` = 1), then after issuing a temporary disable request, the `CAN_MBTD.TDA` flag is set. The mailbox is not processed. If there is an incoming message for a temporarily disabled mailbox, the internal logic waits until reception is complete or there is an error on the CAN bus before setting `CAN_MBTD.TDA`. Once this flag is set, the mailbox can then be disabled (`CAN_MC1` = 0) without the risk of losing an incoming frame. The `CAN_MBTD.TDR` bit must then be reset as soon as possible.

When the `CAN_MBTD.TDA` flag is set for a given mailbox, only the data field of that mailbox can be updated. Accesses to the control bits and the identifier are denied.

Bit Timing

The CAN controller does not have a dedicated clock. Instead, the CAN clock is derived from the system clock based on a configurable number of time quanta. The time quantum (TQ) is derived from the formula:

TQ = (BRP + 1)/CDU0_CLKO4

where BRP is the 10-bit bit rate prescaler field in the `CAN_CLK` register.

Although the `CAN_CLK.BRP` field can be set to any value, it is recommended that the value be greater than or equal to 4. Restrictions apply to the bit timing configuration when BRP is less than 4.

The `CAN_CLK` register defines the TQ value, and multiple time quanta make up the duration of a CAN bit on the bus. The `CAN_TIMING` register controls the nominal bit time and the sample point of the individual bits in the CAN protocol. The ***Three Phases of a CAN Bit*** figure shows the three phases of a CAN bit: the synchronization segment, the segment before the sample point, and the segment after the sample point.

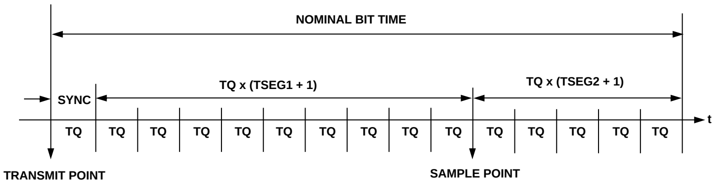


Figure 25-11: Three Phases of a CAN Bit

The synchronization segment is fixed to one TQ. Synchronize the nodes on the bus. All signal edges are expected to occur within this segment.

The `CAN_TIMING.TSEG1` and `CAN_TIMING.TSEG2` fields control how many TQs the CAN bits consist of, resulting in the CAN bit rate. The following formula gives the nominal bit time.

$t_{BIT} = TQ \times [1 + (1 + TSEG1) + (1 + TSEG2)]$

For safe receive operations on given physical networks, the sample point is programmable by the `CAN_TIMING.TSEG1` field. The `CAN_TIMING.TSEG2` field holds the number of TQs to complete the bit time. Often, best sample reliability is achieved with sample points in the high 80% range of the bit time. Never use sample points lower than 50%. Therefore, `CAN_TIMING.TSEG1` must always be greater than or equal to `CAN_TIMING.TSEG2`.

The CAN module does not distinguish between the propagation segment and the phase segment-1 as defined by the standard. The `CAN_TIMING.TSEG1` value is intended to cover both of them. The `CAN_TIMING.TSEG2` value represents the phase segment-2.

If the CAN module detects a recessive-to-dominant edge outside the synchronization segment, it can automatically move the sampling point such that the CAN bit is still handled properly. The synchronization jump width (`CAN_TIMING.SJW`) field specifies the maximum number of TQs, ranging from 1 to 4 (SJW + 1), allowed for such a resynchronization attempt. The SJW value must not exceed `CAN_TIMING.TSEG2` or `CAN_TIMING.TSEG1`. Therefore, the fundamental rule for writing `CAN_TIMING` is:

$SJW \leq TSEG2 \leq TSEG1$

In addition to this fundamental rule, `CAN_TIMING.TSEG2` must also be greater than or equal to the information processing time (IPT). IPT is the time required by the logic to sample the `CAN_RX` input, which is 3 system clock cycles.

Therefore, restrictions apply to the minimal value of `CAN_TIMING.TSEG2` if `CAN_CLK.BRP` is less than 2. If `CAN_CLK.BRP` is set to 0, the `CAN_TIMING.TSEG2` field must be greater than or equal to 2. If `CAN_CLK.BRP` is set to 1, the minimum `CAN_TIMING.TSEG2` value is 1.

NOTE: Use the same nominal bit rate for all nodes on a CAN bus.

With all the timing parameters set, the final consideration is sampling performance. The default behavior of the CAN controller is to sample the CAN bit once. The controller samples at the point described by the `CAN_TIMING` register and controlled by the `CAN_TIMING.SAM` bit. If this bit is set, however, the input signal is oversampled three times at the system clock rate. The resulting value is generated by a majority decision of the three sample values. Always keep the `CAN_TIMING.SAM` bit cleared if the BRP value is less than 4.

Do not modify the `CAN_CLK` and `CAN_TIMING` registers during normal operation. Always enter configuration mode first. Writes to these registers have no effect when CAN is not in configuration or debug mode. If not coming out of processor reset, enter configuration mode by setting the `CAN_CTL.CCR` bit and poll the `CAN_STAT` register until `CAN_STAT.CCA` is set.

NOTE: If the `CAN_TIMING.TSEG1` field is programmed to 0, the module does not leave the configuration mode.

During configuration mode, the module is not active on the CAN bus line. The `CAN_TX` output pin remains recessive and the module does not receive or transmit messages or error frames. After leaving the configuration mode, all CAN internal core registers and the CAN error counters are set to their initial values.

A soft reset does not change the values of `CAN_CLK` and `CAN_TIMING`. Therefore, an ongoing transfer through the CAN bus cannot be corrupted by changing the bit timing parameter or initiating the soft reset (by setting the `CAN_CTL.SRS` bit).

CAN Low Power Features

The CAN module includes built-in sleep and suspend modes to save power.

The following sections describe the behavior of the CAN module in these modes.

Built-In Suspend Mode

The most modest of power savings mode is the suspend mode. This mode is entered by setting the `CAN_CTL.CSR` bit. The module enters the suspend mode after the current operation of the CAN bus finishes. Then, the internal logic sets the `CAN_STAT.CSA` bit. Once CAN enters this mode, the module is no longer active on the CAN bus line, slightly reducing power consumption.

In suspend mode, the `CAN_TX` output pin remains in a recessive state, and the module does not receive or transmit messages or error frames. The content of the `CAN_CEC` register remains unchanged. Clear `CAN_CTL.CSR` to exit suspend mode.

The only difference between suspend mode and configuration mode is that the `CAN_CTL` and `CAN_STAT` registers are not reset when exiting suspend mode.

Built-In Sleep Mode

The next level of power savings can be realized by using the built-in sleep mode for the module. This mode is entered by setting the `CAN_CTL.SMR` bit. The module enters the sleep mode after the current operation of the CAN bus finishes. Once this mode is entered, many of the internal CAN module clocks are shut off, reducing power consumption, and the `CAN_INT.SMACK` bit is set.

When the CAN module is in sleep mode, all register reads return the contents of `CAN_INT` instead of the usual contents. All register writes, except to `CAN_INT`, are ignored in sleep mode. A small part of the module is clocked continuously to allow for waking up out of sleep mode.

A write to the `CAN_INT` register ends sleep mode. If the `CAN_CTL.WBA` bit is set before entering sleep mode, a dominant bit on the `CAN_RX` pin also ends sleep mode. When software sets the `CAN_CTL.SMR` bit, hardware sets the `CAN_CTL.CSR` bit as well, making sleep mode a super set of suspend mode. When the controller wakes up from sleep mode, hardware automatically clears `CAN_CTL.SMR` and `CAN_CTL.CSR`. If, however, the controller never enters sleep mode because the wake-up condition was met before `CAN_INT.SMACK` bit turns to 1, the `CAN_CTL.SMR` and `CAN_CTL.CSR` bits do not always automatically clear. Therefore, clear the two bits using software, when returning from sleep mode.

Soft Reset

The CAN controller features a build-in reset mechanism called soft reset. Soft reset is entered immediately after software has set the `CAN_CTL.SRS` bit. Soft reset brings all control registers to a defined state. Mailbox and error registers remain unaffected. Soft reset does not alter the `CAN_TIMING` and `CAN_CLK` registers and does not disturb the on-going transmission of a currently pending message, acknowledge bit or error frame. However, when recovering from soft reset, software can lose track of transmission or reception reports and interrupt requests.

CAN Event Control

The following is a description of how CAN generates and controls events.

CAN Interrupt Signals

The CAN module provides three independent interrupt requests: two mailbox interrupt requests (mailbox receive interrupt request (MBRIRQ) and mailbox transmit interrupt request (MBTIRQ)) and a global CAN status interrupt request (GIRQ). The values of these three interrupt requests can also be read through the `CAN_GIS` register.

Mailbox Interrupts

Each of the 32 mailboxes in the CAN module can generate a receive or transmit interrupt request, depending on the mailbox configuration. To enable a mailbox to generate an interrupt request, set the corresponding `CAN_MBIM1` bit.

If a mailbox is configured as a receive mailbox, the corresponding `CAN_MBRIF1` bit and `CAN_RMP1` bit are set after a received message is stored in mailbox n. When using the feature for automatic remote frame handling (`CAN_RFH1`=1), the receive interrupt flag is set after the requested data frame is stored in the mailbox.

If any `CAN_MBRIF1` bits are set, the mailbox generates a `CAN_INT.MBRIRQ` interrupt request. To clear the `CAN_INT.MBRIRQ` interrupt request, software must clear all of the set `CAN_MBRIF1` bits by writing a 1 to those bit locations in `CAN_MBRIF1`. Prior to this operation, software must clear the corresponding `CAN_RMP1` bit.

If a mailbox is configured as a transmit mailbox, the corresponding `CAN_MBTIF1` bit in the transmit interrupt flag is set after the message in mailbox n is sent correctly. The corresponding `CAN_TA1` bit is also set. The `CAN_TA1` bits maintain their state even after the corresponding mailbox n is disabled (`CAN_MC1`=0). When using the feature for automatic remote frame handling, the transmit interrupt flag is set after the requested data frame is sent from the mailbox.

If any `CAN_MBTIF1.MB` bits are set, the MBTIRQ interrupt output is raised in the `CAN_INT` register. To clear the MBTIRQ interrupt request, software must clear all of the bits that are set in the `CAN_MBTIF1` register by writing a 1 to those bit locations. Additionally, software must clear the associated `CAN_TA1` bit or set the associated `CAN_TRS1` bit to clear the interrupt source that asserts the `CAN_MBTIF1` bit.

Global Interrupt

The global CAN interrupt logic implements with three registers:

- The `CAN_GIM` register, where each interrupt source can be enabled or disabled separately
- The `CAN_GIS` register
- The `CAN_GIF` register

The interrupt mask bits only affect the content of the `CAN_GIF` register. If the mask bit is not set in the `CAN_GIM` register, the corresponding flag bit is not set when the event occurs. The interrupt status bits in the `CAN_GIS` register, however, are always set when the corresponding interrupt event occurs, independent of the mask bits. Thus, the interrupt status bits can be used to poll interrupt events.

The `CAN_INT.GIRQ` bit is only asserted if a bit in the `CAN_GIF` register is set. The read-only `CAN_INT.GIRQ` bit remains set as long as at least 1 bit in `CAN_GIF` is set. All bits in the interrupt status and interrupt flag registers remain set until cleared by software or a soft reset has occurred.

NOTE: The `CAN_GIF` register is read-only (RO). In the global CAN interrupt ISR, clear the interrupt latch by writing a 1 to the corresponding bit of the `CAN_GIS` register. The operations clear the related bits of the `CAN_GIS` and `CAN_GIF` registers, as well as the `CAN_INT.GIRQ` bit.

There are several interrupt events that can activate this GIRQ interrupt request:

- Access denied event interrupt (`CAN_GIM.ADIM`, `CAN_GIS.ADIS`, `CAN_GIF.ADIF`): At least one access to the mailbox RAM occurred during a data update by internal logic.

- Universal counter exceeded event interrupt (`CAN_GIM.UCEIM`, `CAN_GIS.UCEIS`, `CAN_GIF.UCEIF`): There is an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).
- Receive message lost event interrupt (`CAN_GIM.RMLIM`, `CAN_GIS.RMLIS`, `CAN_GIF.RMLIF`): A message is received for a mailbox that currently contains unread data. At least 1 bit in the `CAN_RMLn` register is set. If the bit in `CAN_GIS` and `CAN_GIF` registers is cleared and there is at least 1 bit in `CAN_RML1` still set, then the bit in the `CAN_GIS` and `CAN_GIF` registers is not set again. The internal interrupt source signal is only active if a new bit in `CAN_RML1` is set.
- Abort acknowledge event interrupt (`CAN_GIM.AAIM`, `CAN_GIS.AAIS`, `CAN_GIF.AAIF`): At least 1 `CAN_AA1.MB` bit in the `CAN_AA1` registers is set. If the bit in the `CAN_GIS` and `CAN_GIF` registers is cleared and there is at least 1 bit in `CAN_AA1` still set, then the bit in the `CAN_GIS` and `CAN_GIF` registers is not set again. The internal interrupt source signal is only active if a new bit in `CAN_AA1` is set. The `CAN_AA1.MB` bits maintain state even after the corresponding mailbox n is disabled (`CAN_MC1` = 0).
- Access to unimplemented address event interrupt (`CAN_GIM.UIAIM`, `CAN_GIS.UIAIS`, `CAN_GIF.UIAIF`): There was a CPU access to an address which is not implemented in the controller module.
- Wake-up event interrupt (`CAN_GIM.WUIM`, `CAN_GIS.WUIS`, `CAN_GIF.WUIF`): The CAN module has left the sleep mode because of detected activity on the CAN bus line.
- Bus-off event interrupt (`CAN_GIM.BOIM`, `CAN_GIS.BOIS`, `CAN_GIF.BOIF`): The CAN module has entered the bus-off state. This interrupt source is active if the status of the CAN core changes from normal operation mode to the bus-off mode. If the bit in the `CAN_GIS` and `CAN_GIF` registers is cleared and the bus-off mode is still active, then this bit is not set again. If the module leaves the bus-off mode, the bit in the `CAN_GIS` and `CAN_GIF` registers remains set, if not explicitly cleared.
- Error-passive event interrupt (`CAN_GIM.EPIM`, `CAN_GIS.EPIS`, `CAN_GIF.EPIF`): The CAN module has entered the error-passive state. This interrupt source is active if the status of the CAN module changes from the error-active mode to the error-passive mode. If the bit in the `CAN_GIS` and `CAN_GIF` registers is cleared and the error-passive mode is still active, then this bit is not set again. If the module leaves the error-passive mode, the bit in the `CAN_GIS` and `CAN_GIF` registers remains set, if not explicitly cleared.
- Error warning receive event interrupt (`CAN_GIM.EWRIM`, `CAN_GIS.EWRIS`, `CAN_GIF.EWRIF`): The CAN receive error counter (`CAN_CEC.RXECNT`) has reached the warning limit. If the bit in the `CAN_GIS` and `CAN_GIF` registers) is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the `CAN_GIS` and `CAN_GIF` registers remains set, if not explicitly cleared.
- Error warning transmit interrupt (`CAN_GIM.EWTIM`, `CAN_GIS.EWTIS`, `CAN_GIF.EWTIF`): The CAN transmit error counter (`CAN_CEC.TXECNT`) has reached the warning limit. If the bit in the `CAN_GIS` and `CAN_GIF` registers is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the `CAN_GIS` and `CAN_GIF` registers remains set, if not explicitly cleared.

Event Counter

For diagnostic functions, it is possible to use the universal counter as an event counter. The counter can be programmed in the CAN_UCCNF[3:0] field to increment on one of these conditions:

- 0x6 – CAN error frame. Counter increments if there is an error frame on the CAN bus line.
- 0x7 – CAN overload frame. Counter increments if there is an overload frame on the CAN bus line.
- 0x8 – Lost arbitration. Counter increments every time arbitration on the CAN line is lost during transmission.
- 0x9 – Transmission aborted. Counter increments every time arbitration is lost or a transmit request is canceled (CAN_AA1 is set).
- 0xA – Transmission succeeded. Counter increments every time a message sends without detected errors (CAN_TA1 is set).
- 0xB – Receive message rejected. Counter increments every time a message is received without detected errors but not stored in a mailbox because there is no matching identifier found.
- 0xC – Receive message lost. Counter increments every time a message is received without detected errors but not stored in a mailbox because the mailbox contains unread data (CAN_RML1 is set).
- 0xD – Message received. Counter increments every time a message is received without detected errors, whether the received message is rejected or stored in a mailbox.
- 0xE – Message stored. Counter increments every time a message is received without detected errors, has an identifier that matches an enabled receive mailbox, and is stored in the receive mailbox (CAN_RMP1 is set).
- 0xF – Valid message. Counter increments every time a valid transmit or receive message is detected on the CAN bus line.

CAN Warnings and Errors

The processor controls CAN warnings and errors using the error counter (CAN_CEC) register, the error status (CAN_ESR) register, and the error counter warning level (CAN_EWR) register. The following sections describe error handling.

Programmable Warning Limits

Programs can set the warning level for CAN_GIS.EWTIS and CAN_GIS.EWRIS separately by writing to the CAN_EWR.EWLREC and CAN_EWR.EWLTEC fields. After power-on reset, the CAN_EWR register is set to the default warning level of 96 for both error counters. After a soft reset, the contents of this register remain unchanged.

Error Handling

Error management is a part of the CAN standard. Several different kinds of bus errors can occur during transmissions:

- Bit error – Only the transmitting node detects this error. Whenever a node transmits, it continuously monitors its receive pin (CAN_RX) and compares the received bit with the transmitted bit. During the arbitration phase, the node postpones the transmission if the received and transmitted bits do not match. However, after the arbitration phase, a bit error is signaled any time the value on `CAN_RX` does not equal what is transmitted on the `CAN_TX` pin. (The arbitration phase completes when the `CAN_MB[nn]_ID1.RTR` bit is sent successfully.)

- Form error. Occurs when a fixed-form bit position in the CAN frame contains one or more illegal bits. Occurs when a dominant bit is detected at a delimiter or end of frame bit position.
- Acknowledge error. Occurs whenever a message is sent and no receivers drive an acknowledge bit.
- CRC error. Occurs whenever a receiver calculates the CRC on the data it received and finds it different than the CRC that transmitted on the bus itself.
- Stuff error. The CAN specification requires the transmitter to insert an extra stuff bit of opposite value after 5 bits have transmitted with the same value. The receiver disregards the value of the stuff bits. However, it takes advantage of the signal edge to resynchronize itself. A stuff error occurs on receiving nodes whenever the sixth consecutive bit value is the same as the previous 5 bits.

Once the CAN module detects any of the errors, it updates the `CAN_ESR` and `CAN_CEC` registers. In addition to the standard errors, the `CAN_ESR.SAO` flag signals when the `CAN_RX` pin sticks at dominant level, indicating a possibility of shorted wires.

Error Frames

It is important that all nodes on the CAN bus ignore data frames that any single node failed to receive. Every node sends an error frame as soon as it has detected an error as shown in the ***CAN Error Example*** figure.

A device that has detected an error still completes the ongoing bit. It initiates an error frame by sending six dominant and eight recessive bits to the bus. Since this activity is a violation of the bit stuffing rule, all nodes are signaled to discard the ongoing frame. (All receivers that did not detect the transmission error in the first instance now detect a stuff bit error.)

The transmitter can detect a normal bit error sooner. It aborts the transmission of the ongoing frame and tries resending it later.

When all nodes on the bus have detected the error, they also send six dominant and eight recessive bits to the bus. The resulting error frame consists of two different fields. The first field is the superposition of error flags contributed from the different stations, which are a sequence of 6–12 dominant bits. The second field is the error delimiter and consists of eight recessive bits indicating the end of frame.

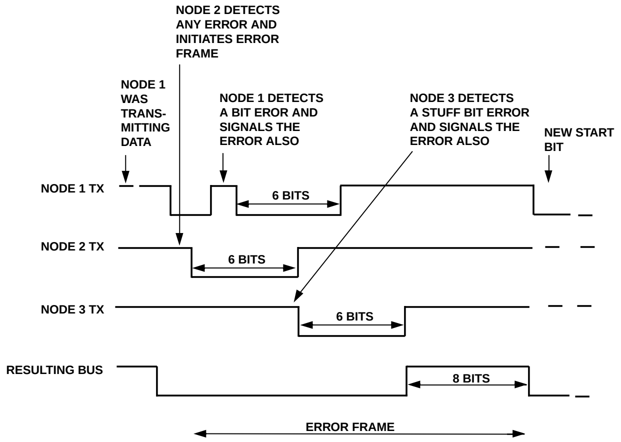

NODE 2 DETECTS ANY ERROR AND INITIATES ERROR FRAME
NODE 1 WAS TRANS-MITTING DATA
NODE 1 DETECTS A BIT EROR AND SIGNALS THE ERROR ALSO
NODE 3 DETECTS A STUFF BIT ERROR AND SIGNALS THE ERROR ALSO
NEW START BIT
NODE 1 TX
6 BITS
NODE 2 TX
6 BITS
NODE 3 TX
6 BITS
RESULTING BUS
8 BITS
ERROR FRAME

Figure 25-12: CAN Error Example

For CRC errors, the error frame initiates at the end of the frame, rather than immediately after the failing bit.

After having received eight recessive bits, every node knows that the error condition is resolved and, if messages are pending, starts transmission. The transmitter that had to abort its operation must win the new arbitration again; otherwise its message is delayed as determined by priority.

Because the transmission of an error frame destroys the frame under transmission, a faulty node erroneously detecting an error can block the bus. So, there are two node states which determine a nodes right to signal an error—error-active and error-passive.

- *Error-active* nodes have an error detection rate below a certain limit. These nodes drive an active error flag of six dominant bits.
- *Error-passive* nodes have a higher error detection rate and can have a local problem and therefore have a limited right to signal errors. These nodes drive a passive error flag consisting of six recessive bits. Therefore, an error-passive transmitting node is still able to inform the other nodes about the aborting of a self-transmitted frame. But, it is no longer able to destroy correctly received frames of other nodes.

Error Levels

The CAN specification requires each node in the system to operate at one of three levels. The ***CAN Error Level Description*** table describes the levels. This functionality prevents nodes with high error rates from blocking the entire network, as local hardware can cause the errors. The CAN module provides an error counter for transmit (TEC) and an error counter for receive (REC). The `CAN_CEC` register contains each of these 8-bit counters.

After initialization, both the TEC and the REC counters are 0. Each time a bus error occurs, one of the counters increments by either 1 or 8, depending on the error situation. Refer to version 2.0 of the CAN specification. Successful transmit or receive operations decrement the respective counter by 1.

If either of the error counters exceeds 127, the CAN module goes into an error-passive state and the `CAN_STAT.EP` bit is set. Once this state occurs, the module is not allowed to send any more active error frames. However, the module can still transmit messages and signal passive error frames in case the transmission fails due to bit errors.

If one of the counters exceeds 255 (that is, when an 8-bit counter overflows), the CAN module disconnects from the bus and it goes into bus-off mode. In this mode, the `CAN_STAT.EBO` bit is set. Software intervention is needed for recovery from this state, unless the `CAN_CTL.ABO` bit is enabled. The bit puts the module into active mode after the bus-off recovery sequence.

Table 25-5: CAN Error Level Description

Level	Condition	Description
Error active	Transmit and receive error counters <128	This level is the initial condition level. As long as errors stay below 128, the node drives active error flags during error frames.
Error passive	Transmit or receive error counter-value from 128 through 255, inclusive	Errors have accumulated to a level that requires the node to drive passive error flags during error frames
Bus off	Transmit or receive error counters greater than 255	CAN module goes into bus-off mode

In addition to the three levels in the table, the CAN module also generates separate transmit and receive warnings (CAN specification enhancement). By default, when one of the error counters exceeds 96, it signals and reports a warning in the `CAN_STAT` register. The CAN receive warning flag (`CAN_STAT.WR`) bit is set when `CAN_CEC.RXECNT` exceeds 96. The CAN transmit warning flag (`CAN_STAT.WT`) bit is set when `CAN_CEC.TXECNT` exceeds 96. The error warning level can be programmed using the error warning register (`CAN_EWR`).

Additionally, interrupt requests can occur for all of these levels by unmasking them in the global CAN interrupt mask register (`CAN_GIM`). These sources include: the bus-off interrupt (`CAN_GIM.BOIM`), the error-passive interrupt (`CAN_GIM.EPIM`), the error warning receive interrupt (`CAN_GIM.EWRIM`), and the error warning transmit interrupt (`CAN_GIM.EWTIM`).

During the bus-off recovery sequence, internal logic sets the configuration mode request `CAN_CTL.CCR` bit. The CAN core module does not automatically come out of the bus-off mode. The `CAN_CTL.CCR` bit cannot be reset until the bus-off recovery sequence completes.

NOTE: Set the `CAN_CTL.ABO` bit to override this behavior. After exiting the bus-off or configuration modes, the CAN error counters are reset.

CAN Debug and Test Modes

The CAN module contains test mode features that aid in the debugging of the CAN software and system.

NOTE: When using these features, the CAN module does not always comply to the CAN specification. Enable or disable all test modes only when the module is in configuration mode (`CAN_STAT.CCA`=1) or suspend mode (`CAN_STAT.CSA`=1).

The `CAN_DBG.CDE` bit provides access to all of the debug features. Set this bit to enable the test mode. Write to the bit first before writing to the `CAN_DBG` register. When the `CAN_DBG.CDE` bit is cleared, all debug features are disabled.

When the `CAN_DBG.CDE` bit is set, it enables writes to the other bits of the `CAN_DBG` register. It also enables these features, which are not compliant to the CAN standard:

- Bit timing registers can be changed anytime, not only during configuration mode. The group includes the `CAN_CLK` and `CAN_TIMING` registers.
- Write access is allowed to the normally read-only `CAN_CEC` register.

The following list describes other bits in the debug register.

- The CAN module uses the `CAN_DBG.MRB` bit to enable the read back mode. In this mode, a message transmitted on the CAN bus (or through an internal loopback mode) is received back directly to the internal receive buffer. After a correct transmission, the internal logic treats this transfer as a normal receive message. This feature allows testing of most of the CAN features without an external device.
- The `CAN_DBG.MAA` bit allows the CAN module to generate its own acknowledge during the ACK slot of the CAN frame. No external devices or connections are necessary to read back a transmit message. In this mode, the sent message is automatically stored in the internal receive buffer. In auto-acknowledge mode, the module itself transmits the acknowledge. This acknowledge can be programmed to appear on the `CAN_TX` pin, if `CAN_DBG.DIL`=1 and `CAN_DBG.DTO`= 0. If the acknowledge is only used internally, then set these test mode bits to `CAN_DBG.DIL`= 0 and `CAN_DBG.DTO`=1.
- The CAN module uses the `CAN_DBG.DIL` bit to internally enable the transmit output to be routed back to the receive input.
- The CAN module uses the `CAN_DBG.DTO` bit to disable the `CAN_TX` output pin. When this bit is set, the `CAN_TX` pin continuously drives recessive bits.
- The CAN module uses the `CAN_DBG.DRI` bit to disable the `CAN_RX` input. When set, the internal logic receives recessive bits or receives the internally-generated transmit value in the case of the internal loop enabled (`CAN_DBG.DIL`= 0). In either case, the value on the `CAN_RX` input pin is ignored.
- The CAN module uses the `CAN_DBG.DEC` bit to disable the transmit and receive error counters in the `CAN_CEC` register. When this bit is set, the `CAN_CEC` holds its current contents and is not allowed to increment or decrement the error counters. This mode does not conform to the CAN specification.

NOTE: Write to the error counter registers in debug mode only. Write-access during reception can lead to undefined values. The maximum value which can be written into the error counters is 255. Therefore, the error counter value of 256, which forces the module into the bus off state, cannot be written into the error counter registers.

Table 25-6: Common CAN Test Mode Bit Combinations

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
X	X	X	X	X	0	Normal mode, not debug mode
0	X	X	X	X	X	No readback of transmit message

Table 25-6: Common CAN Test Mode Bit Combinations (Continued)

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
1	0	1	0	0	1	Normal transmission on CAN bus line. Read back. External acknowledge from external device required.
1	1	1	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. `CAN_RX` input is enabled.
1	1	0	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge transmit on CAN bus line. `CAN_RX` input and internal loop are enabled (internal OR of TX and RX)
1	1	0	0	1	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. `CAN_RX` input is ignored. Internal loop is enabled.
1	1	0	1	1	1	No transmission on CAN bus line. Read back. No external acknowledge required. Nether transmit message nor acknowledge are transmitted on `CAN_TX`. `CAN_RX` input is ignored. Internal loop is enabled.

ADSP-SC58x CAN Register Descriptions

Controller Area Network (CAN) contains the following registers.

Table 25-7: ADSP-SC58x CAN Register List

Name	Description
`CAN_AA1`	Abort Acknowledge 1 Register

Table 25-7: ADSP-SC58x CAN Register List (Continued)

Name	Description
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Time Stamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register

Table 25-7: ADSP-SC58x CAN Register List (Continued)

Name	Description
`CAN_MD2`	Mailbox Direction 2 Register
`CAN_OPSS1`	Overwrite Protection/Single Shot Transmission 1 Register
`CAN_OPSS2`	Overwrite Protection/Single Shot Transmission 2 Register
`CAN_RFH1`	Remote Frame Handling 1 Register
`CAN_RFH2`	Remote Frame Handling 2 Register
`CAN_RML1`	Receive Message Lost 1 Register
`CAN_RML2`	Receive Message Lost 2 Register
`CAN_RMP1`	Receive Message Pending 1 Register
`CAN_RMP2`	Receive Message Pending 2 Register
`CAN_STAT`	Status Register
`CAN_TA1`	Transmission Acknowledge 1 Register
`CAN_TA2`	Transmission Acknowledge 2 Register
`CAN_TIMING`	Timing Register
`CAN_TRR1`	Transmission Request Reset 1 Register
`CAN_TRR2`	Transmission Request Reset 2 Register
`CAN_TRS1`	Transmission Request Set 1 Register
`CAN_TRS2`	Transmission Request Set 2 Register
`CAN_UCCNF`	Universal Counter Configuration Mode Register
`CAN_UCCNT`	Universal Counter Register
`CAN_UCRC`	Universal Counter Reload/Capture Register

Abort Acknowledge 1 Register

The `CAN_AA1` register indicates a transmission abort (due to lost arbitration or a CAN error) for mailboxes 8 through 15. Each bit in this register indicates a transmission abort for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

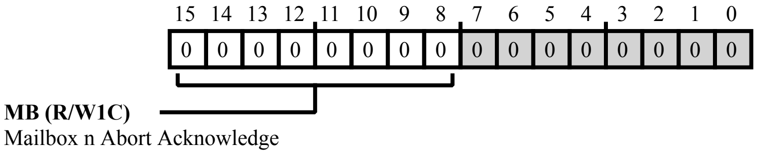

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Abort Acknowledge

Figure 25-13: CAN_AA1 Register Diagram

Table 25-8: CAN_AA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Abort Acknowledge 2 Register

The `CAN_AA2` register indicates a transmission abort (due to lost arbitration or a CAN error) for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates a transmission abort for the corresponding mailbox when set (=1).

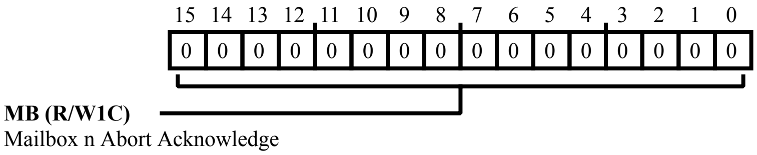

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Abort Acknowledge

Figure 25-14: CAN_AA2 Register Diagram

Table 25-9: CAN_AA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Acceptance Mask (H) Register

The `CAN_AM[nn]H` register and `CAN_AM[nn]L` register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

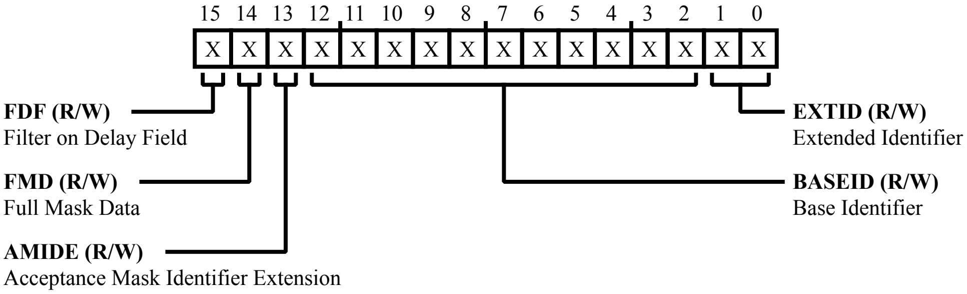

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
FDF (R/W)
Filter on Delay Field
FMD (R/W)
Full Mask Data
AMIDE (R/W)
Acceptance Mask Identifier Extension
EXTID (R/W)
Extended Identifier
BASEID (R/W)
Base Identifier

Figure 25-15: CAN_AM[nn]H Register Diagram

Table 25-10: CAN_AM[nn]H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	FDF	Filter on Delay Field. The `CAN_AM[nn]H.FDF` bit selects the operation of the `CAN_AM[nn]H` register and `CAN_AM[nn]L` register when the `CAN_CTL.DNM` bit is enabled. If the `CAN_AM[nn]H.FDF` bit is set, the corresponding `CAN_AM[nn]L.EXTID` bits hold the data field mask. If the `CAN_AM[nn]H.FDF` bit is cleared, the corresponding `CAN_AM[nn]L.EXTID` bits hold the high bits of the extended identifier mask.
14 (R/W)	FMD	Full Mask Data. The `CAN_AM[nn]H.FMD` bit works with the `CAN_AM[nn]H.FDF` bit to determine data field filtering. For information about data field filtering, see the Receive Operation section.
13 (R/W)	AMIDE	Acceptance Mask Identifier Extension. The `CAN_AM[nn]H.AMIDE` bit enables the comparison of the received message ID to the value in the `CAN_AM[nn]H.EXTID` and `CAN_AM[nn]L.EXTID` bits.
12:2 (R/W)	BASEID	Base Identifier. The `CAN_AM[nn]H.BASEID` bits hold the base ID for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The `CAN_AM[nn]H.EXTID` bits hold the extended ID (upper two bits) for acceptance mask operations.

Acceptance Mask (L) Register

The `CAN_AM[nn]L` register and `CAN_AM[nn]H` register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

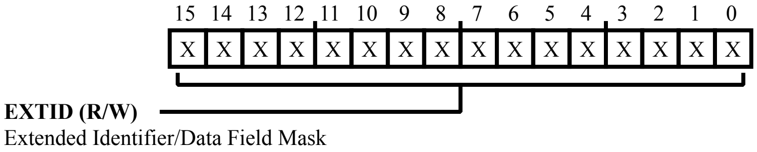

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
EXTID (R/W)
Extended Identifier/Data Field Mask

Figure 25-16: CAN_AM[nn]L Register Diagram

Table 25-11: CAN_AM[nn]L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Mask. The `CAN_AM[nn]L.EXTID` bits hold the extended ID (lower 16 bits) for data field mask in acceptance mask operations.

Error Counter Register

The `CAN_CEC` register, `CAN_ESR` register, and `CAN_EWR` register control CAN warnings and errors. For detailed information about error and warning operations, see the Event Control section.

The `CAN_CEC` register holds an error counter for transmit (`CAN_CEC.TXECNT`) and an error counter for receive (`CAN_CEC.RXECNT`). After initialization, both counters are 0. Each time a bus error occurs, one of the counters is incremented by either 1 or 8, depending on the error situation (documented in Version 2.0 of CAN Specification). Successful transmit and receive operations decrement the respective counter by 1.

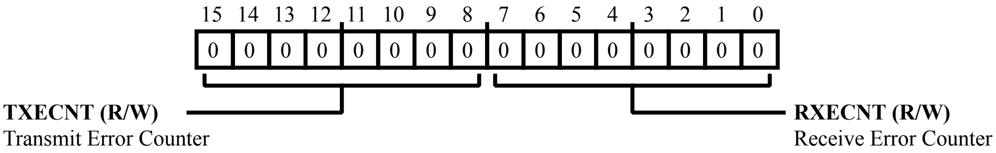


Figure 25-17: CAN_CEC Register Diagram

Table 25-12: CAN_CEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	TXECNT	Transmit Error Counter. The `CAN_CEC.TXECNT` bits hold the transmit error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful transmit operations.
7:0 (R/W)	RXECNT	Receive Error Counter. The `CAN_CEC.RXECNT` bits hold the receive error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful receive operations.

Clock Register

The CAN_CLK register selects the bit rate prescaler for calculating the time quantum (TQ), which is used to derive the CAN clock from the system clock (CDU0_CLKO4). For more information about bit timing and clock operation, see the CAN Operating Modes section.

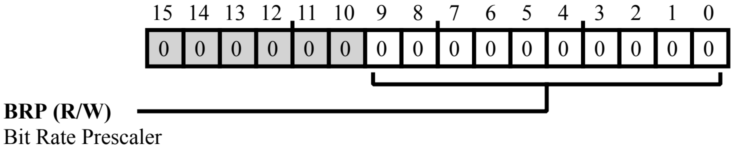


Figure 25-18: CAN_CLK Register Diagram

Table 25-13: CAN_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	BRP	Bit Rate Prescaler. The `CAN_CLK.BRP` bits select the bit rate prescaler value, which is used to calculate the time quantum for CAN bit timing. The formula using `CAN_CLK.BRP` to calculate the time quantum is: TQ = (BRP+1) / CDU0_CLKO4 Note that it is recommended that the `CAN_CLK.BRP` value be greater than or equal to 4. For more information about bit timing, see the Operating Modes section.

CAN Master Control Register

The CAN_CTL register controls CAN mode requests, including soft reset.

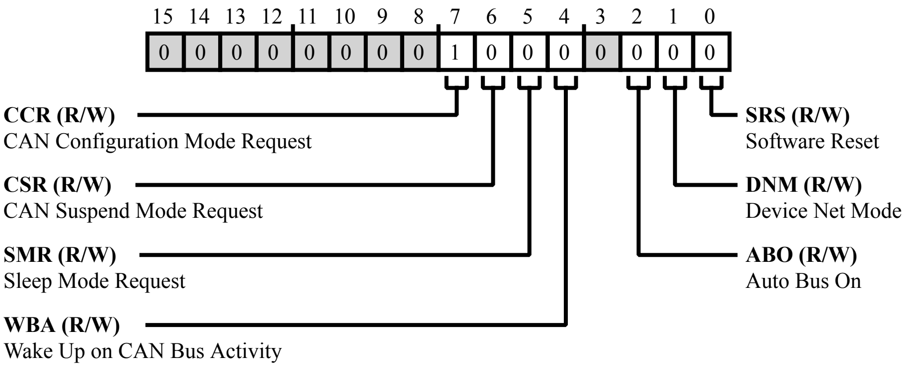

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
CCR (R/W)
CAN Configuration Mode Request
CSR (R/W)
CAN Suspend Mode Request
SMR (R/W)
Sleep Mode Request
WBA (R/W)
Wake Up on CAN Bus Activity
SRS (R/W)
Software Reset
DNM (R/W)
Device Net Mode
ABO (R/W)
Auto Bus On

Figure 25-19: CAN_CTL Register Diagram

Table 25-14: CAN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CCR	CAN Configuration Mode Request. The `CAN_CTL.CCR` bit requests that the CAN enter configuration mode. Note that the CAN should always be put in configuration mode before modifying the CAN_CLK or CAN_TIMING registers.	
		0	No Request (Exit Configuration Mode)
		1	Request Configuration Mode
6 (R/W)	CSR	CAN Suspend Mode Request. The `CAN_CTL.CSR` bit requests that the CAN enter suspend mode. The CAN enters suspend mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Suspend Mode)
		1	Request Suspend Mode
5 (R/W)	SMR	Sleep Mode Request. The `CAN_CTL.SMR` bit requests that the CAN enter sleep mode. The CAN enters sleep mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Sleep Mode)
		1	Request Sleep Mode

Table 25-14: CAN_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	WBA	Wake Up on CAN Bus Activity. The `CAN_CTL.WBA` bit enables wake on CAN bus activity. When enabled, a dominant bit on the `CAN_RX` pin ends sleep mode (also, the default wake up condition of a write to the `CAN_INT` register).	
		0	Disable Wake on Bus Activity
		1	Enable Wake on Bus Activity
2 (R/W)	ABO	Auto Bus On. The `CAN_CTL.ABO` bit selects whether (if enabled) the CAN enters active mode after the bus-off recovery sequence or (if disabled) the CAN enters configuration mode after the bus-off recovery sequence.	
		0	Disable Auto Bus On
		1	Enable Auto Bus On
1 (R/W)	DNM	Device Net Mode. The `CAN_CTL.DNM` bit enables mailbox filtering on a data field. The filtering is done on the standard ID of the message and data fields. For more information, see the `CAN_AM[nn]H.FDF` bit description.	
		0	Disable Device Net Mode
		1	Enable Device Net Mode
0 (R/W)	SRS	Software Reset. The `CAN_CTL.SRS` bit resets the CAN, bringing all control registers to a defined state. Soft reset is entered immediately after software has set the `CAN_CTL.SRS` bit.	
		0	No Action
		1	Reset CAN

Debug Register

The CAN_DBG register controls CAN debug modes, including CAN_TX and CAN_RX pin enable and disable.

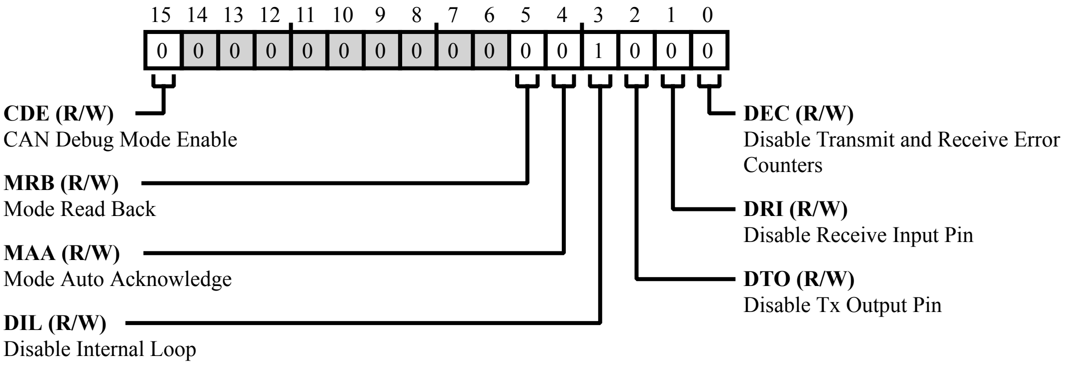

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
CDE (R/W)
CAN Debug Mode Enable
MRB (R/W)
Mode Read Back
MAA (R/W)
Mode Auto Acknowledge
DIL (R/W)
Disable Internal Loop
DEC (R/W)
Disable Transmit and Receive Error Counters
DRI (R/W)
Disable Receive Input Pin
DTO (R/W)
Disable Tx Output Pin

Figure 25-20: CAN_DBG Register Diagram

Table 25-15: CAN_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	CDE	CAN Debug Mode Enable. The CAN_DBG.CDE bit enables debug mode. This bit must be written first before subsequent writes to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all CAN debug features are disabled.	
		0	Disable Debug Mode
		1	Enable Debug Mode
5 (R/W)	MRB	Mode Read Back. The CAN_DBG.MRB bit enables read back mode. When enabled, a message transmitted on the CAN bus or through an internal loop back mode is received back directly to the internal receive buffer.	
		0	Disable Read Back Mode
		1	Enable Read Back Mode
4 (R/W)	MAA	Mode Auto Acknowledge. The CAN_DBG.MAA bit enables auto acknowledge mode, allowing the CAN to generate its own acknowledge during the ACK slot of the CAN frame. The CAN_DBG.MAA acknowledge appears on the CAN_TX pin if CAN_DBG.DIL =1 and CAN_DBG.DTO =0. If the acknowledge is only going to be used internally, these test mode bits should be set to CAN_DBG.DIL = 0 and CAN_DBG.DTO =1.	
		0	Disable Auto Acknowledge Mode
		1	Enable Auto Acknowledge Mode

Table 25-15: CAN_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIL	Disable Internal Loop. The `CAN_DBG.DIL` bit disables internal loop mode, which routes the transmit output to the receive input.	
		0	Enable Internal Loop
		1	Disable Internal Loop
2 (R/W)	DTO	Disable Tx Output Pin. The `CAN_DBG.DTO` bit disables the `CAN_TX` pin.	
		0	Enable Tx Output Pin
		1	Disable Tx Output Pin, Drive Recessive
1 (R/W)	DRI	Disable Receive Input Pin. The `CAN_DBG.DRI` bit disables the `CAN_RX` pin.	
		0	Enable Rx Input Pin
		1	Disable Rx Input Pin, Drive Recessive Internally
0 (R/W)	DEC	Disable Transmit and Receive Error Counters. The `CAN_DBG.DEC` bit disables the transmit and receive error counters in the `CAN_CEC` register. When set, the `CAN_CEC` holds its current contents and is not allowed to increment or decrement the error counters. Note that this mode does not conform to the CAN specification.	
		0	Enable CEC Tx and Rx Error Counters
		1	Disable CEC Tx and Rx Error Counters

Error Status Register

The CAN_ESR register, CAN_CEC register, and CAN_EWR register control CAN warnings and errors. All bits in the CAN_ESR register are W1C. Note that the CAN updates the CAN_CEC register when error status is detected in the CAN_ESR register. For detailed information about error and warning operations, see the Operating Modes section.

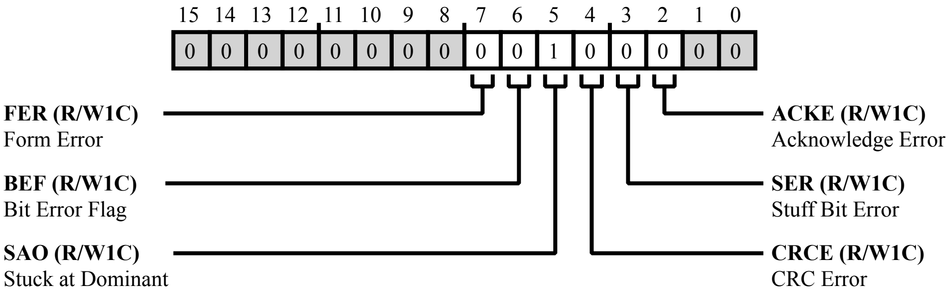

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
FER (R/W1C)
Form Error
BEF (R/W1C)
Bit Error Flag
SAO (R/W1C)
Stuck at Dominant
ACKE (R/W1C)
Acknowledge Error
SER (R/W1C)
Stuff Bit Error
CRCE (R/W1C)
CRC Error

Figure 25-21: CAN_ESR Register Diagram

Table 25-16: CAN_ESR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	FER	Form Error. The `CAN_ESR.FER` bit indicates when a form error occurs, indicating that a fixed-form bit position in the CAN frame contains one or more illegal bits. This occurs when a dominant bit is detected at a delimiter or end-of-frame bit position.	
		0	No Status
		1	Form Error
6 (R/W1C)	BEF	Bit Error Flag. The `CAN_ESR.BEF` bit indicates (detected by the transmitting node only) when the value on the `CAN_RX` pin does not equal what is being transmitted on the `CAN_TX` pin. When a node is transmitting, it continuously monitors its receive pin (`CAN_RX`) and compares the received data with the transmitted data. The node postpones the transmission (during the arbitration phase) if the received and transmitted data do not match. After the arbitration phase (`CAN_MB[nn]_ID1.RTR` bit sent successfully), a bit error is signaled when the value on the `CAN_RX` pin does not equal what is being transmitted on the `CAN_TX` pin.	
		0	No Status
		1	Bit Error Flag

Table 25-16: CAN_ESR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	SAO	Stuck at Dominant. The `CAN_ESR.SAO` bit indicates when the `CAN_RX` pin sticks at dominant level, indicating that shorted wires are likely.	
		0	No Status
		1	Stuck At Dominant
4 (R/W1C)	CRCE	CRC Error. The `CAN_ESR.CRCE` bit indicates when a CRC error occurs. This error may occur when a receiver calculates the CRC on the data it received and finds the value different than the CRC that was transmitted on the bus.	
		0	No Status
		1	CRC Error
3 (R/W1C)	SER	Stuff Bit Error. The `CAN_ESR.SER` bit indicates when a stuff bit error (stuffed 6th consecutive bit value is the same as the previous five bits) occurs. The CAN specification requires that the transmitter insert an extra stuff bit of opposite value after 5 bits have been transmitted with the same value. The receiver disregards the value of these stuff bits. The receiver takes advantage of the signal edge to re-synchronize itself. A stuff bit error occurs on receiving nodes when the 6th consecutive bit value is the same as the previous five bits.	
		0	No Status
		1	Stuff Bit Error Receive
2 (R/W1C)	ACKE	Acknowledge Error. The `CAN_ESR.ACKE` bit indicates when an acknowledge error occurs, indicating that a message is sent and no receivers drive an acknowledge bit.	
		0	No Status
		1	Acknowledge Error

Error Counter Warning Level Register

The `CAN_EWR` register, `CAN_CEC` register, and `CAN_ESR` register control CAN warnings and errors. For detailed information about error and warning operations, see the Operating Modes section.

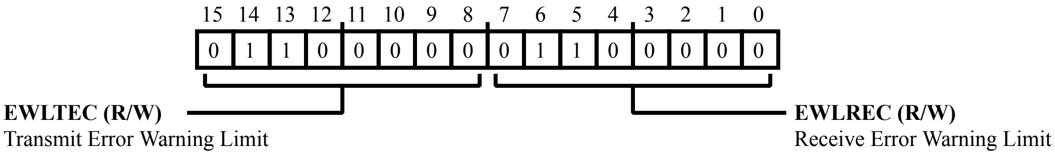

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0
EWLTEC (R/W)
Transmit Error Warning Limit
EWLREC (R/W)
Receive Error Warning Limit

Figure 25-22: CAN_EWR Register Diagram

Table 25-17: CAN_EWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	EWLTEC	Transmit Error Warning Limit. The `CAN_EWR.EWLTEC` bits select the transmit error warning limit, which is used as a condition for the `CAN_GIS.EWTIS` interrupt.
7:0 (R/W)	EWLREC	Receive Error Warning Limit. The `CAN_EWR.EWLREC` bits select the receive error warning limit, which is used as a condition for the `CAN_GIS.EWRIS` interrupt.

Global CAN Interrupt Flag Register

The CAN_GIF register, CAN_GIF register, and CAN_GIM register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIF register holds the interrupt flag. The `CAN_INT.GIRQ` bit is only asserted if a bit in the CAN_GIF is set. The `CAN_INT.GIRQ` bit remains set as long as at least one bit in the CAN_GIF register is set. All bits in this register are W1C.

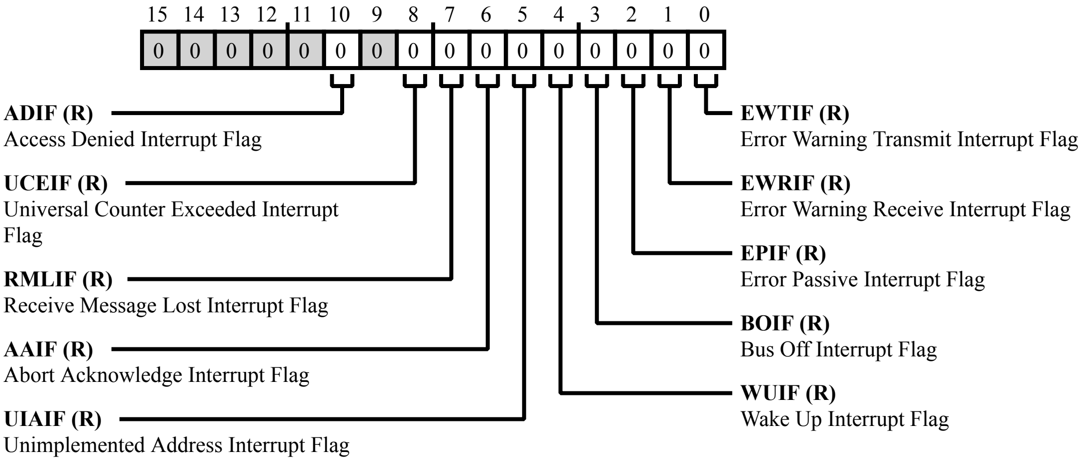

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADIF (R)
Access Denied Interrupt Flag
UCEIF (R)
Universal Counter Exceeded Interrupt Flag
RMLIF (R)
Receive Message Lost Interrupt Flag
AAIF (R)
Abort Acknowledge Interrupt Flag
UIAIF (R)
Unimplemented Address Interrupt Flag
EWTIF (R)
Error Warning Transmit Interrupt Flag
EWRIF (R)
Error Warning Receive Interrupt Flag
EPIF (R)
Error Passive Interrupt Flag
BOIF (R)
Bus Off Interrupt Flag
WUIF (R)
Wake Up Interrupt Flag

Figure 25-23: CAN_GIF Register Diagram

Table 25-18: CAN_GIF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	ADIF	Access Denied Interrupt Flag. The `CAN_GIF.ADIF` bit indicates that the access denied interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
8 (R/NW)	UCEIF	Universal Counter Exceeded Interrupt Flag. The `CAN_GIF.UCEIF` bit indicates that the universal counter exceeded interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 25-18: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	RMLIF	Receive Message Lost Interrupt Flag. The `CAN_GIF.RMLIF` bit indicates that the receive message lost interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
6 (R/NW)	AAIF	Abort Acknowledge Interrupt Flag. The `CAN_GIF.AAIF` bit indicates that the abort acknowledge interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
5 (R/NW)	UIAIF	Unimplemented Address Interrupt Flag. The `CAN_GIF.UIAIF` bit indicates that the unimplemented address interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
4 (R/NW)	WUIF	Wake Up Interrupt Flag. The `CAN_GIF.WUIF` bit indicates that the wake up interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
3 (R/NW)	BOIF	Bus Off Interrupt Flag. The `CAN_GIF.BOIF` bit indicates that the bus off interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
2 (R/NW)	EPIF	Error Passive Interrupt Flag. The `CAN_GIF.EPIF` bit indicates that the error passive mode interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
1 (R/NW)	EWRIF	Error Warning Receive Interrupt Flag. The `CAN_GIF.EWRIF` bit indicates that the error warning receive interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 25-18: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	EWTIF	Error Warning Transmit Interrupt Flag. The `CAN_GIF.EWTIF` bit indicates that the error warning transmit interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Global CAN Interrupt Mask Register

The `CAN_GIM` register, `CAN_GIF` register, and `CAN_GIF` register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The `CAN_GIM` register holds the interrupt mask. The interrupt mask bits only affect the content of the `CAN_GIF` register. If the mask bit is not set (enabled/unmasked), the corresponding flag bit is not set when the event occurs.

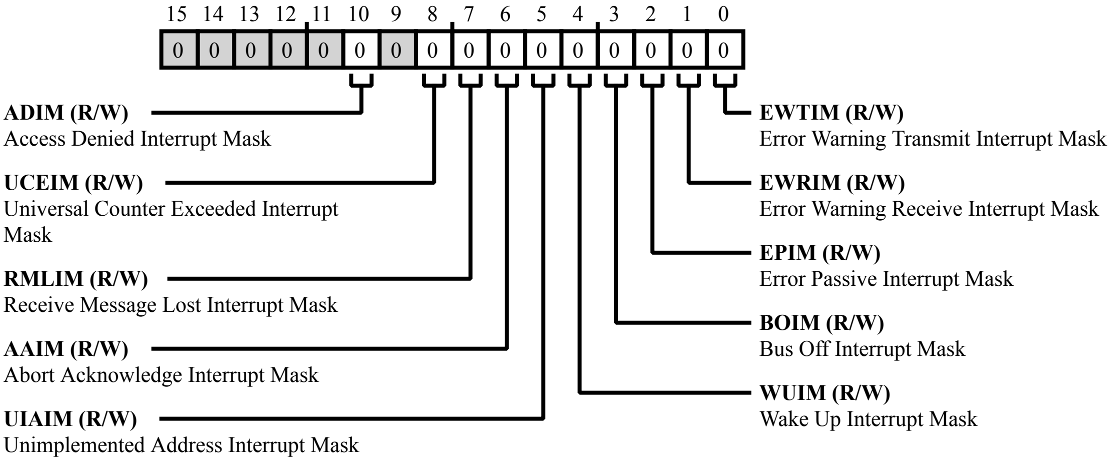

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADIM (R/W)
Access Denied Interrupt Mask
UCEIM (R/W)
Universal Counter Exceeded Interrupt Mask
RMLIM (R/W)
Receive Message Lost Interrupt Mask
AAIM (R/W)
Abort Acknowledge Interrupt Mask
UIAIM (R/W)
Unimplemented Address Interrupt Mask
EWTIM (R/W)
Error Warning Transmit Interrupt Mask
EWRIM (R/W)
Error Warning Receive Interrupt Mask
EPIM (R/W)
Error Passive Interrupt Mask
BOIM (R/W)
Bus Off Interrupt Mask
WUIM (R/W)
Wake Up Interrupt Mask

Figure 25-24: CAN_GIM Register Diagram

Table 25-19: CAN_GIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	ADIM	Access Denied Interrupt Mask. The `CAN_GIM.ADIM` bit enables (unmasks) the access denied interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
8 (R/W)	UCEIM	Universal Counter Exceeded Interrupt Mask. The `CAN_GIM.UCEIM` bit enables (unmasks) the universal counter exceeded interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
7 (R/W)	RMLIM	Receive Message Lost Interrupt Mask. The `CAN_GIM.RMLIM` bit enables (unmasks) the receive message lost interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 25-19: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	AAIM	Abort Acknowledge Interrupt Mask. The `CAN_GIM.AAIM` bit enables (unmasks) the abort acknowledge interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
5 (R/W)	UIAIM	Unimplemented Address Interrupt Mask. The `CAN_GIM.UIAIM` bit enables (unmasks) the unimplemented address interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
4 (R/W)	WUIM	Wake Up Interrupt Mask. The `CAN_GIM.WUIM` bit enables (unmasks) the wake up interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
3 (R/W)	BOIM	Bus Off Interrupt Mask. The `CAN_GIM.BOIM` bit enables (unmasks) the bus off interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
2 (R/W)	EPIM	Error Passive Interrupt Mask. The `CAN_GIM.EPIM` bit enables (unmasks) the error passive mode interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
1 (R/W)	EWRIM	Error Warning Receive Interrupt Mask. The `CAN_GIM.EWRIM` bit enables (unmasks) the error warning receive interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
0 (R/W)	EWTIM	Error Warning Transmit Interrupt Mask. The `CAN_GIM.EWTIM` bit enables (unmasks) the error warning transmit interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Global CAN Interrupt Status Register

The CAN_GIS register, CAN_GIF register, and CAN_GIM register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIS register holds the interrupt status. All bits in this register are W1C.

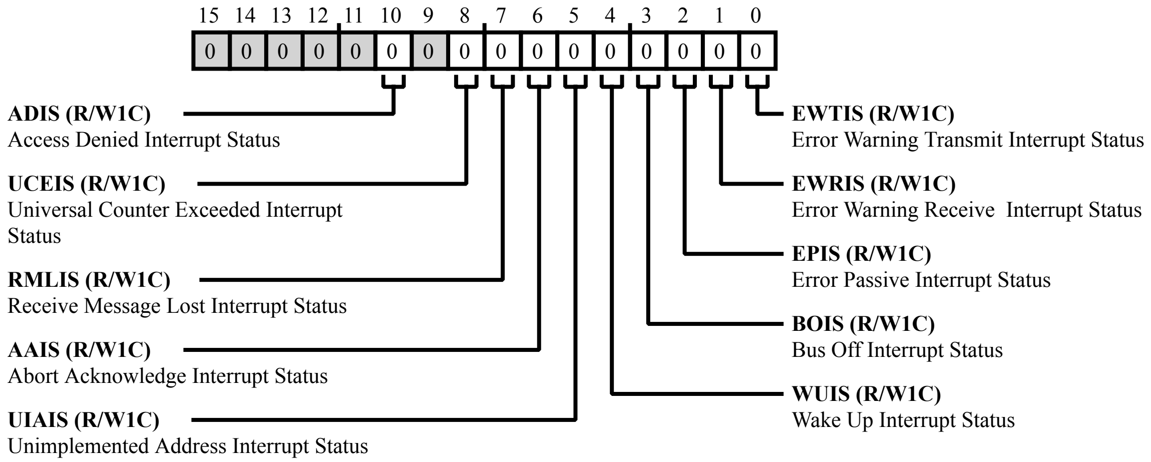

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADIS (R/W1C)
Access Denied Interrupt Status
UCEIS (R/W1C)
Universal Counter Exceeded Interrupt Status
RMLIS (R/W1C)
Receive Message Lost Interrupt Status
AAIS (R/W1C)
Abort Acknowledge Interrupt Status
UIAIS (R/W1C)
Unimplemented Address Interrupt Status
EWTIS (R/W1C)
Error Warning Transmit Interrupt Status
EWRIS (R/W1C)
Error Warning Receive Interrupt Status
EPIS (R/W1C)
Error Passive Interrupt Status
BOIS (R/W1C)
Bus Off Interrupt Status
WUIS (R/W1C)
Wake Up Interrupt Status

Figure 25-25: CAN_GIS Register Diagram

Table 25-20: CAN_GIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	ADIS	Access Denied Interrupt Status. The CAN_GIS.ADIS bit indicates when at least one access to the mailbox RAM occurred during a data update by internal logic.	
		0	No Interrupt Pending
		1	Interrupt Pending
8 (R/W1C)	UCEIS	Universal Counter Exceeded Interrupt Status. The CAN_GIS.UCEIS bit indicates when there has been an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 25-20: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RMLIS	Receive Message Lost Interrupt Status. The `CAN_GIS.RMLIS` bit indicates when a message is received for a mailbox that currently contains unread data. At least one bit in the receive message lost register (`CAN_RML1` or `CAN_RML2`) is set. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and there is at least one bit in `CAN_RML1` or `CAN_RML2` still set, the bit in `CAN_GIF` (and `CAN_GIF`) is not set again. The internal interrupt source signal is only active if a new bit in `CAN_RML1` or `CAN_RML2` is set.	
		0	No Interrupt Pending
		1	Interrupt Pending
6 (R/W1C)	AAIS	Abort Acknowledge Interrupt Status. The `CAN_GIS.AAIS` bit indicates when At least one abort acknowledge bit is set in the `CAN_AA1` or the `CAN_AA2` registers. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and there is at least one bit in `CAN_AA1` or `CAN_AA2` still set, the bit in `CAN_GIS` (and `CAN_GIF`) is not set again. The internal interrupt source signal is only active if a new bit in `CAN_AA1` or `CAN_AA2` is set. The abort acknowledge bits maintain state even after the corresponding mailbox n is disabled.	
		0	No Interrupt Pending
		1	Interrupt Pending
5 (R/W1C)	UIAIS	Unimplemented Address Interrupt Status. The `CAN_GIS.UIAIS` bit indicates when there was a processor core access to an address that is not implemented in the CAN.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (R/W1C)	WUIS	Wake Up Interrupt Status. The `CAN_GIS.WUIS` bit indicates when the CAN has left the sleep mode because of detected activity on the CAN bus line.	
		0	No Interrupt Pending
		1	Interrupt Pending
3 (R/W1C)	BOIS	Bus Off Interrupt Status. The `CAN_GIS.BOIS` bit indicates when the CAN has entered the bus-off state. This interrupt source is active if the status of the CAN changes from normal operation mode to the bus-off mode. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and the bus-off mode is still active, this bit is not set again. If the module leaves the bus-off mode, the bit in `CAN_GIS` (and `CAN_GIF`) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 25-20: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	EPIS	Error Passive Interrupt Status. The `CAN_GIS.EPIS` bit indicates when the CAN has entered the error passive state. This interrupt source is active if the status of the CAN changes from the error active mode to the error passive mode. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and the error passive mode is still active, this bit is not set again. If the CAN leaves the error passive mode, the bit in `CAN_GIS` (and `CAN_GIF`) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (R/W1C)	EWRIS	Error Warning Receive Interrupt Status. The `CAN_GIS.EWRIS` bit indicates when the `CAN_CEC.RXECNT` has reached the warning limit. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in `CAN_GIS` (and `CAN_GIF`) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
0 (R/W1C)	EWTIS	Error Warning Transmit Interrupt Status. The `CAN_GIS.EWTIS` bit indicates when the `CAN_CEC.TXECNT` has reached the warning limit. If the bit in `CAN_GIS` (and `CAN_GIF`) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in `CAN_GIS` (and `CAN_GIF`) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Interrupt Pending Register

The `CAN_INT` register indicates the status of pending CAN interrupts and indicates the state of the `CAN_RX` and `CAN_TX` pins. Though this register is read-only, a write is allowed to exit the built-in sleep mode of the module on processors supporting this feature.

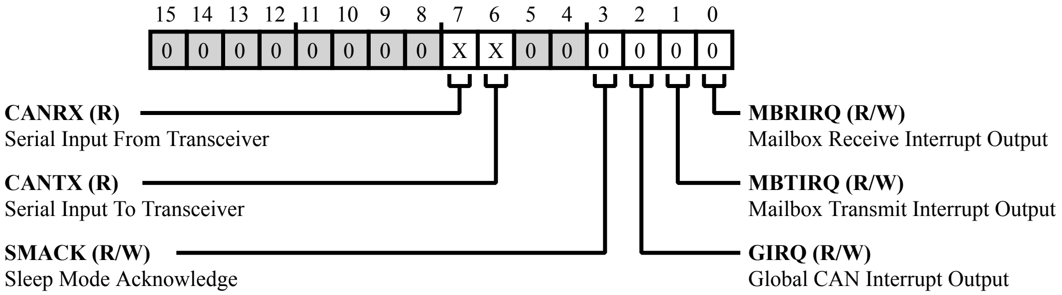

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 X X 0 0 0 0 0 0
CANRX (R)
Serial Input From Transceiver
CANTX (R)
Serial Input To Transceiver
SMACK (R/W)
Sleep Mode Acknowledge
MBRIRQ (R/W)
Mailbox Receive Interrupt Output
MBTIRQ (R/W)
Mailbox Transmit Interrupt Output
GIRQ (R/W)
Global CAN Interrupt Output

Figure 25-26: CAN_INT Register Diagram

Table 25-21: CAN_INT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	CANRX	Serial Input From Transceiver. The `CAN_INT.CANRX` bit indicates the logic value that the CAN detects on the `CAN_RX` pin. Note that the reset/default value for `CAN_INT.CANRX` is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
6 (R/NW)	CANTX	Serial Input To Transceiver. The `CAN_INT.CANTX` bit indicates the logic value that the CAN detects on the `CAN_TX` pin. Note that the reset/default value for `CAN_INT.CANTX` is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
3 (R/W)	SMACK	Sleep Mode Acknowledge. The `CAN_INT.SMACK` bit indicates when the CAN has entered sleep mode.	
		0	Not in Sleep Mode
		1	Sleep Mode

Table 25-21: CAN_INT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	GIRQ	Global CAN Interrupt Output. The `CAN_INT.GIRQ` bit indicates when at least one bit is set in the `CAN_GIF` register, indicating at least one unmasked CAN is flagged (latched). The `CAN_INT.GIRQ` bit remains set as long as at least one bit is set in the `CAN_GIF` register.	
		0	No CAN Global Interrupt Flag Set
		1	CAN Global Interrupt Flag (1 or More) Set
1 (R/W)	MBTIRQ	Mailbox Transmit Interrupt Output. The `CAN_INT.MBTIRQ` bit indicates when any bits are set in the `CAN_MBTIF1` register or `CAN_MBTIF2` register, indicating transmit.	
		0	No CAN Transmit Flags Set
		1	CAN Transmit Flags Set (1 or More)
0 (R/W)	MBRIRQ	Mailbox Receive Interrupt Output. The `CAN_INT.MBRIRQ` bit indicates when any bits are set in the `CAN_MBRIF1` register or `CAN_MBRIF2` register, indicating receive.	
		0	No CAN Receive Flags Set
		1	CAN Receive Flags Set (1 or More)

Mailbox Interrupt Mask 1 Register

The `CAN_MBIM1` register enables transmit and receive interrupt requests for mailboxes 0 through 15. Each bit in this register requests enables the transmit or receive interrupt request for the corresponding mailbox when set (=1).

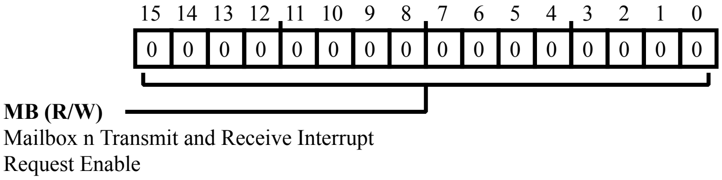

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Transmit and Receive Interrupt Request Enable

Figure 25-27: CAN_MBIM1 Register Diagram

Table 25-22: CAN_MBIM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Request Enable.

Mailbox Interrupt Mask 2 Register

The `CAN_MBIM2` register enables transmit and receive interrupt requests for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests enables the transmit or receive interrupt request for the corresponding mailbox when set (=1).

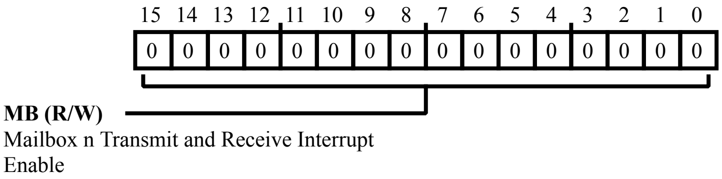

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Transmit and Receive Interrupt Enable

Figure 25-28: CAN_MBIM2 Register Diagram

Table 25-23: CAN_MBIM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Mailbox Receive Interrupt Flag 1 Register

The `CAN_MBRIF1` register indicates when a receive interrupt request is pending---due to successful reception (corresponding `CAN_RMP1` bit set) and the interrupt is enabled (corresponding `CAN_MBIM1` bit set)---for mailboxes 0 through 15. Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in `CAN_MBRIF1` is set, the CAN receive interrupt request is raised (`CAN_INT.MBRIRQ` bit set). To clear the interrupt request, all of the set bits in `CAN_RMP1` must be cleared by software, then the associated bits set in `CAN_MBRIF1` must be cleared (W1C).

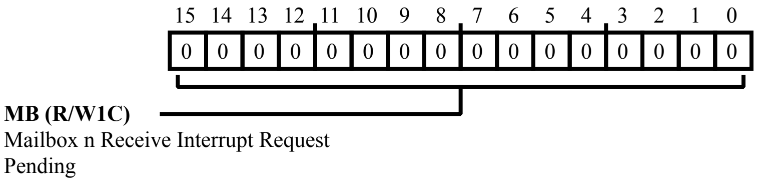


Figure 25-29: CAN_MBRIF1 Register Diagram

Table 25-24: CAN_MBRIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Receive Interrupt Request Pending.

Mailbox Receive Interrupt Flag 2 Register

The `CAN_MBRIF2` register indicates when a receive interrupt request is pending---due to successful reception (corresponding `CAN_RMP2` bit set) and the interrupt is enabled (corresponding `CAN_MBIM2` bit set)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in `CAN_MBRIF2` is set, the CAN receive interrupt request is raised (`CAN_INT.MBRIRQ` bit set). To clear the interrupt request, all of the set bits in `CAN_RMP2` must be cleared by software, then the associated bits set in `CAN_MBRIF2` must be cleared (W1C). Bits 8 through 15 are reserved and read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

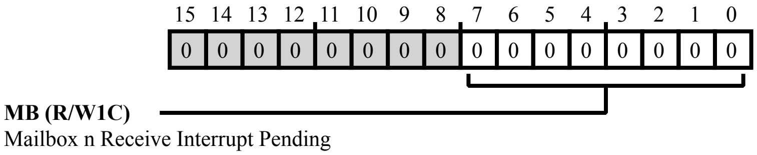


Figure 25-30: CAN_MBRIF2 Register Diagram

Table 25-25: CAN_MBRIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Temporary Mailbox Disable Register

The CAN_MBTD register supports temporarily and selectively disabling CAN mailboxes. For more information about this feature, see the Operating Modes section.

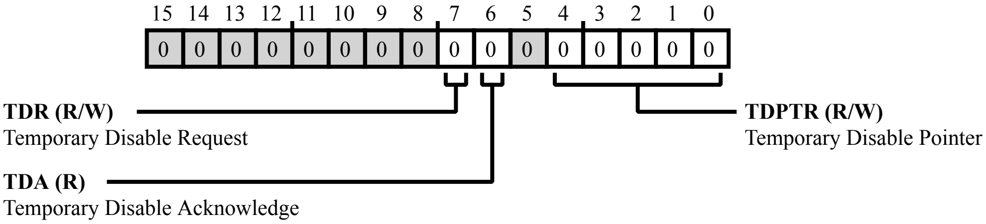

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
TDR (R/W)
Temporary Disable Request
TDA (R)
Temporary Disable Acknowledge
TDPTR (R/W)
Temporary Disable Pointer

Figure 25-31: CAN_MBTD Register Diagram

Table 25-26: CAN_MBTD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	TDR	Temporary Disable Request. The `CAN_MBTD.TDR` bit holds the pointer to mailbox, which is disabled when the `CAN_MBTD.TDR` bit is set.	
		0	No Request
		1	Request Temporary Mailbox Disable
6 (R/NW)	TDA	Temporary Disable Acknowledge. The `CAN_MBTD.TDA` bit indicates when the mailbox (to which the `CAN_MBTD.TDPTR` bit points) is disabled. When this bit is set for a mailbox, only the data field of that mailbox may be updated. Accesses that mailboxs control bits and the identifier are denied.	
		0	No Acknowledge
		1	Acknowledge Temporary Mailbox Disable
4:0 (R/W)	TDPTR	Temporary Disable Pointer. The `CAN_MBTD.TDPTR` bits hold the pointer to mailbox, which is disabled when the `CAN_MBTD.TDR` bit is set.	

Mailbox Transmit Interrupt Flag 1 Register

The `CAN_MBTIF1` register indicates when a transmit interrupt request is pending---due to successful transmission (corresponding `CAN_TA1` bit is set) and the interrupt is enabled (corresponding `CAN_MBIM1` bit is set)---for mailboxes 8 through 15. Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in `CAN_MBTIF1` is set, the CAN transmit interrupt request is raised (`CAN_INT.MBTIRQ` bit set). To clear the interrupt request, all of the set bits in `CAN_MBTIF1` must be cleared by software (W1C). Also, software must clear the associated bits set in `CAN_TA1` or set the associated bits in `CAN_TRS1` bit to clear the interrupt source asserting the bits in `CAN_MBTIF1`. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

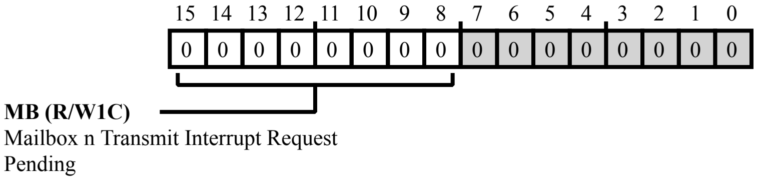


Figure 25-32: CAN_MBTIF1 Register Diagram

Table 25-27: CAN_MBTIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Interrupt Request Pending.

Mailbox Transmit Interrupt Flag 2 Register

The CAN_MBTIF2 register indicates when a transmit interrupt request is pending---due to successful transmission (corresponding CAN_TA2 bit is set) and the interrupt is enabled (corresponding CAN_MBIM2 bit is set)---for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF2 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit is set). To clear the interrupt request, all of the set bits in CAN_MBTIF2 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA2 or set the associated bits in CAN_TRS2 bit to clear the interrupt source asserting the bits in CAN_MBTIF2.

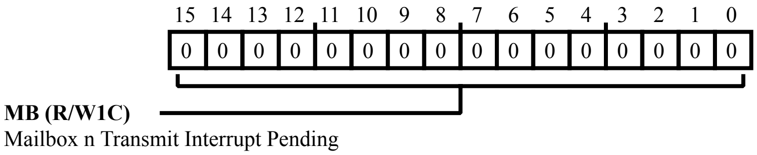


Figure 25-33: CAN_MBTIF2 Register Diagram

Table 25-28: CAN_MBTIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Word 0 Register

The `CAN_MB[nn]_DATA0` register holds mailbox data bytes.

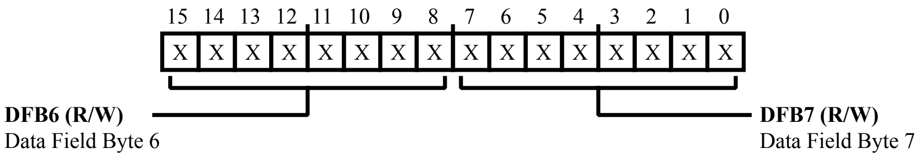

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
DFB6 (R/W)
Data Field Byte 6
DFB7 (R/W)
Data Field Byte 7

Figure 25-34: CAN_MB[nn]_DATA0 Register Diagram

Table 25-29: CAN_MB[nn]_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB6	Data Field Byte 6. The `CAN_MB[nn]_DATA0.DFB6` bits hold mailbox data.
7:0 (R/W)	DFB7	Data Field Byte 7. The `CAN_MB[nn]_DATA0.DFB7` bits hold mailbox data.

Mailbox Word 1 Register

The CAN_MB[nn]_DATA1 register holds mailbox data bytes.

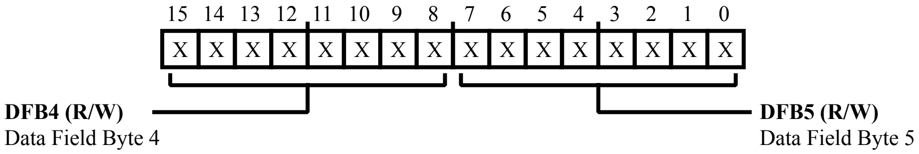

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
DFB4 (R/W)
Data Field Byte 4
DFB5 (R/W)
Data Field Byte 5

Figure 25-35: CAN_MB[nn]_DATA1 Register Diagram

Table 25-30: CAN_MB[nn]_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB4	Data Field Byte 4. The `CAN_MB[nn]_DATA1.DFB4` bits hold mailbox data.
7:0 (R/W)	DFB5	Data Field Byte 5. The `CAN_MB[nn]_DATA1.DFB5` bits hold mailbox data.

Mailbox Word 2 Register

The `CAN_MB[nn]_DATA2` register holds mailbox data bytes.

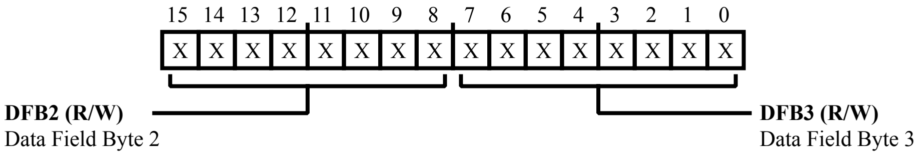

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
DFB2 (R/W)
Data Field Byte 2
DFB3 (R/W)
Data Field Byte 3

Figure 25-36: CAN_MB[nn]_DATA2 Register Diagram

Table 25-31: CAN_MB[nn]_DATA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB2	Data Field Byte 2. The `CAN_MB[nn]_DATA2.DFB2` bits hold mailbox data.
7:0 (R/W)	DFB3	Data Field Byte 3. The `CAN_MB[nn]_DATA2.DFB3` bits hold mailbox data.

Mailbox Word 3 Register

The `CAN_MB[nn]_DATA3` register holds mailbox data bytes.

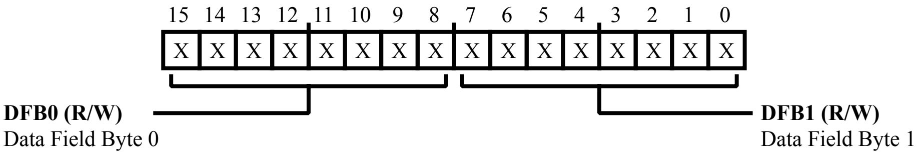

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
DFB0 (R/W)
Data Field Byte 0
DFB1 (R/W)
Data Field Byte 1

Figure 25-37: CAN_MB[nn]_DATA3 Register Diagram

Table 25-32: CAN_MB[nn]_DATA3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB0	Data Field Byte 0. The `CAN_MB[nn]_DATA3.DFB0` bits hold mailbox data.
7:0 (R/W)	DFB1	Data Field Byte 1. The `CAN_MB[nn]_DATA3.DFB1` bits hold mailbox data.

Mailbox ID 0 Register

The CAN_MB[nn]_ID0 register contains the lower 16 bits of the 18-bit extended identifier.

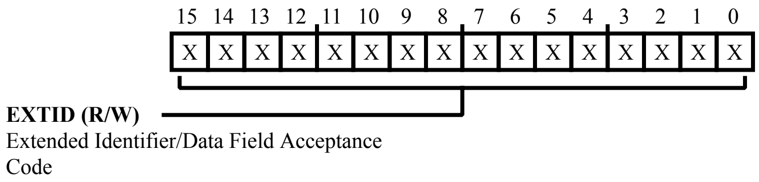

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
EXTID (R/W)
Extended Identifier/Data Field Acceptance Code

Figure 25-38: CAN_MB[nn]_ID0 Register Diagram

Table 25-33: CAN_MB[nn]_ID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Acceptance Code. The `CAN_MB[nn]_ID0.EXTID` bits hold the lower 16 bits of the 18-bit extended ID.

Mailbox ID 1 Register

The `CAN_MB[nn]_ID1` register contains the identifier bits of mailbox. The 11-bit BASE_ID is mapped to The `CAN_MB[nn]_ID1.BASEID` field. It also enables the extended identification and contains upper two bits of 18-bit extended identifier.

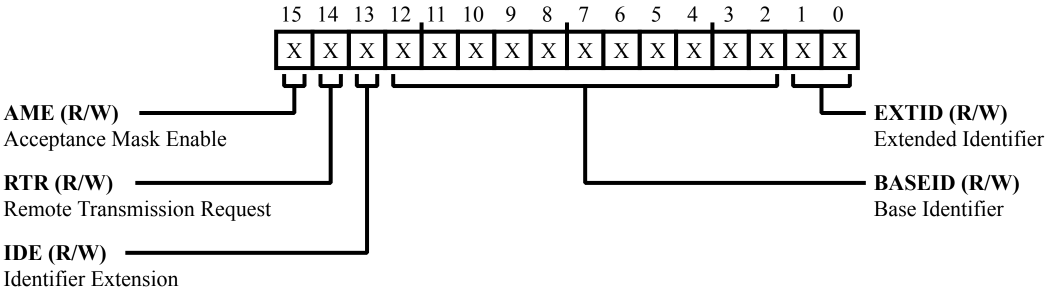

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
AME (R/W)
Acceptance Mask Enable
RTR (R/W)
Remote Transmission Request
IDE (R/W)
Identifier Extension
EXTID (R/W)
Extended Identifier
BASEID (R/W)
Base Identifier

Figure 25-39: CAN_MB[nn]_ID1 Register Diagram

Table 25-34: CAN_MB[nn]_ID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	AME	Acceptance Mask Enable. The `CAN_MB[nn]_ID1.AME` bit enables acceptance mask operations if the mailbox is configured as receiver. When enabled (=1), only those bits that have the corresponding mask bit cleared are compared to the received message ID. A bit position that is set in the mask register does not need to match. This bit should be set to 0 when the mailbox is configured in transmit mode.
14 (R/W)	RTR	Remote Transmission Request. The `CAN_MB[nn]_ID1.RTR` bit selects whether the frame contains data (data frame) or contains a request for data associated with the message identifier in the frame being sent (remote frame).
13 (R/W)	IDE	Identifier Extension. The `CAN_MB[nn]_ID1.IDE` bit enables the comparison of the received message ID to the value in the `CAN_MB[nn]_ID1.EXTID` and `CAN_MB[nn]_ID0.EXTID` bits. When configured as transmitter, it sends the extended identifier in addition to the base identifier.
12:2 (R/W)	BASEID	Base Identifier. The `CAN_MB[nn]_ID1.BASEID` bits hold the base identifier for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The `CAN_MB[nn]_ID1.EXTID` bits hold the upper two bits of 18-bit extended identifier.

Mailbox Length Register

The `CAN_MB[nn]_LENGTH` register holds the data length code for the received remote frame. For more information about remote frames, see the Remote Frame Handling section.

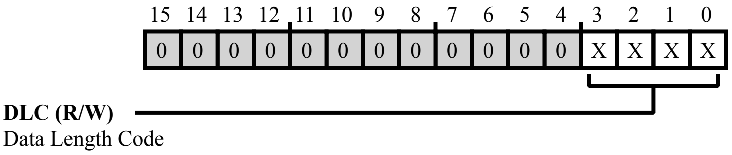

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 X X X X
DLC (R/W)
Data Length Code

Figure 25-40: CAN_MB[nn]_LENGTH Register Diagram

Table 25-35: CAN_MB[nn]_LENGTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	DLC	Data Length Code. The `CAN_MB[nn]_LENGTH.DLC` bits hold the DLC value of the received remote frame. The received value overwrites any previous value.

Mailbox Time Stamp Register

The `CAN_MB[nn]_TIMESTAMP` register holds an indication of the time of reception or transmission for each message, when the universal counter is in time stamp mode (`CAN_UCCNF.UCCNF` =0x1). In this mode, the CAN writes the value of the counter (`CAN_UCCNT`) to the `CAN_MB[nn]_TIMESTAMP` register when a received message is stored or a message is transmitted. For more information about time stamps, see the Time Stamps section.

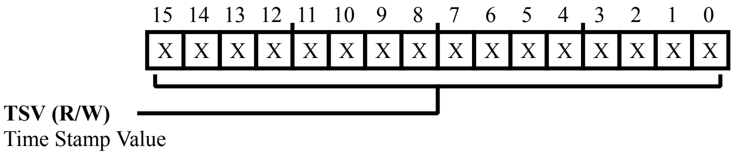


Figure 25-41: CAN_MB[nn]_TIMESTAMP Register Diagram

Table 25-36: CAN_MB[nn]_TIMESTAMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSV	Time Stamp Value. The `CAN_MB[nn]_TIMESTAMP.TSV` bits hold the message time stamp value.

Mailbox Configuration 1 Register

The `CAN_MC1` register enables mailboxes 0 through 15. Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the `CAN_TRS1` bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated `CAN_TRS1` bit is reset by the internal logic can cause unpredictable results.

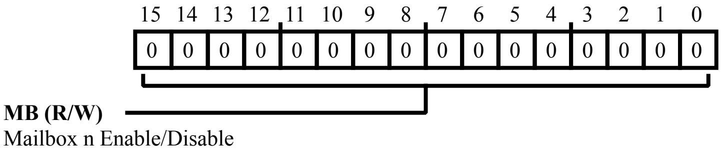


Figure 25-42: CAN_MC1 Register Diagram

Table 25-37: CAN_MC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Configuration 2 Register

The `CAN_MC2` register enables mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the `CAN_TRS2` bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated `CAN_TRS2` bit is reset by the internal logic can cause unpredictable results.

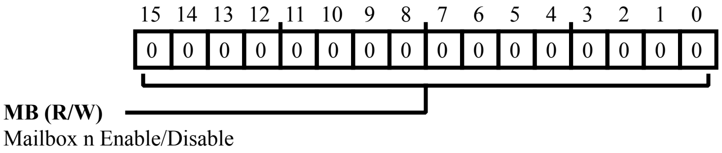


Figure 25-43: CAN_MC2 Register Diagram

Table 25-38: CAN_MC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Direction 1 Register

The `CAN_MD1` register selects the data transfer direction for mailboxes 0 through 15. Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

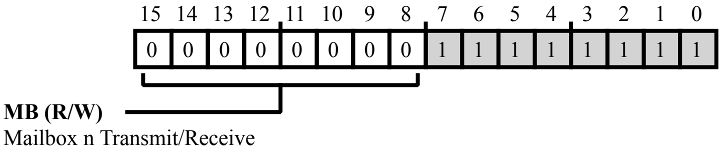

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
MB (R/W)
Mailbox n Transmit/Receive

Figure 25-44: CAN_MD1 Register Diagram

Table 25-39: CAN_MD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit/Receive.

Mailbox Direction 2 Register

The `CAN_MD2` register selects the data transfer direction for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 8 through 15 are read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

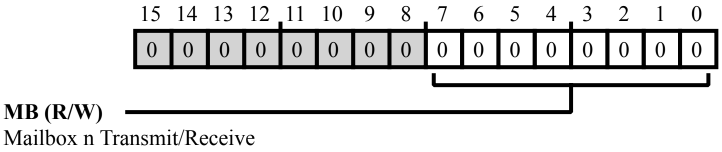

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Transmit/Receive

Figure 25-45: CAN_MD2 Register Diagram

Table 25-40: CAN_MD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Transmit/Receive.

Overwrite Protection/Single Shot Transmission 1 Register

The `CAN_OPSS1` register enables overwrite protection for mailboxes 0 through 15. Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

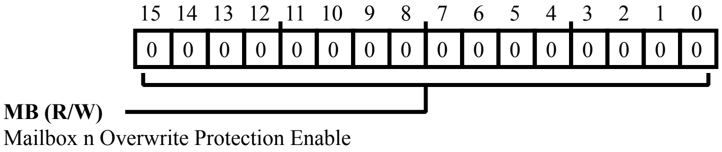


Figure 25-46: CAN_OPSS1 Register Diagram

Table 25-41: CAN_OPSS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Overwrite Protection/Single Shot Transmission 2 Register

The `CAN_OPSS2` register enables overwrite protection for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

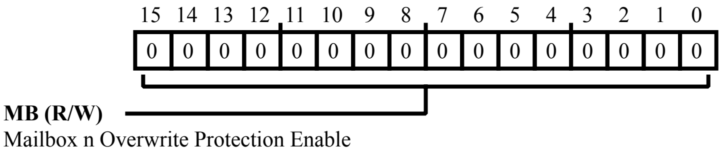


Figure 25-47: CAN_OPSS2 Register Diagram

Table 25-42: CAN_OPSS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Remote Frame Handling 1 Register

The `CAN_RFH1` register enables remote frame handling for mailboxes 8 through 15. Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

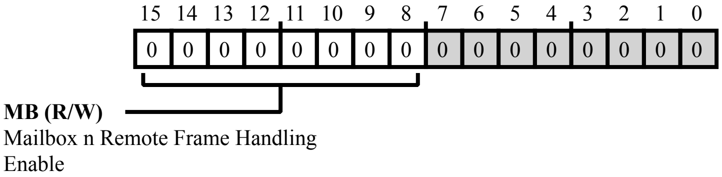

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Remote Frame Handling Enable

Figure 25-48: CAN_RFH1 Register Diagram

Table 25-43: CAN_RFH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Remote Frame Handling 2 Register

The `CAN_RFH2` register enables remote frame handling for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations.

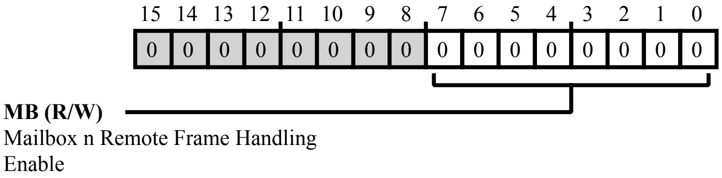

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Remote Frame Handling Enable

Figure 25-49: CAN_RFH2 Register Diagram

Table 25-44: CAN_RFH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Receive Message Lost 1 Register

The `CAN_RML1` register indicates when a message is lost---due to a message coming while there is pending data (corresponding `CAN_RMP1` bit set) and overwrite protection is disabled (`CAN_OPSS1` bit cleared)---for mailboxes 0 through 15. Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1).

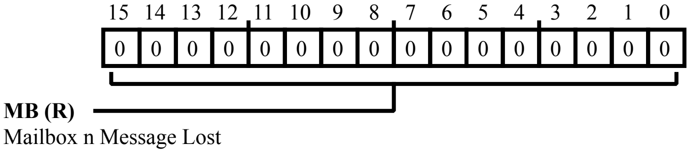

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R)
Mailbox n Message Lost

Figure 25-50: CAN_RML1 Register Diagram

Table 25-45: CAN_RML1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Lost 2 Register

The `CAN_RML2` register indicates when a message is lost---due to a message coming while there is pending data (corresponding `CAN_RMP2` bit set) and overwrite protection is disabled (`CAN_OPSS2` bit cleared)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

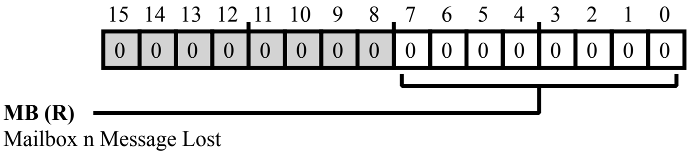


Figure 25-51: CAN_RML2 Register Diagram

Table 25-46: CAN_RML2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Pending 1 Register

The `CAN_RMP1` register indicates when a message is pending (unread data) for mailboxes 0 through 15. Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1).

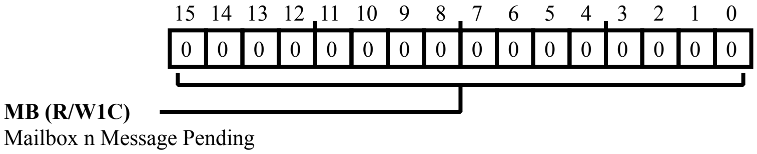

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Message Pending

Figure 25-52: CAN_RMP1 Register Diagram

Table 25-47: CAN_RMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Message Pending.

Receive Message Pending 2 Register

The `CAN_RMP2` register indicates when a message is pending (unread data) for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

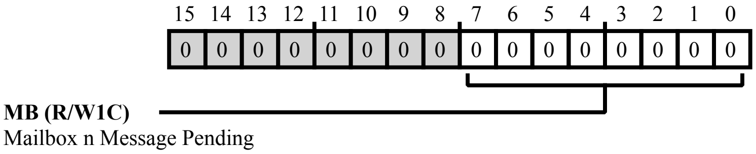

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Message Pending

Figure 25-53: CAN_RMP2 Register Diagram

Table 25-48: CAN_RMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Message Pending.

Status Register

The CAN_STAT register indicates status for CAN modes and error conditions.

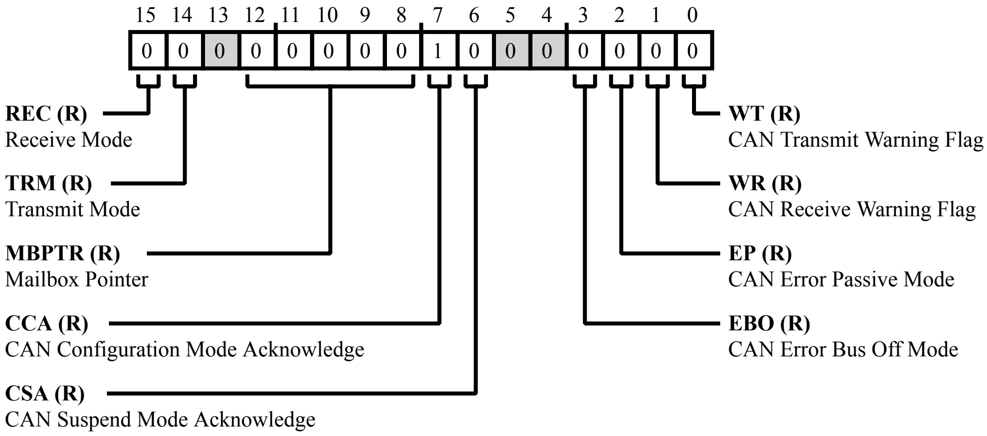

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
REC (R)
Receive Mode
TRM (R)
Transmit Mode
MBPTR (R)
Mailbox Pointer
CCA (R)
CAN Configuration Mode Acknowledge
CSA (R)
CAN Suspend Mode Acknowledge
WT (R)
CAN Transmit Warning Flag
WR (R)
CAN Receive Warning Flag
EP (R)
CAN Error Passive Mode
EBO (R)
CAN Error Bus Off Mode

Figure 25-54: CAN_STAT Register Diagram

Table 25-49: CAN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	REC	Receive Mode. The `CAN_STAT.REC` bit indicates whether the CAN is in receive mode.	
		0	Not in Receive Mode
		1	Receive Mode
14 (R/NW)	TRM	Transmit Mode. The `CAN_STAT.TRM` bit indicates whether the CAN is in transmit mode.	
		0	Not in Transmit Mode
		1	Transmit Mode
12:8 (R/NW)	MBPTR	Mailbox Pointer. The `CAN_STAT.MBPTR` bits represent the mailbox number of the current transmit message. After a successful transmission, these bits remain unchanged.	
		0-31	Processing Mailbox 0 to31 Message
7 (R/NW)	CCA	CAN Configuration Mode Acknowledge. The `CAN_STAT.CCA` bit indicates whether the CAN is in configuration mode.	
		0	Not in Configuration Mode
		1	Configuration mode

Table 25-49: CAN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	CSA	CAN Suspend Mode Acknowledge. The `CAN_STAT.CSA` bit indicates whether the CAN is in suspend mode.	
		0	Not in Suspend Mode
		1	Suspend mode
3 (R/NW)	EBO	CAN Error Bus Off Mode. The `CAN_STAT.EBO` bit indicates whether the CAN is in error bus off mode.	
		0	TXECNT Below 256
		1	TXECNT Above Bus Off Limit
2 (R/NW)	EP	CAN Error Passive Mode. The `CAN_STAT.EP` bit indicates whether the CAN is in error passive mode.	
		0	TXECNT and RXECNT Below 128
		1	TXECNT or RXECNT Above EP Level
1 (R/NW)	WR	CAN Receive Warning Flag. The `CAN_STAT.WR` bit indicates whether the CAN has detected a receive warning flag condition.	
		0	RXECNT Below Limit
		1	RXECNT at Limit
0 (R/NW)	WT	CAN Transmit Warning Flag. The `CAN_STAT.WT` bit indicates whether the CAN detected a transmit warning flag condition.	
		0	TXECNT Below Limit
		1	TXECNT at Limit

Transmission Acknowledge 1 Register

The CAN_TA1 register indicates transmission success for mailboxes 8 through 15. Each bit in this register indicates transmission success for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

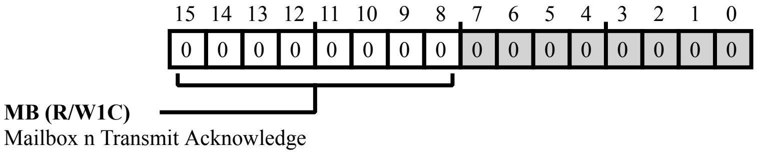

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Transmit Acknowledge

Figure 25-55: CAN_TA1 Register Diagram

Table 25-50: CAN_TA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Transmission Acknowledge 2 Register

The `CAN_TA2` register indicates transmission success for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission success for the corresponding mailbox when set (=1).

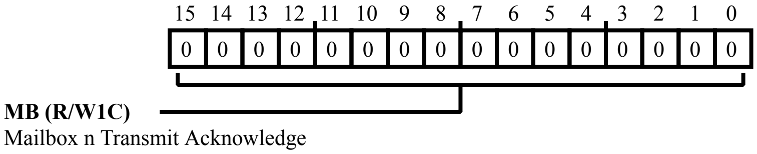

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W1C)
Mailbox n Transmit Acknowledge

Figure 25-56: CAN_TA2 Register Diagram

Table 25-51: CAN_TA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Timing Register

The CAN_TIMING register select the time segments, sampling, and synchronization for CAN bit timing. For more information about bit timing and clock operation, see the CAN Operating Modes section.

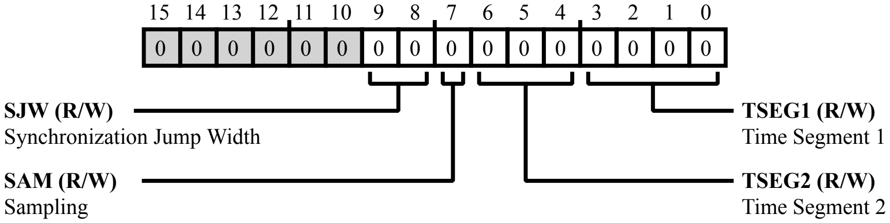


Figure 25-57: CAN_TIMING Register Diagram

Table 25-52: CAN_TIMING Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	SJW	Synchronization Jump Width. The CAN_TIMING.SJW bits select the maximum number of time quanta, ranging from 1 to 4(SJW + 1). This selection allows for a re-synchronization attempt when the CAN detects a recessive-to-dominant edge outside the synchronization segment. The re-synchronization automatically moves the sampling point such that the CAN bit is still handled properly. Note that the CAN_TIMING.SJW value should not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1.
7 (R/W)	SAM	Sampling. The CAN_TIMING.SAM bit selects whether the CAN performs normal sampling (once at the sampling point described by the CAN_TIMING register) or performs over sampling. If CAN_TIMING.SAM is set, the CAN over samples the input signal at three times at the CDU0_CLKO4 rate. The resulting value is generated by a majority decision of the three sample values. Note that the CAN_TIMING.SAM bit should always be cleared if the CAN_CLK.BRP value is less than 4.
6:4 (R/W)	TSEG2	Time Segment 2. The CAN_TIMING.TSEG2 bits and CAN_TIMING.TSEG1 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.
3:0 (R/W)	TSEG1	Time Segment 1. The CAN_TIMING.TSEG1 bits and CAN_TIMING.TSEG2 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.

Transmission Request Reset 1 Register

The `CAN_TRR1` register requests transmit abort for mailboxes 8 through 15. Bits in this register request transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (`CAN_TRS1`) and in the `CAN_TRR1` are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

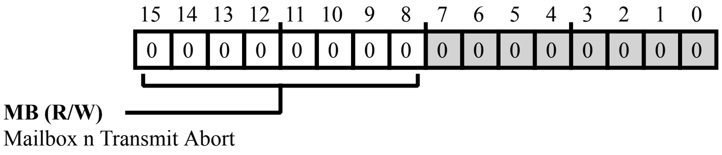

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Transmit Abort

Figure 25-58: CAN_TRR1 Register Diagram

Table 25-53: CAN_TRR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Reset 2 Register

The `CAN_TRR2` register requests transmit abort for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (`CAN_TRS2`) and in the `CAN_TRR2` are cleared.

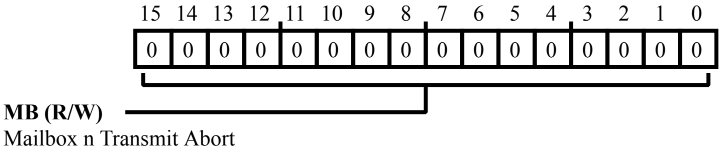

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MB (R/W)
Mailbox n Transmit Abort

Figure 25-59: CAN_TRR2 Register Diagram

Table 25-54: CAN_TRR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Set 1 Register

The `CAN_TRS1` register requests transmit for mailboxes 8 through 15. Bits in this register request transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in `CAN_MC1` = 1), and (subsequently) the corresponding transmit request bit is set (in `CAN_TRS1`). When a transmission completes, the corresponding bits in `CAN_TRS1` and in the transmit request reset register (`CAN_TRR1`) are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

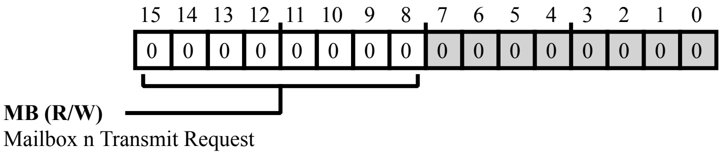


Figure 25-60: CAN_TRS1 Register Diagram

Table 25-55: CAN_TRS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Request.

Transmission Request Set 2 Register

The CAN_TRS2 register requests transmit for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC2 = 1), and (subsequently) the corresponding transmit request bit is set (in CAN_TRS2). When a transmission completes, the corresponding bits in CAN_TRS2 and in the transmit request reset register (CAN_TRR2) are cleared.

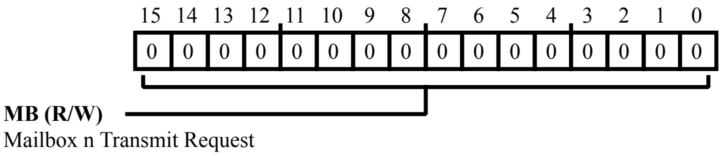


Figure 25-61: CAN_TRS2 Register Diagram

Table 25-56: CAN_TRS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Request.

Universal Counter Configuration Mode Register

The CAN_UCCNF register controls the operation of the universal counter, including counter enable and counter mode selection.

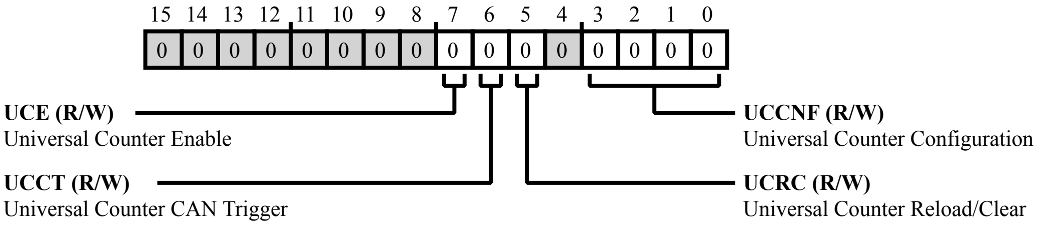

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
UCE (R/W)
Universal Counter Enable
UCCT (R/W)
Universal Counter CAN Trigger
UCCNF (R/W)
Universal Counter Configuration
UCRC (R/W)
Universal Counter Reload/Clear

Figure 25-62: CAN_UCCNF Register Diagram

Table 25-57: CAN_UCCNF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	UCE	Universal Counter Enable. The `CAN_UCCNF.UCE` bit enables universal counter operation in the mode selected by the `CAN_UCCNF.UCCNF` bits.	
		0	Disable Counter
		1	Enable Counter
6 (R/W)	UCCT	Universal Counter CAN Trigger. The `CAN_UCCNF.UCCT` bit enables the universal counter trigger, directing the CAN to re-load the counter on mailbox 4 reception in watchdog mode and clear the counter on mailbox 4 reception in time stamp mode. This bit has no effect in all other modes.	
		0	Disable Trigger
		1	Enable Trigger
5 (R/W)	UCRC	Universal Counter Reload/Clear. The `CAN_UCCNF.UCRC` bit re-loads or clears the universal counter, depending on the counter mode. In watchdog mode, setting this bit directs the CAN to re-load the counter. In all other modes, setting this bit directs the CAN to clear the counter.	
		0	No Action
		1	Re-load or Clear the Counter
3:0 (R/W)	UCCNF	Universal Counter Configuration. The `CAN_UCCNF.UCCNF` bits select the universal counter operating mode. For more information about these modes, see the Operating Modes section.	
		0	Reserved
		1	Time Stamp Mode
		2	Watchdog Mode

Table 25-57: CAN_UCCNF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Auto-transmit Mode
		4	Reserved
		5	Reserved
		6	Count Error Frames
		7	Count Overload Frames
		8	Count Arbitration Lost
		9	Count Aborted Transmissions
		10	Count Successful Transmissions
		11	Count Rejected Receive Messages
		12	Count Receive Message Lost
		13	Count Successful Receptions
		14	Count Stored Receptions
		15	Count Valid Messages

Universal Counter Register

The `CAN_UCCNT` register holds the current universal count. This register is reloaded from the `CAN_UCRC` register when counter the decrements to zero in auto-transmit mode.

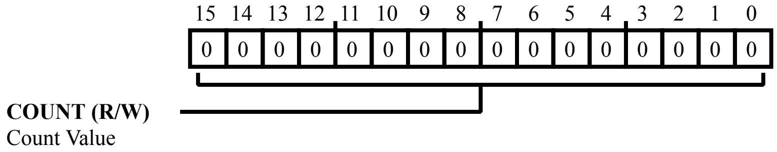

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT (R/W)
Count Value

Figure 25-63: CAN_UCCNT Register Diagram

Table 25-58: CAN_UCCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count Value. The `CAN_UCCNT.COUNT` bits hold the current universal count value.

Universal Counter Reload/Capture Register

The `CAN_UCRC` register holds the period value (universal count), which is used in auto-transmit mode as the period for sending the message in mailbox 11 (broadcast heartbeat) to all CAN nodes. Accordingly, messages sent this way usually have high priority.

The period value is written to the `CAN_UCRC` register. When auto-transmit mode is enabled (`CAN_UCCNF.UCCNF` = 0x3), the CAN loads the counter with the value in `CAN_UCRC`. The counter decrements to 0 at the CAN bit clock rate, then is reloaded. Each time the counter decrements to 0, the CAN sets the `CAN_TRS1.MB` bit for mailbox 11 and sends the corresponding message from mailbox 11.

Note that for auto-transmit mode, mailbox 11 must be configured as a transmit mailbox and must contain valid data (identifier, control bits, and data). This setup must occur before the counter first expires after this mode is enabled.

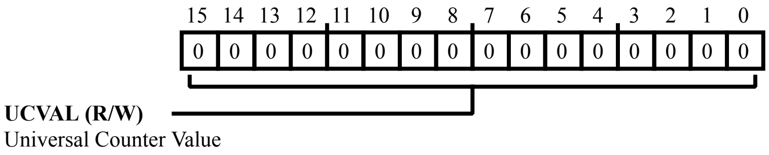


Figure 25-64: CAN_UCRC Register Diagram

Table 25-59: CAN_UCRC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	UCVAL	Universal Counter Value. The `CAN_UCRC.UCVAL` bits hold the value for the universal count period, which is used in auto-transmit mode.

26 Mobile Storage Interface (MSI)

The mobile storage interface (MSI) is a fast, synchronous controller that uses various protocols to communicate with MMC, SD, and SDIO cards. It addresses the growing storage need in embedded systems, handheld, and consumer electronics applications that require low power. The MSI is compatible with the following protocols.

- MMC (Multimedia Card) bus protocol
- SD (Secure Digital) bus protocol
- SDIO (Secure Digital Input Output) bus protocol

All of these storage solutions use similar interface protocols. The main difference between MMC and SD support is the initialization sequence. The main difference between SD and SDIO support is the use of interrupt and read wait signals for SDIO.

NOTE: The MSI does not support the SPI bus protocol.

MSI Features

The MSI includes the following features.

- Supports Secure Digital memory protocol (version 3.0)
- Supports Secure Digital I/O protocol (version 3.0)
- Supports Multimedia Card protocol (MMC version 4.41, eMMC version 4.5)
- Support for a single SD or SDIO card
- Support for a single MMC device (removable or embedded)
- Support for 1-bit and 4-bit SD modes (SPI mode is not supported)
- Support for 1-, 4-, and 8-bit MMC modes (SPI mode is not supported)
- Supports MMC boot operation
- Supports SDIO interrupts
- Supports command completion signal and interrupt to host processor

- Supports CRC generation and error detection
- 1024-byte transmit/receive FIFO
- Integrated DMA controller
- Card detection capabilities
- Programmable clock frequency
- Supports power management and clock control.

MSI Functional Description

This section provides information on the function of the MSI module.

MMC booting

Normal boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards and is performed in push-pull mode. Also alternate Boot Operation; removable MMC4.3 card.

CRC generation and error detection

Cyclic redundancy codes (CRC) are used to protect commands, responses, and data transfers from transmission errors.

FIFO controller

Interfaces the internal FIFO to the host or DMA interface and the card controller unit. The FIFO depth is configured for 1024 bytes

Integrated DMA controller

Contains a single transmit or receive engine, which transfers data from host memory to the device port and conversely. The controller uses a descriptor to move data efficiently from source to destination with minimal core intervention

Power management and clock control

A low-power mode is available. A clock control block provides the clock frequencies required for SD/MMC cards

ADSP-SC58x MSI Register List

The Mobile Storage Interface (MSI) supports access to mobile memory and devices. A set of registers governs MSI operations. For more information on MSI functionality, see the MSI register descriptions.

Table 26-1: ADSP-SC58x MSI Register List

Name	Description
MSI_BLKSIZ	Block Size Register
MSI_BUFADDR	Current Buffer Descriptor Address Register
MSI_BUSMODE	Bus Mode Register
MSI_BYTCNT	Byte Count Register
MSI_CDETECT	Card Detect Register
MSI_CDTHRCTL	Card Threshold Control Register
MSI_CLKDIV	Clock Divider Register
MSI_CLKEN	Clock Enable Register
MSI_CMD	Command Register
MSI_CMDARG	Command Argument Register
MSI_CTL	Control Register
MSI_CTYPE	Card Type Register
MSI_DBADDR	Descriptor List Base Address Register
MSI_DEBNCE	Debounce Count Register
MSI_DSCADDR	Current Host Descriptor Address Register
MSI_ENSHIFT	Enable Phase Shift Register
MSI_FIFOTH	FIFO Threshold Watermark Register
MSI_IDINTEN	Internal DMA Interrupt Enable Register
MSI_IDSTS	Internal DMA Status Register
MSI_IMSK	Interrupt Mask Register
MSI_ISTAT	Raw Interrupt Status Register
MSI_MSKISTAT	Masked Interrupt Status Register
MSI_PLDMND	Poll Demand Register
MSI_RESP0	Response Register 0
MSI_RESP1	Response Register 1
MSI_RESP2	Response Register 2
MSI_RESP3	Response Register 3
MSI_STAT	Status Register
MSI_TBBCNT	Transferred Host to BIU-FIFO Byte Count Register
MSI_TCBCNT	Transferred CIU Card Byte Count Register
MSI_TMOUT	Timeout Register

ADSP-SC58x MSI Interrupt List

Table 26-2: ADSP-SC58x MSI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
131	MSI0_STAT	MSI0 Status	Level	

ADSP-SC58x MSI Trigger List

Table 26-3: ADSP-SC58x MSI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
136	`MSI0_DONE`	MSI0 Transfer Done	Level

Table 26-4: ADSP-SC58x MSI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

MSI Block Diagram

The ***MSI Block Diagram*** shows the functional blocks within the MSI. These blocks are described in more detail in the MSI Architectural Concepts section.

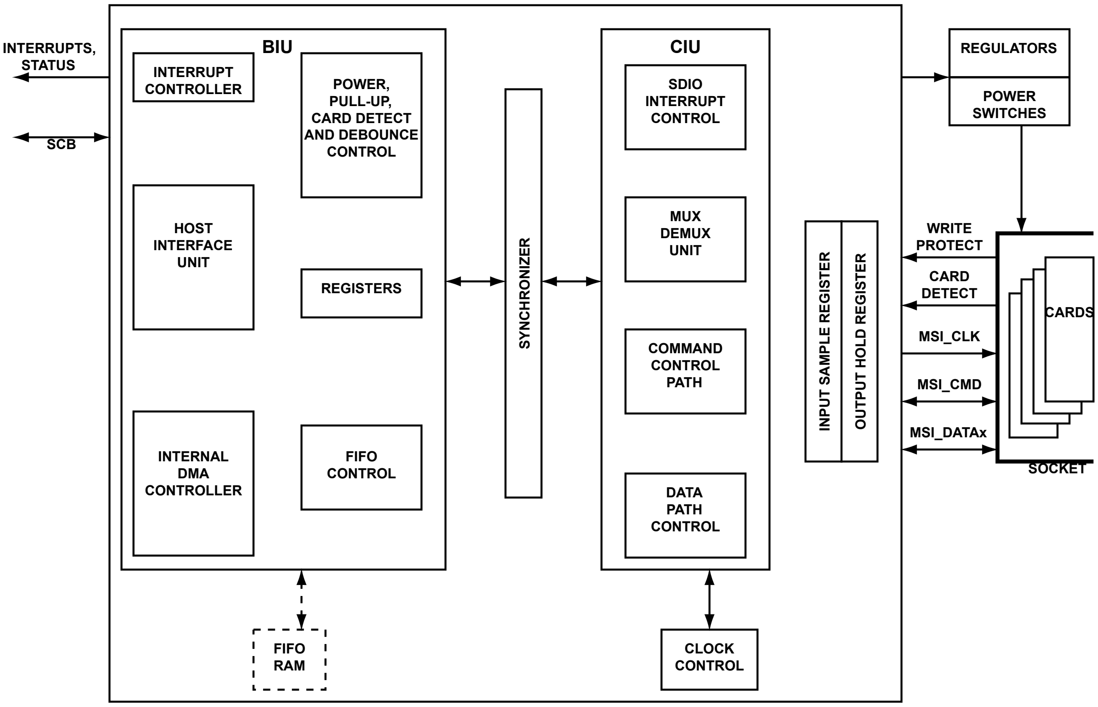

INTERRUPTS, STATUS
SCB
BIU
INTERRUPT CONTROLLER
POWER, PULL-UP, CARD DETECT AND DEBOUNCE CONTROL
HOST INTERFACE UNIT
REGISTERS
INTERNAL DMA CONTROLLER
FIFO CONTROL
SYNCHRONIZER
CIU
SDIO INTERRUPT CONTROL
MUX DEMUX UNIT
COMMAND CONTROL PATH
DATA PATH CONTROL
INPUT SAMPLE REGISTER
OUTPUT HOLD REGISTER
REGULATORS
POWER SWITCHES
WRITE PROTECT
CARD DETECT
MSI_CLK
MSI_CMD
MSI_DATAx
CARDS
SOCKET
FIFO RAM
CLOCK CONTROL

Figure 26-1: MSI Block Diagram

NOTE: The card-detect and write-protect signals are from the SD/MMC card socket and not from the SD/MMC card.

MSI Architectural Concepts

The following sections describe the functions and features of the MSI controller as well as the MMC, SD, and SDIO protocols.

Communication is through a master and slave configuration, where the MSI is the master device and the card is the slave device. The MSI communicates with the device using a message-based bus protocol in which the host sends commands serially using the `MSI_CMD` signal. Some commands require that the card provide a response back to the host. This response is also sent serially on the `MSI_CMD` signal.

Data transfers, both to and from the card, occur using the data signals. The number of data lines used for the data transfer can be configured to 1 (`MSI_D0`), 4 (`MSI_D3`–`MSI_D0`), or 8 (`MSI_D7`–`MSI_D0`). All `MSI_CMD` and `MSI_D7`–`MSI_D0` transfers are synchronous with `MSI_CLK`. Cyclic redundancy codes (CRC) are used to protect commands, responses, and data transfers from transmission errors. A CRC7 code is generated for every command sent by the host and for almost every response returned by the card on the `MSI_CMD` signal.

The MSI architecture can be described in terms of its submodules. The primary parts of the architecture are:

- Bus Interface Unit (BIU). This unit provides the host interface to the registers through the Host Interface Unit (HIU). Additionally, it provides independent data FIFO access through a DMA interface.

- Internal Direct Memory Access Controller (IDMAC). This unit is responsible for exchanging data between the FIFO and the host memory. A set of IDMAC registers is accessible to the host for controlling the IDMAC operation.
- Card Interface Unit (CIU). This unit controls the card-specific protocols. Within the CIU, the command path control unit and datapath control unit interface the controller to the command and data ports of the SD/MMC/SDIO cards. The CIU also provides clock control.

Bus Interface Unit (BIU)

The BIU provides the following functions:

- Host interface
- Interrupt control
- Register access
- FIFO access
- Power and pull-up control and card detection

Host Interface Unit (HIU)

The Host Interface Unit (HIU) is a slave bus interface, which provides the interface between the MSI and the processor system bus. It supports the burst accesses to the data FIFO address region only. The register address region is accessed through the standard accesses.

Interrupt Controller Unit

The interrupt controller unit generates an interrupt that depends on the controller interrupt status, the interrupt-mask register, and the global interrupt-enable register bit. Once an interrupt condition is detected, the software sets the corresponding interrupt bit in the interrupt status register. The interrupt status bit remains set until the software clears it by writing a 1 to the interrupt bit (a 0 leaves the bit unchanged).

NOTE: Before enabling the interrupt, programs must write 32'hFFFF_FFFF to the interrupt status register (`MSI_ISTAT`) in order to clear any pending unserviced interrupts. When clearing interrupts during normal operation, only clear the interrupt bits that are serviced.

Register Unit

The register unit is part of the Bus Interface Unit (BIU). It provides read and write access to the registers.

All registers reside in the BIU clock domain. When a command is sent to a card by setting the `MSI_CMD.STARTCMD` bit, all relevant registers needed for the CIU operation are transferred to the CIU block. (The `MSI_CMD.STARTCMD` bit is bit[31] of the `MSI_CMD` register). During this time, software must not write to the registers that are transferred from the BIU to the CIU. The software must wait for the hardware to clear the start bit before writing to these registers again. The register unit has a hardware locking feature to prevent illegal writes to registers.

Once a command start is issued by setting the `MSI_CMD.STARTCMD` bit, the following registers cannot be reprogrammed until the Card Interface Unit (CIU) accepts the command:

- `MSI_CMD` – Command
- `MSI_CMDARG` – Command Argument
- `MSI_BYTCNT` – Byte Count
- `MSI_BLKSIZ` – Block Size
- `MSI_CLKDIV` – Clock Divider
- `MSI_CLKEN` – Clock Enable
- `MSI_TMOUT` – Timeout
- `MSI_CTYPE` – Card Type

The hardware resets the `MSI_CMD.STARTCMD` bit once the CIU accepts the command. If a host writes to any of these registers during this locked time, then the write is ignored and the hardware lock error bit is set in the `MSI_ISTAT` (status) register. Additionally, if the interrupt is enabled and not masked for a hardware lock error, then an interrupt is sent to the host.

Once a command is accepted, software can send another command to the CIU which has a one-deep command queue under the following conditions:

- If the previous command was not a data transfer command, the new command is sent to the card once the previous command completes.
- If the previous command is a data transfer command and if the `MSI_CMD.WTPRIVDATA` bit is set for the new command, the new command is sent to the card only when the data transfer completes.
- If the `MSI_CMD.WTPRIVDATA` is 0, then the new command is sent to the card as soon as the previous command is sent. Typically, software uses this option only to stop or abort a previous data transfer or query the card status in the middle of a data transfer.

FIFO Controller Unit

The FIFO controller interfaces the internal FIFO to the host or DMA interface and the card controller unit. The FIFO depth is configured for 1024 bytes. A single shared FIFO is used for read and write operations because read and write transfers to the cards do not occur simultaneously.

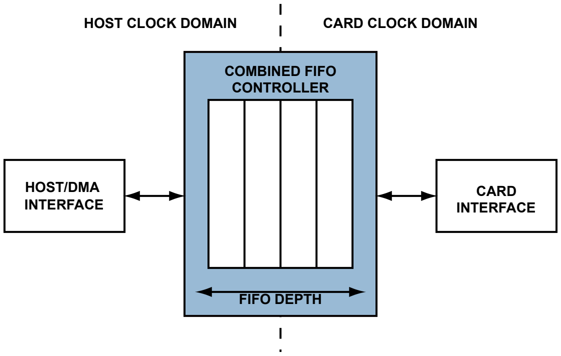


Figure 26-2: Combined Transmit and Receive FIFO

Power and Pull-up Control and Card Detection Unit

The card detection unit looks for any changes in the card-detect signal for card insertion or card removal. The unit filters out the debounce associated with mechanical insertion or removal, and generates one interrupt to the host. The debounce filter value is programmed through the `MSI_DEBNCE` register.

On power-on, the controller reads in the `MSI_CDETECT` register and stores the value in the memory. Upon receiving a card-detect interrupt, the controller again reads the `MSI_CDETECT` register to decide whether card was removed or inserted. The ***Card-Detect Signals*** figure shows the timing for the card-detect signal.

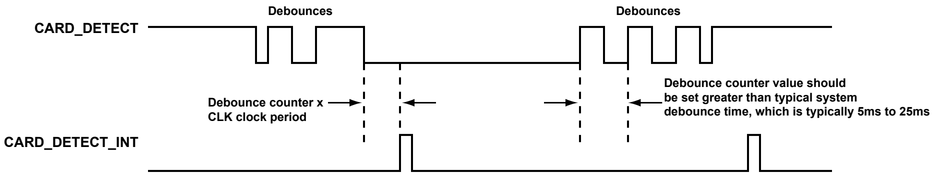

CARD_DETECT
Debounces
Debounces
Debounce counter x
CLK clock period
Debounce counter value should
be set greater than typical system
debounce time, which is typically 5ms to 25ms
CARD_DETECT_INT

Figure 26-3: Card-Detect Signals

Internal Direct Memory Access Controller (IDMAC)

The Internal Direct Memory Access Controller (IDMAC) contains a single transmit or receive engine, which transfers data from the host memory to the device port and conversely. The controller uses a descriptor to move data efficiently from source to destination with minimal core intervention. Programs can configure the controller to interrupt the core in situations such as data transmit and receive transfer completion from the card, as well as other normal or error conditions.

The IDMAC and the core communicate through a single data structure. The IDMAC transfers the data received from the card to the data buffer in the host memory, and it transfers transmit data from the data buffer in the host memory to the MSI FIFO. Descriptors that reside in the memory act as pointers to these buffers.

A data buffer resides in physical memory space of the processor and consists of complete data or partial data. Buffers contain only data, while buffer status is maintained in the descriptor. Data chaining refers to data that spans multiple data buffers. However, a single descriptor cannot span multiple data.

A single descriptor is used for both reception and transmission. The base address of the list is written into descriptor list base address register (`MSI_DBADDR`). A descriptor list is forward-linked. The last descriptor can point back to the first entry creating a ring structure. The descriptor list resides in the physical memory address space of the host. Each descriptor can point to a maximum of two data buffers.

Programs can enable or disable the IDMAC using the `MSI_CTL.INTDMAC` bit of the BIU.

DMA Descriptors

The IDMAC uses two types of descriptor structures:

- Dual-Buffer Structure – The skip length value programmed in the `MSI_BUSMODE.DSL` bit field determines the distance between two descriptors.
- Chain Structure – Each descriptor points to a unique buffer and the next descriptor.

The ***Dual-Buffer Descriptor Structure*** and ***Chain Descriptor Structure*** figures show these descriptor structures.

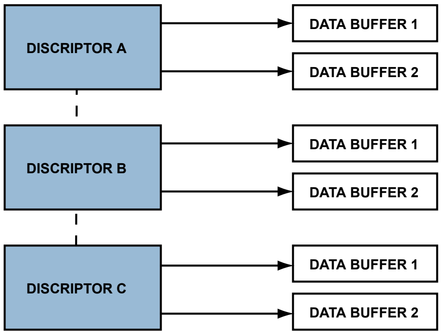

DISCRIPTOR A
DATA BUFFER 1
DATA BUFFER 2
DISCRIPTOR B
DATA BUFFER 1
DATA BUFFER 2
DISCRIPTOR C
DATA BUFFER 1
DATA BUFFER 2

Figure 26-4: Dual-Buffer Descriptor Structure

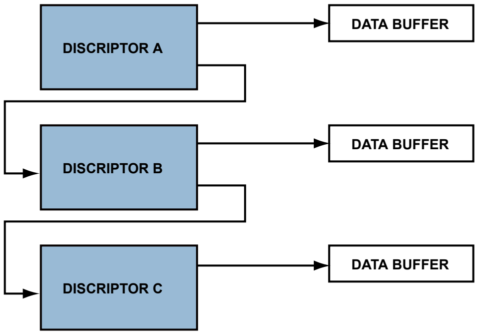

DISCRIPTOR A
DATA BUFFER
DISCRIPTOR B
DATA BUFFER
DISCRIPTOR C
DATA BUFFER

Figure 26-5: Chain Descriptor Structure

The ***Descriptor Format*** figure illustrates the internal format of a descriptor. The descriptor addresses must align with the 32-bit bus width. Each descriptor contains 16 bytes of control and status information. DES0 is a notation used to denote the [31:0] bits, DES1 to denote [63:32] bits, DES2 to denote [95:64] bits, and DES3 to denote [127:96] bits in a descriptor.

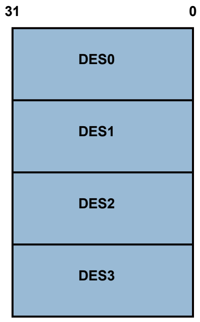

31
0
DES0
DES1
DES2
DES3

Figure 26-6: Descriptor Format

The following tables provide descriptor bit descriptions. Bits not shown are reserved. The DES0 descriptor in the IDMAC (described in the ***IDMAC DES0 Descriptor*** table) contains control and status information.

Table 26-5: IDMAC DES0 Descriptor

Bits	Name	Description
31	OWN	When set, this bit indicates that the IDMAC owns the descriptor. When this bit is reset, it indicates that the host owns the descriptor. The IDMAC clears this bit when it completes the data transfer.
30	Card Error Summary (CES)	This error bit indicates the status of the transaction to or from the card. This bit is also present in the `MSI_ISTAT` register and indicates the logical OR of the following bits: EBE: End bit error RTO: Response timeout RCRC: Response CRC SBE: Start bit error DRTO: Data Read timeout DCRC: Data CRC for receive RE: Response error
5	End of Ring (ER)	When set, this bit indicates that the descriptor list reached its final descriptor. The IDMAC returns to the base address of the list, creating a descriptor ring. This feature is meaningful for only a dual-buffer descriptor structure.
4	Second Address Chained (CH)	When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, BS2 (DES1[25:13]) is all zeros.
3	First Descriptor (FS)	When set, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, next descriptor contains the beginning of the data.
2	Last Descriptor (LD)	This bit is associated with the last block of a DMA transfer. When set, the bit indicates that the buffers pointed to by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count is 0.
1	Disable Interrupt on Completion (DIC)	When set, this bit prevents the setting of the TI/RI bit of the IDMAC Status Register (IDSTS) for the data that ends in the buffer pointed to by this descriptor.

The DES1 descriptor (described in the ***IDMAC DES1 Descriptor*** table) contains the buffer size.

Table 26-6: IDMAC DES1 Descriptor

Bits	Name	Description
25:13	Buffer 2 Size (BS2)	These bits indicate the second data buffer byte size which must be a multiple of 4. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and proceeds to the next buffer if there is a dual-buffer structure. This field is not valid for chain structure; that is, if DES0[4] is set.
12:0	Buffer 1 Size (BS1)	Indicates the data buffer byte size which must be a multiple of 4. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. This field must not be zero. Note: If there is only one buffer to be programmed, use only the buffer 1, and not buffer 2.

The DES2 descriptor (described in the ***IDMAC DES2 Descriptor*** table) is the address pointer to the data buffer.

Table 26-7: IDMAC DES2 Descriptor

Bits	Name	Description
31:0	Buffer Address Pointer 1 (BAP1)	These bits indicate the physical address of the first data buffer. The IDMAC ignores DES2 [1:0], internally.

The DES3 descriptor (described in the ***IDMAC DES3 Descriptor*** table) is the address pointer to the next descriptor when the present descriptor is not

- the last descriptor in a chained descriptor structure, or
- the second buffer address for a dual-buffer structure

Table 26-8: IDMAC DES3 Descriptor

Bits	Name	Description
31:0	Buffer Address Pointer 2/ Next Descriptor Address (BAP2)	These bits indicate the physical address of the second buffer when the dual-buffer structure is used. If the Second Address Chained (DES0[4]) bit is set, then this address contains the pointer to the physical memory where the next descriptor is present. If this pointer is not the last descriptor, then the next descriptor address pointer must be bus-width aligned (DES3[1:0] = 0. Internally, the LSBs are ignored.

Initialization

IDMAC initialization occurs as follows.

1. Write to IDMAC bus mode register (`MSI_BUSMODE`) to set host bus access parameters.
2. Write to IDMAC interrupt enable register (`MSI_IDINTEN`) to mask unnecessary interrupt causes.
3. The software driver creates either the transmit or the receive descriptor list. Then it writes to IDMAC descriptor list base address register (`MSI_DBADDR`), which provides the IDMAC the starting address of the list.
4. The IDMAC engine attempts to acquire descriptors from the descriptor lists.

Host Bus Burst Access

The IDMAC executes fixed-length burst transfers on the bus interface when the `MSI_BUSMODE.FB` bit =1. The maximum burst length is indicated and limited by the `MSI_BUSMODE.PBL` bit field. The descriptors are always accessed in the maximum burst-size for the 16-bytes to be read (4-word burst).

The IDMAC initiates a data transfer only when sufficient space to accommodate the configured burst is available in the FIFO or the number of bytes to the end of data, when less than the configured burst-length. The IDMAC indicates the start address and the number of transfers required to the bus interface. When the bus interface is configured for fixed-length bursts, it transfers data using the best combination of INCR4/8/16 and SINGLE transactions. Otherwise, in no fixed-length bursts, it transfers data using INCR (undefined length) and SINGLE transactions.

The transmit and receive data buffers must be aligned to data bus width (32-bit).

NOTE: Due to the bus protocol limitation, the burst mode address should not cross the 1 KB boundary. Address [31:10] should not change during a burst.

Buffer Size Calculations

The software knows the amount of data to transmit or receive. For transmitting to the card, the IDMAC transfers the exact number of bytes to the FIFO, indicated by the buffer size field of DES1.

If a descriptor is not marked as last (LS bit of DES0), then the corresponding buffers of the descriptor are full. Its buffer size field indicates the amount of valid data in a buffer. If a descriptor is marked as last, then the buffer cannot be full, as indicated by the buffer size in DES1. The software is aware of the number of locations that are valid in this case.

Data Transmit/Receive

IDMAC transmission occurs as follows:

1. The software sets up the descriptor (DES0–DES3) for transmission and sets the OWN bit (DES0[31]). The software also prepares the data buffer.
2. The software programs the write data command in the `MSI_CMD` register in the BIU.
3. The software also programs the required transmit threshold level (`MSI_FIFOTH.TXWM` bit field).
4. The IDMAC determines that a write data transfer must occur as a consequence of step 2.
5. The IDMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the software owns the descriptor. In this case, the IDMAC enters the suspend state and asserts the `MSI_IDSTS.DU` bit. The software must release the IDMAC by writing any value to the poll demand register.
6. It then waits for command done (`MSI_ISTAT.CMDDONE`) bit and no errors from BIU which indicates that a transfer can occur.
7. The IDMAC engine waits for a DMA interface request from the BIU. This request is generated based on the programmed transmit threshold value. For the last bytes of data which cannot be accessed using a burst, SINGLE transfers are performed.
8. The IDMAC fetches the transmit data from the data buffer and transfers it to the FIFO for transmission to the card.
9. When data spans across multiple descriptors, the IDMAC fetches the next descriptor and continues with its operation with the next descriptor. The last descriptor bit in the descriptor indicates whether the data spans multiple descriptors or not.
10. When data transmission is complete, status information is updated in IDMAC status register (`MSI_IDSTS`) by setting the transmit interrupt, when enabled. Also, the IDMAC performs a write transaction to DES0 to clear the OWN bit.

IDMAC reception occurs as follows:

1. The software sets up the Descriptor (DES0–DES3) for reception, sets the OWN bit (DES0[31]).

2. The software programs the read data command in the `MSI_CMD` register in the BIU.
3. The software programs the required receive threshold level (`MSI_FIFOTH.RXWM` bit field).
4. The IDMAC determines that a read data transfer must occur as a consequence of step 2.
5. The IDMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMA enters suspend state and asserts the release the IDMAC by writing any value to the poll demand register.
6. It then waits for the command done (`MSI_ISTAT.CMDDONE`) bit and no errors from BIU which indicates that a transfer can occur.
7. The IDMAC engine waits for a DMA interface request from the BIU. This request is generated based on the programmed receive threshold value. For the last bytes of data which cannot be accessed using a burst, SINGLE transfers are performed.
8. The IDMAC fetches the data from the FIFO and transfers to the memory.
9. When data spans across multiple descriptors, the IDMAC fetches the next descriptor and continues with its operation with the next descriptor. The last descriptor bit in the descriptor indicates whether the data spans multiple descriptors or not.
10. When data reception is complete, status information is updated in the IDMAC status register (`MSI_IDSTS`) by setting the receive interrupt, when enabled. Also, the IDMAC performs a write transaction to DES0 to clear the OWN bit.

Interrupts

Interrupts can be generated as a result of various DMAC events. The IDMAC status register (`MSI_IDSTS`) contains all the bits that can cause an interrupt. The IDMAC interrupt enable register (`MSI_IDINTEN`) contains an enable bit for each of the events that can cause an interrupt.

There are two groups of summary interrupts—normal and abnormal—as outlined in the `MSI_IDSTS` register. Interrupts are cleared by writing a 1 to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared.

Interrupts are not queued and if the interrupt event occurs before the software has responded to it, no additional interrupts are generated. For example, the receive interrupt (`MSI_IDSTS.RI`) indicates that one or more data was transferred to the buffer. An interrupt is generated only once for simultaneous, multiple events. The software must scan the `MSI_IDSTS` register for the interrupt cause.

NOTE: The final interrupt signal from MSI is a logical OR of the interrupt from the BIU and the IDMAC.

Finite State Machine (FSM)

The IDMAC finite state machine can be in any one of the states reflected in the `MSI_IDSTS.FSM` bit field.

The FSM uses the following sequence.

1. IDMAC performs four accesses for fetching the descriptor.

2. Stores descriptors in internal register and also issues a FIFO reset when it is first descriptor.
3. Each bit is checked for correctness. In case of bit mismatches, appropriate error bit is set. If it is first descriptor, then issues a FIFO reset and wait until FIFO reset is complete. The error status indicates one of the following:
   - Response timeout
   - Response CRC error
   - Data receive timeout
   - Response error
4. The FSM waits in current state until DMA request is asserted, which implies that, FIFO:
   - For DMA write request wait – Holds the number of data, indicated by FIFO RX watermark. In case of error due to response timeout or error, FSM goes to DESC_CLOSE state to close descriptor.
   - For DMA read request wait – Holds number of data, indicated by FIFO TX watermark. In case of error, FSM goes to DESC_CLOSE state to close descriptor.
5. FSM performs the following:
   - For DMA write – Requests a write to SCB. If number of beats in one transfer is greater than PBL, then one of the following occurs:

     Burst count to SCB is PBL value

     Single transfers are initiated
   - For DMA read – Requests a read from SCB. If number of beats in one transfer is greater than PBL, then one of the following occurs:

     Burst count to SCB is PBL value

     Single transfers are initiated
6. After the programmed transfer count is accessed from the memory, the OWN bit in descriptor is closed. If a transfer spans more than one descriptor, the FSM fetches the next descriptor. If the transfer ends with the current descriptor, the FSM goes to the idle state after setting the receive or transmit interrupt. Depending on the descriptor structure—dual buffer or chained—the appropriate starting address of descriptor is loaded. If it is the second data buffer of the dual buffer, the descriptor is not fetched again.

Abort Operation

The host issues CMD12 when a data transfer on the card data lines is in progress. The FSM closes the present descriptor after completing the transfer of data until a DTO interrupt is asserted. Once an abort command is issued, the DMAC performs single burst transfers.

1. For a card write, the FSM keeps pushing data to the FIFO after fetching it from the memory until a DTO interrupt is asserted. The controller asserts a busy clear interrupt after the DTO interrupt. It ensures that the card has completed driving the busy signal. This sequence occurs to keep the card clock running so that CMD12 is reliably sent to the card.

2. For a card read, the FSM keeps popping data from the FIFO and writes to the memory until a DTO interrupt is generated. This sequence is required since the DTO interrupt is not generated until and unless all the FIFO data is emptied.

NOTE: If an FBE occurs, the IDMAC FSM does not close the current descriptor and the remaining unread descriptors.

If a write abort occurs, the IDMAC FSM closes only the current descriptor during which the abort occurred. The IDMAC FSM does not close the current descriptor and the remaining unread descriptors.

If a read abort occurs, the IDMAC FSM pops the data out of the FIFO and writes it to the corresponding descriptor data buffers. The remaining unread descriptors are not closed.

FIFO Overflow and Underflow

During normal data transfer conditions, FIFO overflow and underflow do not occur. If there is a programming error, then a FIFO overflow or underflow can result as shown in the following examples.

Transmit settings: `MSI_BUSMODE.PBL` =4, `MSI_FIFOTH.TXWM` =1

In this example, if the FIFO has only one location empty, it issues a DMA request to the IDMAC FSM. Because `MSI_BUSMODE.PBL` =4, the IDMAC FSM performs 4 pushes into the FIFO which results in a FIFO overflow interrupt.

Receive settings: `MSI_BUSMODE.PBL` =4, `MSI_FIFOTH.RXWM` =1

In this example, if the FIFO has only one location filled, it issues a DMA request to the IDMAC FSM. Because `MSI_BUSMODE.PBL` =4, the IDMAC FSM performs 4 pops to the FIFO which results in a FIFO underflow interrupt.

The software must ensure that the number of bytes to be transferred as indicated in the descriptor are a multiple of 4. For example, if the `MSI_BYTCNT` register =13, the number of bytes indicated in the descriptor must be 16.

Card Interface Unit

The Card Interface Unit (CIU) interfaces with the Bus Interface Unit (BIU) on one side and with the SD/MMC/SDIO cards or devices on other side. The software writes command parameters to the MSI BIU control registers, and these parameters are then passed to the CIU. Depending on control register values, the CIU generates card command and data traffic on a selected card bus according to card protocol.

As shown in the MSI Block Diagram, the CIU consists of the following primary functional blocks:

- Command path
- Datapath
- SDIO interrupt control
- Clock control
- Multiplex or demultiplex unit

The following software restrictions must be met for proper CIU operation:

- Only one data transfer command can be issued at a time.
- During an open-ended card write operation, if the card clock is stopped because the FIFO is empty, the software must first fill the data into the FIFO and start the card clock. It can then issue only a stop or abort command to the card.
- During an SDIO card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset the FIFO and start the resume command as if it were a new data transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52) while a card data transfer is in progress, the software must set the `MSI_CMD.STPABORTCMD` bit so that the MSI can stop the data transfer after issuing the card reset command.
- When the data end error bit (`MSI_ISTAT.EBE`) is set, the MSI does not guarantee SDIO interrupts. The software must ignore the SDIO interrupts and issue the stop or abort command to the card, so that the card stops sending the read data.
- If the card clock is stopped because the FIFO is full during a card read, the software or DMA must read at least two FIFO locations to start the card clock.

Command Path

The command path performs the following functions.

- Loads clock parameters
- Loads card command parameters
- Sends commands to card bus
- Receives responses from card bus
- Sends responses to BIU
- Drives the P-bit on command line

A new command is issued to the MSI by programming the BIU registers and setting the `MSI_CMD.STARTCMD` bit. The command path loads this new command (command, command argument, timeout) and sends acknowledge to the BIU.

Once the new command is loaded, the command path state machine sends a command to the SD/MMC bus (this includes the internally generated CRC7). The command path state machine receives a response if there is any. The state machine then sends the received response and signals to the BIU that the command is done, and then waits for eight clock cycles before loading a new command.

Load Command Parameters

One of the following commands or responses is loaded in the command path:

- New command from BIU when the `MSI_CMD.STARTCMD` bit is set.

- Internally generated auto-stop command when the data path ends, the stop command request is loaded.
- IRQ response with RCA =0x000 when the command path is waiting for an IRQ response from the MMC card, then the `MSI_CTL.IRQRESP` bit is set.

Loading a new command from the BIU in the command path depends on the following `MSI_CMD` register bit settings:

- Update clock registers only using the `MSI_CMD.UCLKREGS` bit. If the `MSI_CMD.UCLKREGS` bit =1, the command path updates only the clock enable and clock divider registers. If `MSI_CMD.UCLKREGS` =0, the command path loads the command, command argument, and timeout registers, then starts processing the new command.
- Wait for previous data to complete using the `MSI_CMD.WTPRIVDATA` bit. If the `MSI_CMD.WTPRIVDATA` bit =1, the command path loads the new command under one of the following conditions:
  - Immediately, if the data path is free (that is, there is no data transfer in progress), or if an open-ended data transfer is in progress (`MSI_BYTCNT` =0).
  - After completion of the current data transfer, if a predefined data transfer is in progress.

Send Command and Receive Response

Once a new command is loaded in the command path with `MSI_CMD.UCLKREGS` bit =0, the command path state machine sends out a command on the SD/MMC bus. The ***SD_MMC Command Path State Machine*** figure illustrates the command path state machine.

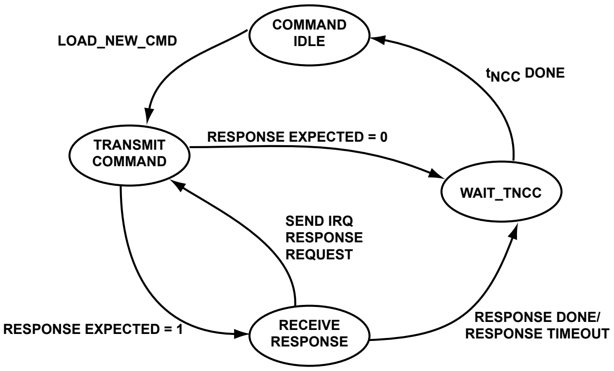


Figure 26-7: SD_MMC Command Path State Machine

The command path state machine performs the following functions, according to following command register bit values:

1. `MSI_CMD.SENDINIT` (send initialization). Initialization sequence of 80 clocks is sent before sending the command.
2. `MSI_CMD.RXPECT` (response expected). A response is expected for the command. After the command is sent out, the command path state machine receives a 48-bit or 136-bit response and sends it to the BIU. If the start

bit of the card response is not received within the number of clocks programmed in the timeout register, then the `MSI_ISTAT.RTO`(response timeout) and `MSI_ISTAT.CD` (command done) bits are set as a signal to the BIU. If the `MSI_CMD.RXPECT` bit is not set, the command path sends out a command and signals a response done to the BIU, that is, the `MSI_ISTAT.CD` bit is set.

3. `MSI_CMD.RLEN` (response length). If =1, a 136-bit response is received; if =0, a 48-bit response is received.
4. `MSI_CMD.CHKRESPCRC` (check response CRC). If =1, the command path compares CRC7 received in the response with the internally-generated CRC7. If the two do not match, the response CRC error is signaled to the BIU, that is, the response CRC error bit (`MSI_ISTAT.RCRC`) is set.

Send Response to BIU

If the `MSI_CMD.RXPECT` bit =1, the received response is sent to the BIU. The `MSI_RESP0` register is updated for a short response, and the `MSI_RESP3`, `MSI_RESP2`, `MSI_RESP1`, and `MSI_RESP0` registers are updated on a long response, after which the `MSI_ISTAT.CMDDONE` bit is set. If the CIU sends an auto-stop command as a response, the response is saved in the `MSI_RESP1` register, after which the `MSI_ISTAT.ACD` (auto command done) bit is set.

Additionally, the command path checks for the following.

- Transmission bit =0
- Command index in response matches command index of the sent command
- End bit =1 in received card response

The command index is not checked for a 136-bit response or if the `MSI_CMD.CHKRESPCRC` bit is cleared. For a 136-bit response and reserved CRC 48-bit responses, the command index is reserved (=111111).

Driving a P-bit on CMD Line

The command path drives a P-bit =1 on the CMD line between two commands when a response is not expected. If a response is expected, the P-bit is driven after the response is received and before the start of the next command.

Datapath

The datapath block pops the data FIFO and transmits data on MSI data lines during a write data transfer. Or, it receives data from data lines and pushes it into the FIFO during a read data transfer. Whenever a data transfer command is not in-progress, the datapath loads new data parameters;

- data expected
- read/write data transfer
- stream or block transfer
- block size
- byte count

- card type
- timeout registers

If the `MSI_CMD.DXPECT` bit =1, the new command is a data transfer command and the datapath starts one of the following:

- Data transmit if the `MSI_CMD.RDWR` (read/write) bit =1
- Data receive if `MSI_CMD.RDWR` (read/write) bit 0

Auto-Stop

The MSI internally generates a stop command and is loaded in the command path when the `MSI_CMD.SENDASTOP` bit is set. The auto-stop command helps to send an exact number of data bytes using a stream read or write for the MMC, and a multiple-block read or write for the SD memory transfer for SD cards.

The software must set the `MSI_CMD.SENDASTOP` bit according to details listed in the ***Auto-Stop Generation*** table.

Table 26-9: Auto-Stop Generation

Card Type	Transfer Type	Byte Count	**`MSI_CMD.SENDASTOP` bit =1**	Comments
MMC	Stream read	0	No	Open-ended stream
	Stream read	>0	Yes	Auto-stop after all bytes transfer
	Stream write	0	No	Open-ended stream
	Stream write	>0	Yes	Auto-stop after all bytes transfer
	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal
	Multiple-block read	0	No	Open-ended multiple block
	Multiple-block read	>0	Yes[*1]	Pre-defined multiple block
	Multiple-block write	0	No	Open-ended multiple block
	Multiple-block write	>0	Yes	Pre-defined multiple block
SDMEM	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal
	Multiple-block read	0	No	Open-ended multiple block
	Multiple-block read	>0	Yes	Auto-stop after all bytes transfer
	Multiple-block write	0	No	Open-ended multiple block
	Multiple-block write	>0	Yes	Auto-stop after all bytes transfer
SDIO	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal

Table 26-9: Auto-Stop Generation (Continued)

Card Type	Transfer Type	Byte Count	`MSI_CMD.SENDASTOP` bit =1	Comments
	Multiple-block read	0	No	Open-ended multiple block
	Multiple-block read	>0	No	Pre-defined multiple block
	Multiple-block write	0	No	Open-ended multiple block
	Multiple-block write	>0	No	Pre-defined multiple block

*1 The condition under which the transfer mode blocks transfer and `byte_count` equals block size is treated as a single-block data transfer command for both MMC and SD cards. If byte_count = n × block_size (n = 2, 3, …), the condition is treated as a predefined multiple-block data transfer command. For an MMC card, the host software can perform a predefined data transfer in two ways:

1. Issue the CMD23 command before issuing CMD18/CMD25 commands to the card – in this case, issue CMD18/CMD25 commands without setting the `MSI_CMD.SENDASTOP` bit.
2. Issue CMD18/CMD25 commands without issuing CMD23 command to the card, with the `MSI_CMD.SENDASTOP` bit set. In this case, an internally generated auto-stop command after the programmed byte count terminates the multiple-block data transfer.

The following list explains different conditions for the auto-stop command.

- Stream read for MMC card with byte count greater than 0. The MSI generates an internal stop command and loads it into the command path. The end bit of the stop command is sent out when the last byte of data is read from the card and no extra data byte is received. If the byte count is less than 6 (48 bits), a few extra data bytes are received from the card before the end bit of the stop command is sent.
- Stream write for MMC card with byte count greater than 0. The MSI generates an internal stop command and loads it into the command path. The end bit of the stop command is sent when the last byte of data is transmitted on the card bus and no extra data byte is transmitted. If the byte count is less than 6 (48 bits), the datapath transmits the data last in order to meet the above condition.
- Multiple-block read memory for SD card with byte count greater than 0. If the block size is less than 4 (single-bit data bus), 16 (4-bit data bus), or 32 (8-bit data bus) bytes, the auto-stop command is loaded in the command path after all the bytes are read. Otherwise, the stop command is loaded in the command path so that the end bit of the stop command is sent after the last data block is received.
- Multiple-block write memory for SD card with byte count greater than 0. If the block size is less than 3 (single-bit data bus), 12 (4-bit data bus), or 24 (8-bit data bus), the auto-stop command is loaded in the command path after all data blocks are transmitted. Otherwise, the stop command is loaded in the command path so that the end bit of the stop command is sent after the end bit of the CRC status is received.
- Precaution for host software during auto-stop. Whenever an auto-stop command is issued, the host software must not issue a new command to the MSI until the MSI sends the auto-stop command and the data transfer completes. If the host issues a new command during a data transfer with the auto-stop in progress, an auto-stop command can be sent after the new command is sent and its response is received. This process can delay sending the stop command, which transfers extra data bytes. For a stream write, extra data bytes are erroneous data

that can corrupt the card data. If the host wants to terminate the data transfer before the data transfer is complete, it can issue a stop or abort command. In this case, the MSI does not generate an auto-stop command.

Non-Data Transfer Commands that Use Datapath

Some non-data transfer commands (non-read/write commands) also use the datapath. The ***Non-Data Transfer Commands and Requirements*** table lists the commands and register programming requirements.

Table 26-10: Non-Data Transfer Commands and Requirements

	CMD27	CMD30	CMD42	ACMD13	ACMD22	ACMD51
Command Register Programming						
Cmd_index	6'h1B	6'h1E	6'h2A	6'h0D	6'h16	6'h33
Response_expect	1	1	1	1	1	1
Response_length	0	0	0	0	0	0
Check_response_crc	1	1	1	1	1	1
Data_expected	1	1	1	1	1	1
Read/write	1	0	1	0	0	0
Transfer_mode	0	0	0	0	0	0
Send_auto_stop	0	0	0	0	0	0
Wait_prevdata_complete	0	0	0	0	0	0
Stop_abort_cmd	0	0	0	0	0	0
Command Argument register programming						
	Stuff bits	32-bit write protect data address	Stuff bits	Stuff bits	Stuff bits	Stuff bits
Block Size register programming						
	16	4	Num_bytes*1	64	4	8
Byte Count register programming						
	16	4	Num_bytes*1	64	4	8

*1 Number of bytes specified as per the lock card data structure (Refer to the SD specification and the MMC specification).

SDIO Interrupt Control

Interrupts for SD cards are reported to the BIU by asserting an interrupt signal for two clock cycles. SDIO cards signal an interrupt by asserting `MSI0_D1` low during the interrupt period; The interrupt control state machine determines an interrupt period for the selected card. An interrupt period is always valid for cards in 1-bit data mode. An interrupt period for a wide-bus active or selected card is valid for the following conditions:

- Card is idle
- Non-data transfer command in progress

- Third clock after end bit of data block between two data blocks
- From two clocks after end bit of last data until end bit of next data transfer command

Bear in mind that, in the following situations, the MSI does not sample the SDIO interrupt of the selected card when the card data width is 4 bits. Since the SDIO interrupt is level-triggered, it is sampled in a further interrupt period and the host does not lose any SDIO interrupt from the card.

1. Read/write resume. The CIU treats the resume command as a normal data transfer command. SDIO interrupts during the resume command are handled similarly to other data commands. According to the SDIO specification, for the normal data command the interrupt period ends after the command end bit of the data command. For the resume command, it ends after the response end bit. For the resume command, the MSI stops the interrupt sampling period after the resume command end bit, instead of stopping after the response end bit of the resume command.
2. Suspend during read transfer. If the host suspends the read data transfer, the host sets the `MSI_CTL.RDABORT` bit to reset the data state machine. In the CIU, the SDIO interrupts are handled such that the interrupt sampling starts after the host sets the `MSI_CTL.RDABORT` bit. In this case, the MSI does not sample SDIO interrupts between the period from response of the suspend command.

Clock Control

The clock control block provides the clock frequencies required for SD/MMC cards.

The source clock for the MSI block and the input clock for clock dividers of the clock control block is clocked by CLK09 clock from CDU module. Refer to CDU chapter for more details on CLK09 clock.

The source clock is used to generate the card clock frequencies. The card clock can have different clock frequencies, since the SD card can be a low-speed SD card or a full-speed SD card. The MSI allows the card to operate at different clock frequencies.

The clock frequency of a card depends on the following clock control registers:

- Clock divider register (`MSI_CLKDIV`). An internal clock divider is used to generate different clock frequencies required for the card. The clock divider is an 8-bit value that provides a clock division factor from 1 to 510. A value of 0 represents a clock-divider bypass, a value of 1 represents a divide by 2, a value of 2 represents a divide by 4, and so on.
- Clock control register (`MSI_CLKEN`). The MSI module can enable or disable the card clock under the following conditions:
  - The clock for a card is enabled when the `MSI_CLKEN.EN0` bit for a card is programmed (= 1) or disabled (= 0).
  - Setting the `MSI_CLKEN.LP0` =1. enables the low-power mode of a card. If low-power mode is enabled to save card power, the clock signal is disabled when the card is idle for at least 8 card clock cycles. It is enabled when a new command is loaded and the command path goes to a non-idle state.

Additionally, the clock of the card is disabled when:

- An internal FIFO is full on a card read (no more data can be received from card)
- The FIFO is empty on a card write (no data is available for transmission).

Disabling the clock in these situations helps to avoid FIFO overrun and underrun conditions.

Under the following conditions, the card clock is stopped or disabled for the card:

- The clock can be disabled by writing to the clock enable `MSI_CLKEN` register.
- If low-power mode is selected and a card is idle, or not selected for 8 clocks.
- The FIFO is full and data path cannot accept more data from the card and data transfer is incomplete to avoid FIFO overrun.
- The FIFO is empty and data path cannot transmit more data to the card and data transfer is incomplete to avoid FIFO underrun.

NOTE: The host software must take care while changing the `MSI_CLKDIV` register values. The card clock must be disabled through the `MSI_CLKEN` register before changing the values of the `MSI_CLKDIV` register.

MSI Data Transfer Modes

The following sections provide information on data transfer using the MSI interface.

Data Transmit

The data transmit state machine starts data transmission two clocks after a response for the data write command is received. The transmission occurs even if the command path detects a response error or response CRC error. See the ***Data Transmit State Machine*** figure. If a response is not received from the card because of a response timeout, data is not transmitted. Depending upon the value of the `MSI_CMD.XFRMODE` bit, the data transmit state machine puts data on the card data bus in a stream or in blocks.

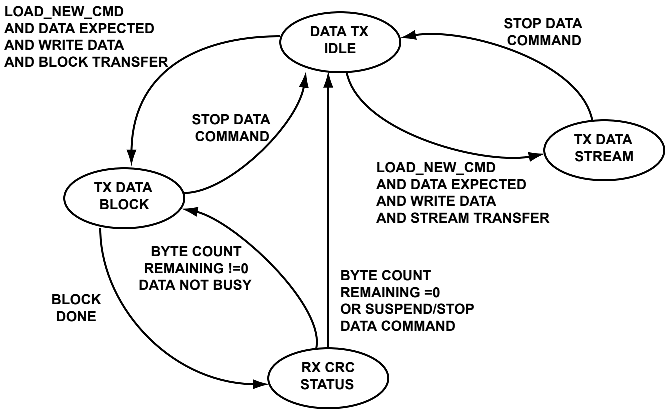

LOAD_NEW_CMD
AND DATA EXPECTED
AND WRITE DATA
AND BLOCK TRANSFER
DATA TX
IDLE
STOP DATA
COMMAND
STOP DATA
COMMAND
TX DATA
STREAM
TX DATA
BLOCK
LOAD_NEW_CMD
AND DATA EXPECTED
AND WRITE DATA
AND STREAM TRANSFER
BYTE COUNT
REMAINING !=0
DATA NOT BUSY
BLOCK
DONE
BYTE COUNT
REMAINING =0
OR SUSPEND/STOP
DATA COMMAND
RX CRC
STATUS

Figure 26-8: Data Transmit State Machine

Stream Data Transmit

If the `MSI_CMD.XFRMODE` bit =1, it is a stream-write data transfer. The data path pops the FIFO from the BIU and transmits in a stream to the card data bus. If the FIFO becomes empty, the card clock is stopped and restarted once data is available in the FIFO.

If the `MSI_BYTCNT` register is programmed to 0, it is an open-ended stream-write data transfer. During this data transfer, the data path continuously transmits data in a stream until the host software issues a stop command. A stream data transfer is terminated when the end bit of the stop command and end bit of the data match over two clocks.

If the `MSI_BYTCNT` register is programmed with a non-zero value and the `MSI_CMD.SENDASTOP` bit is set, the stop command is internally generated. The command is loaded in the command path when the end bit of the stop command occurs after a match with the last byte of the stream-write transfer. This data transfer can also terminate if the software issues a stop command before all the data bytes transfer to the card bus.

Single Block Data

If the `MSI_CMD.XFRMODE` bit =0 and the `MSI_BYTCNT` register value is equal to the value of the `MSI_BLKSIZ` register, a single-block write-data transfer occurs. The data transmit state machine sends data in a single block, where the number of bytes equals the block size, including the internally-generated CRC16.

If the `MSI_CTYPE` register bit is set for a 1-bit, 4-bit, or 8-bit data transfer, the data transmits on 1, 4, or 8 data lines, respectively. CRC16 is separately generated and transmitted for 1, 4, or 8 data lines, respectively.

After a single data block is transmitted, the data transmit state machine receives the CRC status from the card and signals a data transfer to the BIU. This operation happens when the `MSI_ISTAT.DTO` bit =1. If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the `MSI_ISTAT.DCRC` bit.

Additionally, if the start bit of the CRC status is not received before two clocks after the end of the data block, a CRC status start bit error is signaled to the BIU. The `MSI_ISTAT.EBE` bit is set.

Multiple Block Data

A multiple-block write-data transfer occurs if the `MSI_CMD.XFRMODE` bit =0 and the value in the `MSI_BYTCNT` register is not equal to the value of the `MSI_BLKSIZ` register. The data transmit state machine sends data in blocks, where the number of bytes in a block equals the block size, including the internally-generated CRC16.

If the `MSI_CTYPE` register bit is set to 1-bit, 4-bit, or 8-bit data transfer, the data transmits on 1, 4, or 8 data lines, respectively. CRC16 is separately generated and transmitted on 1, 4, or 8 data lines, respectively.

After one data block is transmitted, the data transmit state machine receives the CRC status from the card. If the remaining `MSI_BYTCNT` becomes 0, the data path signals to the BIU that the data transfer is complete. This operation happens when the `MSI_ISTAT.DTO` bit =1. If the remaining data bytes are greater than 0, the data path state machine starts to transmit another data block.

If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the `MSI_ISTAT.DCRC` bit. The block continues further data transmission until all the bytes are sent. Additionally, if the CRC status start bit is not received within two clocks after the end of a data block, a CRC status start bit error is signaled to the BIU. The `MSI_ISTAT.EBE` bit is set. Further data transfer is terminated.

If the `MSI_CMD.SENDASTOP` is set, the stop command is internally generated during the transfer of the last data block. No extra bytes are transferred to the card. The end bit of the stop command does not always exactly match the end bit of the CRC status in the last data block.

If the block size is less than 4, 16, or 32 for card data widths of 1 bit, 4 bits, or 8 bits, respectively, the data transmit state machine terminates the data transfer when all the data has transferred. The internally generated stop command is loaded in the command path.

If the `MSI_BYTCNT` register =0 (the block size must be greater than 0), it is an open-ended block transfer. The data transmit state machine for this type of data transfer continues the block-write data transfer until the host software issues a stop or abort command.

Data Receive

The data-receive state machine receives data two clock cycles after the end bit of a data read command, even if the command path detects a response error or response CRC error. See the ***Data Receive State Machine*** figure. If a response is not received from the card because a timeout error occurs, the BIU does not receive the signal that the data transfer is complete. This error happens when the command sent by the MSI is an illegal operation for the card. The error prevents the card from starting a read data transfer.

If data is not received before the data timeout, the data path signals a data timeout to the BIU and an end to the data transfer done. Based on the value of the `MSI_CMD.XFRMODE` bit, the data-receive state machine gets data from the card data bus in a stream or blocks.

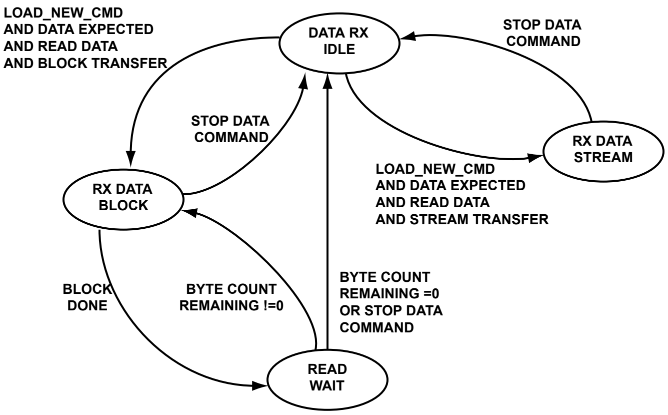

LOAD_NEW_CMD
AND DATA EXPECTED
AND READ DATA
AND BLOCK TRANSFER
DATA RX
IDLE
STOP DATA
COMMAND
STOP DATA
COMMAND
RX DATA
STREAM
LOAD_NEW_CMD
AND DATA EXPECTED
AND READ DATA
AND STREAM TRANSFER
RX DATA
BLOCK
BLOCK
DONE
BYTE COUNT
REMAINING !=0
BYTE COUNT
REMAINING =0
OR STOP DATA
COMMAND
READ
WAIT

Figure 26-9: Data Receive State Machine

Stream Data Read

A stream-read data transfer occurs if the `MSI_CMD.XFRMODE` bit =1, at which time the data path receives data from the card and pushes it to the FIFO. If the FIFO becomes full, the card clock stops and restarts once the FIFO is no longer full.

An open-ended stream-read data transfer occurs when the `MSI_BYTCNT` register equals 0. During this type of data transfer, the data path continuously receives data in a stream until the host software issues a stop command. A stream data transfer terminates two clock cycles after the end bit of the stop command.

If the `MSI_BYTCNT` register contains a non-zero value and the `MSI_CMD.SENDASTOP` bit =1, a stop command is internally generated and loaded into the command path. In the command path, the end bit of the stop command occurs after the last byte of the stream data transfer is received. This data transfer can terminate if the host issues a stop or abort command before all the data bytes are received from the card.

Single Block Data Read

A single-block read-data transfer occurs if the `MSI_CMD.XFRMODE` bit =0 and the value of the `MSI_BYTCNT` register is equal to the value of the `MSI_BLKSIZ` register. When a start bit is received before the data times out, data bytes equal to the block size and CRC16 are received and checked with the internally-generated CRC16.

If the `MSI_CTYPE` register bit for the card is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively. CRC16 is separately generated and checked for 1, 4, or 8 data lines, respectively. If there is a CRC16 mismatch, the data path signals a data CRC error to the BIU. If the received end bit is not 1, the BIU receives an end-bit error.

Multiple Block Data Read

If the `MSI_CMD.XFRMODE` =0 and the value of the `MSI_BYTCNT` register is not equal to the value of the `MSI_BLKSIZ` register, it is a multiple-block read-data transfer. The data-receive state machine receives data in blocks, where the number of bytes in a block is equal to the block size, including the internally-generated CRC16.

If the `MSI_CTYPE` register bit for the card is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively. CRC16 is separately generated and checked for 1, 4, or 8 data lines, respectively. After a data block is received, if the remaining `MSI_BYTCNT` becomes 0, the data path signals a data transfer to the BIU.

If the remaining data bytes are greater than 0, the data path state machine causes another data block to be received. If CRC16 of a received data block does not match the internally-generated CRC16, a data CRC error to the BIU and data reception continue further data transmission until all bytes are transmitted. Additionally, if the end of a received data block is not 1, data on the data path signals terminate the bit error to the CIU. The data-receive state machine terminates data reception, waits for a data timeout, and signals to the BIU that the data transfer is complete.

If the `MSI_CMD.SENDASTOP` bit =1, the stop command is internally generated when the last data block is transferred, where no extra bytes are transferred from the card. The end bit of the stop command does not always exactly match the end bit of the last data block.

If the requested block size for data transfers to cards is less than 4, 16, or 32 bytes for 1-bit, 4-bit, or 8-bit data transfer modes, respectively, the data-transmit state machine terminates the data transfer when all data transfers. The internally-generated stop command is loaded in the command path. The data path ignores any subsequent data received from the card.

If the `MSI_BYTCNT` =0 (the block size must be greater than 0), it is an open-ended block transfer. For this type of data transfer, the data-receive state machine continues the block-read data transfer until the host software issues a stop or abort command.

Data Transfer Commands

Data transfer commands transfer data between the memory card and the MSI. To send a data command, the MSI needs a command argument, total data size, and block size.

Prior to a data transfer command, software must confirm that the card is not busy and is in a transfer state. This confirmation uses the CMD13 and CMD7 commands, respectively. For the data transfer commands, it is important that the same bus width that is programmed in the card is set in the `MSI_CTYPE` register.

The MSI generates an interrupt for different conditions during data transfer, which are reflected in the `MSI_ISTAT` register.

NOTE: The DCRC, SBEBCI, EBE, and SBEBCI conditions indicate that the received data could have errors. If there was a response timeout, then no data transfer occurred. For more information, see the Register Descriptions section.

Transmission and Reception with Internal DMAC (IDMAC)

The general sequence of events for transmit and receive is as follows.

1. Program the required programming in the bus mode register (`MSI_BUSMODE`). If the `MSI_CTL.INTDMAC` bit is disabled during the middle of an IDMAC transfer, it has no effect. It only takes effect for a new data transfer command. Issuing a software reset immediately terminates the transfer. It is recommended that the software issue a reset to the DMA interface by setting the `MSI_CTL.DMARST` bit and then issuing an IDMAC software reset using the `MSI_CTL.CTLRST` bit. Program the fixed burst bit (`MSI_BUSMODE.FB`) appropriately for system performance.
2. When a descriptor unavailable interrupt is asserted, the software must form the descriptor, appropriately set its own bit and then write to poll the demand register for the IDMAC to refetch the descriptor.
3. It is always appropriate for the application to enable abnormal interrupts since any errors related to the transfer are reported to the application.

MSI Event Control

The following sections describe MSI events.

Dedicated Interrupt Pins

Interrupt lines are defined for only eSDIO devices, which are connected to the controller interrupt line in MSI. These interrupt lines can operate even when the card clock is switched off and can be used only during an asynchronous interrupt period.

MSI Status and Error Signals

The following provides information on MSI errors.

Error Detection

The MSI has an error detection mechanism which operates for any of the following situations.

Response

- Response timeout – Response expected with the response start bit is not received within programmed number of clocks in timeout register.
- Response CRC error – Response is expected and check response CRC requested; response CRC7 does not match with the internally-generated CRC7.
- Response error – Response transmission bit is not 0, command index does not match with the command index of the send command, or response end bit is not 1.

Data Transmit

- No CRC status – During a write data transfer, if the CRC status start bit is not received two clocks after the end bit of the data block is sent out, the datapath does the following:
  - Signals no CRC status error to the BIU
  - Terminates further data transfer
  - Signals data transfer done to the BIU
- Negative CRC – If the CRC status received after the write data block is negative (that is, not 010), a data CRC error is signaled to the BIU and further data transfer is continued.
- Data starvation due to empty FIFO – If the FIFO becomes empty during a write data transmission, or if the card clock is stopped and the FIFO remains empty for data timeout clocks, then a data-starvation error is signaled to the BIU. The datapath continues to wait for data in the FIFO.

Data Receive

- Data timeout – During a read-data transfer, if the data start bit is not received before the number of clocks that are programmed in the timeout register, the datapath does the following.
  - Signals data-timeout error to the BIU
  - Terminates further data transfer
  - Signals data transfer done to BIU

- Data start bit error – During a 4-bit or 8-bit read-data transfer, when an SBE occurs, the application or driver must issue CMD12 for SD/MMC cards and CMD52 for a SDIO card to exit the error condition. After a CMD done is received, the application or driver must reset IDMAC and CIU (if required) to clear the condition. The FIFO must be reset before issuing any data transfer commands in general.
- Data CRC error – During a read-data-block transfer, if the received CRC16 does not match with the internally-generated CRC16, the datapath signals a data CRC error to the BIU and continues further data transfer.
- Data end-bit error – During a read-data transfer, if the end bit of the received data is not 1, the datapath signals an end-bit error to the BIU, terminates further data transfer, and signals to the BIU that the data transfer is done.
- Data starvation due to FIFO full – During a read data transmission and when the FIFO becomes full, the card clock is stopped. If the FIFO remains full for data timeout clocks, a data starvation error is signaled to the BIU. The datapath continues to wait for the FIFO to start to empty. (The data starvation by host timeout bit is set in the `MSI_ISTAT` register.)

NOTE: In an error situation, DTO generation depends on when the error has occurred, since the descriptors are closed after the error scenarios as follows.

- If the data remains in the FIFO when the DMA detects the error scenario, then a DTO is not generated.
- If the data is completely read out before the error occurs, then a DTO is generated. CMD12 ensures DTO generation in error situations; therefore, issuing CMD12 is recommended.

Error Handling

The MSI implements error checking. Errors are reflected in the `MSI_ISTAT` register and can be communicated to the software through an interrupt, or the software can poll these bits. On power-on, interrupts are disabled, and all the interrupts are masked (bits 31:0 of the `MSI_IMSK` register are all 0). The MSI module captures the following errors:

- Response and data timeout errors. For response timeout, software can retry the command. For a data timeout error, the MSI has not received the data start bit - either for the first block or the intermediate block - within the timeout period. Software can either retry the whole data transfer again or retry from a specified block onwards. By reading the contents of the `MSI_TCBCNT` register later, the software can decide how many bytes remain to be copied.
- Response errors. Set when an error is received during response reception. In this case, the response that copied in the response registers is invalid. Software can retry the command.
- Data errors. Set when error in data reception is observed. For example, this error can occur in the data CRC, when the start bit is not found, when the end bit is not found, and so on. These errors could be set for any block-first block, intermediate block, or last block. On receipt of an error, the software can issue a STOP or ABORT command and retry the command for either whole data or partial data.

- Hardware locked error. Set when the MSI cannot load a command issued by software. When software sets the `MSI_CMD.STARTCMD` bit, the MSI tries to load the command. If the command buffer is already filled with a command, this error is generated. The software then has to reload the command.
- FIFO underrun or overrun error. If the FIFO is full and DMA tries to write data into the FIFO, then an overrun error is set. Conversely, if the FIFO is empty and the DMA tries to read data from the FIFO, an underrun error is set.
- Data starvation by host timeout. Raised when the MSI is waiting for software intervention to transfer the data to or from the FIFO, but the software does not transfer within the stipulated timeout period. Under this condition and when a read transfer is in progress, the software must read data from the FIFO and create space for further data reception. When a transmit operation is in process, the software must fill data in the FIFO to start transferring data to the card.
- CRC Error on command. If a CRC error is detected for a command.

NOTE: During a multiple-block data transfer, if a negative CRC status is received from the device, the datapath signals a data CRC error to the BIU by setting the data `MSI_ISTAT.DCRC` register. It then continues further data transmission until all the bytes are transmitted.

Fatal Bus Errors (FBE)

An FBE occurs due to an error response from SCB. This response is a system error, so the software driver must not perform any further programming of the MSI. The only recovery mechanism from such scenarios is to do one of the following.

- Issue a hard reset.
- Do a controller reset by writing to the `MSI_CTL.CTLRST` bit.

MSI Programming Model

This section provides procedures that allow programmers to use the MSI properly. These procedures include MSI initialization, single and multiple block reads and writes and many others.

MSI Programming Concepts

Using the features, operating modes, and event control for the MSI to their greatest potential requires an understanding of the following MSI-related concepts.

Software and Hardware Restrictions

Before issuing a new data transfer command, the software should ensure that the card is not busy due to any previous data transfer command. Before changing the card clock frequency, the software must ensure that there are no data or command transfers in progress.

To avoid glitches in the card clock outputs, use the following steps when changing the card clock frequency.

1. Before disabling the clocks, ensure that the card is not busy due to any previous data command. To determine this status, check for 0 in the `MSI_STAT.DBUSY` bit.
2. Update the `MSI_CLKEN` register to disable all clocks. To ensure completion of any previous command before this update, send a command to the CIU to update the clock registers by setting the `MSI_CMD.STARTCMD` bit, the `MSI_CMD.WTPRIVDATA` bits, and the `MSI_CMD.UCLKREGS` bit.
3. Set the `MSI_CMD.STARTCMD` bit to update the clock divider register, and send a command to the CIU in order to update the clock registers. Wait for the CIU to take the command.
4. Set the `MSI_CMD.STARTCMD` to update the `MSI_CLKEN` register to enable the required clocks and send a command to the CIU to update the clock registers. Wait for the CIU to take the command.

If the software issues a controller reset command by setting the `MSI_CTL.CTLRST` bit, all the CIU state machines are reset and the FIFO is not cleared. The DMA sends all remaining bytes to the host. In addition to a card-reset, if a FIFO reset is also issued (`MSI_CTL.FIFORST`), then:

- Any pending DMA transfer on the bus completes correctly
- DMA data reads are ignored
- Write data is unknown (x)

Additionally, if a DMA reset is also issued (`MSI_CTL.DMARST`), any pending DMA transfer is abruptly terminated.

If any of the previous data commands do not properly terminate, then the software must issue the FIFO reset. The reset removes any residual data, if any, in the FIFO. After asserting the FIFO reset, the program waits until this bit is cleared. One data transfer requirement between the FIFO and host is that the number of transfers must be a multiple of the FIFO data width. For example, if the FIFO data width = 32-bit and the program must write only 15 bytes to the card, the host must program the DMA to do 16-byte transfers to the card. (The program writes to the card using the `MSI_BYTCNT` register.)

Initializing the MSI

After the power-up, the MSI is reset. The reset initializes the registers, ports, FIFO-pointers, DMA interface controls, and state-machine. After power-on reset, the software performs the following steps which are reflected in the ***Data Transmit State Machine*** figure.

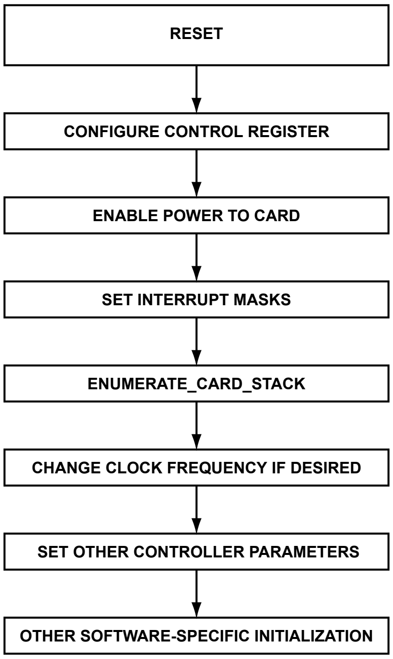

RESET
CONFIGURE CONTROL REGISTER
ENABLE POWER TO CARD
SET INTERRUPT MASKS
ENUMERATE_CARD_STACK
CHANGE CLOCK FREQUENCY IF DESIRED
SET OTHER CONTROLLER PARAMETERS
OTHER SOFTWARE-SPECIFIC INITIALIZATION

Figure 26-10: Data Transmit State Machine

1. Configure the control register (`MSI_CTL`).
2. Set masks for interrupts by clearing appropriate bits in the interrupt mask register. Set the global `MSI_CTL.INTEN` bit. It is recommended that programs write 0xFFFF_FFFF to the `MSI_ISTAT` register to clear any pending interrupts before setting the `MSI_CTL.INTEN` bit.
3. Enumerate the card stack. Each card is enumerated according to card type. For details, refer to Enumerating the Card Stack. For enumeration, restrict the clock frequency to 400 kHz in accordance with SD/MMC standards.
4. Set the card frequency using the `MSI_CLKDIV` register.
5. Set other parameters, which normally do not need changing with every command, with a typical value such as response and data timeout values. Set the values according to the SD/MMC specifications and the FIFO threshold value.

Enumerating the Card Stack

The card stack does the following:

- Enumerates the connected card
- Sets the RCA for the connected card
- Reads card-specific information
- Stores card-specific information locally

Each card is enumerated separately. The identification procedure depends on whether the card connected is SD, SDIO, or MMC.

Identifying Card Types

The MSI can have an MMC, SD, or SDIO type of card connected to it. All types of SDIO cards are supported; that is, SDIO_IO_ONLY, SDIO_MEM_ONLY, and SDIO_COMBO cards. Each card is enumerated separately. The enumeration sequence includes the following steps:

1. Check if the card is connected.
2. Clear the bits in the `MSI_CTYPE` register to configure the controller in 1-bit mode
3. Identify the card type; that is, SD, MMC, SDIO, or COMBO card.
    a. Send CMD5 with argument 0.

    b. Read the response register (`MSI_RESP0`), which gives the voltage window that the card supports-the response to CMD5 gives the voltages that the card supports.

    c. Send CMD5 again with the desired voltage window set in the argument. This CMD5 is used to set the voltage window and move the card state m/c out of the initialization state.

    d. Check the response [27] bit. If this bit = 1, this response indicates that the memory is present and the SDIO card is a COMBO card.

    e. If the card is SDIO go to Step 4. If the card is combinational logic (COMBO) or if the response is not received, continue with the following steps.

    f. Send CMD8 with the following argument:

    ```
    Bit[31:12] = 20'h0; // Reserved bits
    Bit[11:8] = 4'b0001 // VHS value
    Bit[7:0] = 8'b10101010 // Preferred Check Pattern by SD2.0
    ```

    g. If the response is received, the card supports high capacity SD2.0; send ACMD41 with the following argument:

    ```
    Bit[31] = 1'b0; // Reserved bits
    Bit[30] = 1'b1; // High Capacity Status
    Bit[29:25] = 6'h0; // Reserved bits
    Bit[24] = 1'b0;
    Bit[23:0] = Supported Voltage Range
    ```

    h. If the response is received for ACMD41, then the card is SD; otherwise the card is MMC.

    i. If the response is not received for initial CMD8, then the card does not support high capacity SD2.0. Issue CMD0 followed byMD41 with the following argument:

    ```
    Bit[31] = 1'b0; // Reserved bits
    Bit[30] = 1'b0; // High Capacity Status
    ```

```
Bit[29:24] = 6'h0; // Reserved bits
Bit[23:0] = Supported Voltage Range
```

j. If the response is received for ACMD41, then the card is SD. Otherwise, the card is MMC.

4. Enumerate the card according to the card type.
5. Use a card clock frequency = f_{OD} (that is, 400 kHz) and use the following enumeration command sequence:
   - SD card – Send CMD0, CMD8, ACMD41, CMD2, CMD3.
   - SDIO – Send CMD5; if the function count is valid, CMD3. For the SDIO memory section, follow the same commands as for the SD card.
   - MMC – Send CMD0, CMD1, CMD2, CMD3.

The card clock frequency can be configured after enumeration.

Programming Card Clocks

The MSI supports programming the desired operational frequency for the card. The clock for the card can also be enabled or disabled. Registers that support this feature are:

- `MSI_CLKDIV`: Programs clock source frequency.
- `MSI_CLKEN`: Enables or disables clock for the card and enables low-power mode, which automatically stops the clock to a card when the card is idle for more than 8 clocks.

The MSI loads each of these registers only when the `MSI_CMD.STARTCMD` bit and the `MSI_CMD.UCLKREGS` bit in the CMD register are set. When a command is successfully loaded, the MSI clears the `MSI_CMD.STARTCMD` bit. This operation happens unless the MSI already has another command in the queue, at which point it generates a Hardware Locked Error (HLE).

Software must look for the `MSI_CMD.STARTCMD` and the `MSI_CMD.UCLKREGS` bits, and set the `MSI_CMD.WTPRIVDATA` bit to ensure that clock parameters do not change during data transfer.

NOTE: Even though `MSI_CMD.STARTCMD` is set for updating clock registers, the MSI does not raise the `MSI_DONE` signal upon command completion.

Use the following procedure to program the clock-related registers.

1. Confirm that the card is not engaged in any transaction; if there is a transaction, wait until it finishes.
2. Stop all clocks by writing 0 to the `MSI_CLKEN` register.
3. Set the `MSI_CMD.STARTCMD`, `MSI_CMD.UCLKREGS`, and the `MSI_CMD.WTPRIVDATA` bits.
4. Wait until the `MSI_CMD.STARTCMD` bit is cleared or an HLE is set; in case of an HLE, repeat the command.
5. Program the `MSI_CLKDIV` register.

6. Set the `MSI_CMD.STARTCMD`, `MSI_CMD.UCLKREGS`, and `MSI_CMD.WTPRIVDATA` bits.
7. Wait until the `MSI_CMD.STARTCMD` is cleared or an HLE is set; in case of an HLE, repeat the command.
8. Re enable all clocks by programming the `MSI_CLKEN` register.
9. Set the `MSI_CMD.STARTCMD`, `MSI_CMD.UCLKREGS`, and the `MSI_CMD.WTPRIVDATA` bits.
10. Wait until the `MSI_CMD.STARTCMD` bit is cleared or a HLE is set; in case of a HLE, repeat the command.

Sending Non-Data Commands With or Without a Response Sequence

To send any non-data command, the software must program the `MSI_CMD` register and the `MSI_CMDARG` register with appropriate parameters. Using these two registers, the MSI forms the command and sends it to the command bus. The MSI reflects the errors in the command response through the error bits of the `MSI_ISTAT` register. The ***Command Register Settings for No-Data Command*** table shows the `MSI_CMD` bit settings.

When the MSI receives a response, either erroneous or valid, it sets the `MSI_ISTAT.CMDDONE` bit. A short response is copied in the `MSI_RESP0` register, while a long response is copied to all four response registers. The `MSI_RESP3` register bit 31 represents the MSB, and the `MSI_RESP0` register bit 0 represents the LSB of a long response.

Table 26-11: Command Register Settings for No-Data Command

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1/0	Choose value based on speed mode in use; refer to "MSI Register Descriptions"
update_clk_regs_only	0	No clock parameters update command
data_expected	0	No data command
card_number	*n*CardNo	Actual card number
cmd_index	command index	
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
user-selectable		
wait_prvdata_complete	0	Before sending a command on a command line, the host must wait for the completion of any data command in process, if any. (It is recommended to set this bit always, unless the current command is to query status or stop data transfer when transfer is in progress)
check_response_crc	1	If host crosschecks, CRC of response received

Use the following procedure to issue basic commands or non-data commands.

1. Program the `MSI_CMD` register with the appropriate command argument parameter.
2. Program the `MSI_CMD` register with the settings in the ***Command Register Settings for No-Data Command*** table.
3. Wait for command acceptance by host. When software loads the command into the MSI:
   - The MSI accepts the command for execution and clears the `MSI_CMD.STARTCMD` bit. This process occurs unless one command is in process. Then, the MSI can load and keep the second command in the buffer.
   - If the MSI is unable to load the command, then it generates an HLE (hardware locked error). (For example, a command is already in progress, a second command is in the buffer, and a third command is attempted).
4. Check if there is an HLE.
5. Wait for command execution to complete. After receiving either a response from a card or response timeout, the MSI sets the `MSI_ISTAT.CMDDONE` bit. Software can either poll for this bit or respond to a generated interrupt.
6. Check if the response timeout error, response CRC error, or response error is set. This confirmation can be done either by responding to an interrupt raised by these errors or by polling bits RE, RCRC, and RTO from the `MSI_ISTAT` register. If no response error is received, then the response is valid. If necessary, the software can copy the response from the response registers. Software must not modify clock parameters while a command is executing.

Single-Block or Multiple-Block Read

Use the following procedure to perform a single-block or multiple-block read.

1. Write the data size in bytes in the `MSI_BYTCNT` register.
2. Write the block size in bytes in the `MSI_BLKSIZ` register.

   ADDITIONAL INFORMATION: The MSI expects data from the card in blocks of size BLKSIZ each.
3. If card has a round-trip delay of more than 0.5 card clock period, program the `MSI_CDTHRCTL.RDTHR` bit field. The programming ensures that the card clock does not stop in the middle of a block of data transferring from the card to the host.

   ADDITIONAL INFORMATION: For programming guidelines, refer to the Card Read Threshold section. If the card read threshold feature is not enabled for such cards, then the program must ensure that the Rx FIFO does not become full during a read data transfer. The Rx FIFO must drain out at a rate faster than the rate that data is pushed into the FIFO.

4. Program the MSI_CMDARG register with the data address of the beginning of a data read. Program the MSI_CMD register with the parameters listed in the ***Command Register Settings for Single-Block or Multiple-Block Read*** table.

   ADDITIONAL INFORMATION: For SD and MMC cards, use CMD17 for a single-block read and CMD18 for a multiple-block read. For SDIO cards, use CMD53 for both single-block and multiple-block transfers.

   After writing to the MSI_CMD register, the MSI starts executing the command. When the command is sent to the bus, the CMDONE interrupt is generated.

5. Software looks for data error interrupts on bits 7, 9, 13, and 15 of the MSI_ISTAT register. If required, software can terminate the data transfer by sending a STOP command.

6. Software or the DMA looks for a receive FIFO data request or data starvation by host timeout conditions. In both cases, the DMA reads data from the FIFO and makes space in the FIFO for receiving more data.

When all the data is received, a DTO (Data Transfer Over) interrupt is generated.

Table 26-12: Command Register Settings for Single-Block or Multiple-Block Read

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1/0	Choose value based on speed mode in use; refer to “MSI Register Descriptions”
update_clk_regs_only	0	No clock parameters update command
card_number	*n*CardNo	Actual card number
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
send_auto_stop	0 or 1	See Auto-Stop
transfer_mode	0	Block transfer
read_write	0	Read from card
data_expected	1	Data command
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
user-selectable		
cmd_index	command index	Can be 1 for commands to stop data transfer, such as CMD12
wait_prvdata_complete	1	0 = sends command immediately. 1 = sends command after previous data transfer ends

Table 26-12: Command Register Settings for Single-Block or Multiple-Block Read (Continued)

Parameter	Value	Comment
check_response_crc	1	0 = MSI does not check response CRC 1 = MSI checks response CRC

Single-Block or Multiple-Block Write

Use the following procedure to perform a single-block or multiple-block write.

1. Write the data size in bytes in the `MSI_BYTCNT` register.
2. Write the block size in bytes in the `MSI_BLKSIZ` register.

   ADDITIONAL INFORMATION: The MSI sends data in blocks of size BLKSIZ each.
3. Program `MSI_CMDARG` register with the data address to which data is written.
4. Write data in the FIFO; it is best to start filling data the full depth of the FIFO.
5. Program the `MSI_CMD` register with the parameters listed the ***Command Register Settings for Single-Block or Multiple-Block Read*** table.

   ADDITIONAL INFORMATION: For SD and MMC cards, use CMD24 for a single-block write and CMD25 for a multiple-block write. For SDIO cards, use CMD53 for both single-block and multiple-block transfers. After writing to the `MSI_CMD` register, the MSI starts executing a command; when the command is sent to the bus, a CMDONE interrupt is generated.
6. Software looks for data error interrupts in the `MSI_ISTAT.DCRC`, `MSI_ISTAT.DRTO`, and `MSI_ISTAT.EBE` bits. If necessary, software can terminate the data transfer by sending the STOP command.
7. Software looks for a transmit FIFO data request or timeout conditions from data starvation by the host. In both cases, the software or the DMA writes data into the FIFO.
8. When a DTO interrupt is received, the data command is over. For an open-ended block transfer, if the byte count is 0, the software must send the STOP command. If the byte count is not 0, then on completion of a transfer of a given number of bytes, the MSI sends the STOP command, if necessary. The `MSI_ISTAT.ACD` bit reflects the completion of the AUTO-STOP command. A response to AUTO_STOP is stored in the `MSI_RESP1` register.
9. Wait for the busy clear interrupt.

The card can drive the busy clear interrupt on the DAT line; the host controller generates the interrupt after the busy is completed.

Table 26-13: Command Register Settings for Single-Block or Multiple-Block Write

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1 or 0	Choose value based on speed mode in use; refer to "MSI Register Descriptions"
update_clk_regs_only	0	No clock parameters update command
card_number	*n*CardNo	Actual card number
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
send_auto_stop	0 or 1	See Auto-Stop
transfer_mode	0	Block transfer
read_write	1	Write to card
data_expected	1	Data command
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
User-selectable		
cmd_index	command index	Can be 1 for commands to stop data transfer, such as CMD12
wait_prvdata_complete	1	0 – sends command immediately 1 – sends command after previous data transfer ends
check_response_crc	1	0 – MSI does not check response CRC 1 – MSI checks response CRC

Stream Reads and Writes

Stream reads and writes are like the block reads and writes described in the previous sections except for the following bits in the `MSI_CMD.XFRMODE` register:

- For reads, the transfer mode (`MSI_CMD.XFRMODE` bit) =1 and the command index =CMD20.
- For writes, the transfer mode (`MSI_CMD.XFRMODE` bit) =1 and the command index =CMD11.

In a stream transfer, if the byte count is 0, then the software must send the STOP command. If the byte count is not 0, when a given number of bytes completes a transfer, the MSI sends the STOP command

Packed Commands

In order to reduce overhead, read and write commands can be packed in groups of commands. The groups are either all read or all write. The software transfers the data for all commands in the group in one transfer on the bus.

Packed commands can be of two types:

- Packed write – CMD23 > CMD25
- Packed read – CMD23 > CMD25 >CMD23 > CMD18

The application software puts packed commands in packets. The packets are transparent to the core. For more information on packed commands, refer to the eMMC specification.

Sending Stop or Abort in Middle of Transfer

The STOP command can terminate a data transfer between a memory card and the MSI. The ABORT command can terminate an I/O data transfer for only the SDIO_IOONLY and SDIO_COMBO cards.

- Send STOP command. Can be sent on the command line while a data transfer is in-progress. This command can be sent at any time during a data transfer. For information on sending this command, refer to Sending Non-Data Commands With or Without a Response Sequence. Programs can use an extra setting for this command to set the `MSI_CMD` register bits (5–0) to CMD12 and set the `MSI_CMD.STPABORTCMD` bit to 1. If the `MSI_CMD.STPABORTCMD` bit is not set to 1, the MSI does not know that the program stopped a data transfer. Reset the `MSI_CMD.WTPRIVDATA` bit to 0 in order to make the MSI send the command at once, even though there is a data transfer in progress.
- Send ABORT command. Can be used with only an SDIO_IOONLY or SDIO_COMBO card. To abort the function that is transferring data, program the function number in ASx bits (CCCR register of card, address 0x06, bits (0–2)) using CMD52. This command is a non-data command. For information on sending this command, refer to Sending Non-Data Commands With or Without a Response Sequence.

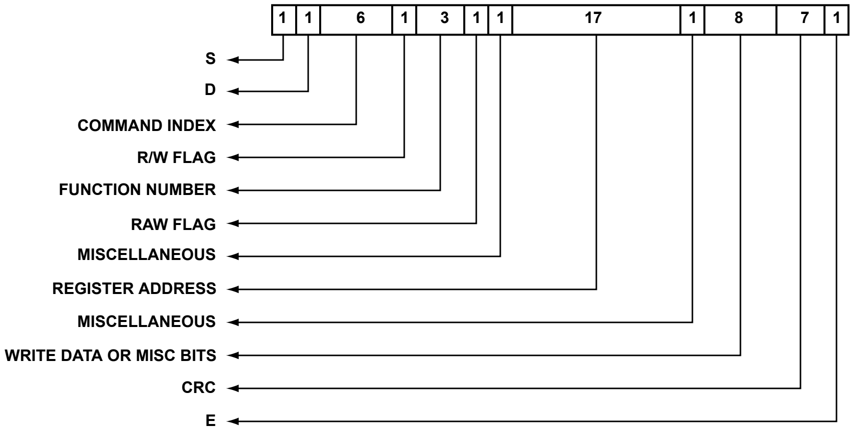


Figure 26-11: Command Format for CMD52

1. Program the `MSI_CMDARG` register with the appropriate command argument parameters listed in the ***Parameters for CMDARG Register*** table.
2. Program the `MSI_CMD` register using the command index as CMD52. Similar to the STOP command, set the `MSI_CMD.STPABORTCMD` bit =1, to inform the MSI that the program aborted the data transfer. Reset the

`MSI_CMD.WTPRIVDATA` bit =0 in order to make the MSI send the command at once, even though a data transfer is in progress.

3. Wait for command transfer over.
4. Check response (R5) for errors.

Table 26-14: Parameters for CMDARG Register

MSI_CMDARG Bits	Contents	Value
31	R/W flag	1
30–28	Function number	0 = for CCCR access
27	RAW flag	1 = if needed to read after write
26	Do-not-care	N/A
25-9	Register address	0x06
8	Do-not-care	N/A
7–0	Write data	Function number to be aborted

Suspend or Resume Sequence

In an SDIO card, the data transfer between an I/O function and the MSI can be temporarily halted using the SUSPEND command. This command can be necessary to perform a high-priority data transfer with another function. When desired, the data transfer can be resume using the RESUME command.

The following functions can be implemented by programming the appropriate bits in the CCCR register of the SDIO card. To read from or write to the CCCR register, use the CMD52 command.

1. SUSPEND data transfer – Non-data command.
    a. Check if the SDIO card supports the SUSPEND or RESUME protocol; this verification can be done through the SBS bit in the CCCR @0x08 register of the card.
    b. Check if the data transfer for the required function number is in process; the function number that is currently active is reflected in bits 0–3 of the CCCR register @0x0D. If the BS (bus status) bit is 1, then only the function number given by the FSx bits is valid.
    c. To suspend the transfer, set BR (bit 2) of the CCCR register @0x0C.
    d. Poll for clear status of bits BR (bit 1) and BS (bit 0) of the CCCR @0x0C. The BS bit is 1 when the currently selected function uses the data bus; the BR (bus release) bit remains 1 until the bus release is complete. When the BR and BS bits =0, the data transfer from the selected function has been suspended.
    e. During a read-data transfer, the MSI can be waiting for the data from the card. If the data transfer is a read from a card, then the MSI must be informed after the successful completion of the SUSPEND command. The MSI then resets the data state machine and comes out of the wait state. To achieve this state, set the `MSI_CTL.RDABORT` bit.

   f. Wait for data completion. Get pending bytes to transfer by reading the `MSI_TCBCNT` register.
   g. For a write data transfer, wait for the busy clear interrupt after the Data Transfer Over (DTO) interrupt.

2. RESUME data transfer – This command is a data command.
   a. Check that the card is not in a transfer state, which confirms that the bus is free for data transfer.
   b. If the card is in a disconnect state, select it using CMD7. The card status can be retrieved in response to CMD52/CMD53 commands.
   c. Check that a function to be resumed is ready for data transfer. Confirm the readiness by reading the RFx flag in CCCR @0x0F. If RF =1, then the function is ready for data transfer.
   d. To resume the transfer, use CMD52 to write the function number at the FSx bits (0–3) in the CCCR register @0x0D. Form the command argument for CMD52 and write it in the `MSI_CMDARG` register. The ***Parameters for CMDARG Register*** table lists the bit values.
   e. Write the block size in the `MSI_BLKSIZ` register. Data transfers in units of this block size.
   f. Write the byte count in the `MSI_BYTCNT` register. This amount is the total size of the data; that is, the remaining bytes for transfer. It is the responsibility of the software to handle the data.
   g. Program the `MSI_CMD` register similarly to a block transfer. For details, refer to the block read and write sections.
   h. When the `MSI_CMD` register is programmed, the command is sent and the function resumes data transfer. Read the DF flag (Resume Data Flag). If it is 1, then the function has data for the transfer and begins a data transfer as soon as the function or memory is resumed. If it is 0, then the function has no data for the transfer.
   i. If the DF flag is 0, the MSI waits for data for a read. After the data timeout period, it gives a data timeout error.

Table 26-15: Parameters for CMDARG Register

CMDARG Bits	Contents	Value
31	R/W flag	1
30–28	Function number	0 = for CCCR access
27	RAW flag	1 = read after write
26	Do-not-care	N/A
25-9	Register address	0x0D
8	Do-not-care	N/A
7–0	Write data	Function number that is to be resumed

Read Wait Sequence

Read_wait is used only with SDIO cards. It can temporarily stall the data transfer either from function or memory and allow the host to send commands to any function within the SDIO device. The host can stall this transfer for as long as required. The MSI provides the facility to signal this stall transfer to the card.

1. Check if the card supports the read_wait facility; read SRW (bit 2) of the CCCR register in SDIO card. If this bit is 1, then all functions in the card support the read_wait facility. Use CMD52 to read this bit.
2. If the card supports the read_wait signal, then assert it by setting the `MSI_CTL.RDWAIT` bit.
3. Clear the `MSI_CTL.RDWAIT` bit.

Card Read Threshold

When an application must perform a single or multiple block read command, the application must program the `MSI_CDTHRCTL` register with the appropriate card read threshold size. It also must set the card `MSI_CDTHRCTL.RDTHREN` (read threshold enable) bit. This additional programming ensures that the controller sends a read command only if there is space equal to the `MSI_CDTHRCTL.RDTHR` (card read threshold) available in the Rx FIFO. This programming in turn ensures that the card clock is not stopped in the middle a block of data being transmitted from the card. The card read threshold can be set to the block size of the transfer. This size guarantees that there is a minimum of one block size of space in the Rx FIFO before the controller enables the card clock.

Recommended Usage Guidelines for Card Read Threshold

1. Program the `MSI_CDTHRCTL` register before the programming the CMD register for a data read command.
2. Do not program the `MSI_CDTHRCTL` register when a data transfer command is in progress.
3. Program the `MSI_CDTHRCTL.RDTHR` value greater than or equal to the block size of the read transfer. This programming ensures that the card clock does not stop in between a block of data.
4. If round-trip delay > 0.5 card clock, then enable the card read threshold and program the card threshold as per guideline #3. This programming guarantees that the card clock does not stop in between a block of data. The controller samples data incorrectly if the card clock stops in between a block of data if round-trip delay > 0.5 card clock.
5. `MSI_CDTHRCTL.RDTHR` is greater than or equal to `MSI_BLKSIZ` (recommended). Program the card read threshold size (`MSI_CDTHRCTL.RDTHR`) to at least 1 × BlockSize of the multi-block transfer. This programming guarantees that the card clock does not stop in between a block of data due to the Rx FIFO becoming full during the read transfer.
6. `MSI_CDTHRCTL.RDTHR` is less than `MSI_BLKSIZ`. If the `MSI_CDTHRCTL.RDTHR` bit is programmed to less than the `MSI_BLKSIZ` of the transfer, then the system must ensure that the receive FIFO never becomes full and overflows during the read transfer. This programming can cause the card clock from the MSI to stop. The MSI is not able to guarantee that the card clock does not stop during a read transfer.

NOTE: If the `MSI_CDTHRCTL.RDTHR`, `MSI_FIFOTH.RXWM`, and `MSI_FIFOTH.DMAMSZ` values are programmed incorrectly, then the card clock can stop indefinitely and no interrupts are generated from the controller.

Card Read Threshold Programming Sequence

Most cards, such as SDHC or SDXC, typically support block sizes that are specified in the card or are fixed to 512 bytes. For SDIO cards, standard capacity SD cards that support READ_BL_PARTIAL =1 and MMC cards, the block size is variable. The application can choose block size.

Use the following steps for the card read threshold feature. The steps guarantee that the card clock does not stop because of a FIFO full condition in the middle of a block of data being read from the card.

1. Choose the block size (configured using the `MSI_BLKSIZ` register). The block size requested by the application from the card for the read transfer card must be 32-bit aligned.
2. Enable the card read threshold feature. The card read threshold can be enabled only if the block size for the given transfer is less than the total depth of the FIFO (*BlkSiz* ≤ *FifoDepth*)

   Where: *BlkSiz* = (block size in bytes) × 8 ÷ 32; that is, the number of the block size in terms of FIFO locations

   FifoDepth = total number of FIFO locations.
3. Choose the card read threshold.
   - If *BlkSiz* ≥ ½ *FifoDepth*, configure `MSI_CDTHRCTL.RDTHR` such that the value ≤ *BlkSize* in bytes
   - If *BlkSiz* < ½ *FifoDepth*, configure `MSI_CDTHRCTL.RDTHR` such that the value = *BlkSize* in bytes
4. Choose the DMA multiple transaction size using the `MSI_FIFOTH.DMAMSZ` bit field. The possible values for the bit fields are 1, 4, 8, 16, 32, 64, 128, and 256 transfers. Choose the value of `MSI_FIFOTH.DMAMSZ` from the transfer values so that BlkSiz is a multiple of DMAMSZ. BlkSize% (DMAMSZ) = 0. Note the following special cases:
   - When an MSIZE transfer =1 (configured using the `DMA_CFG.MSIZE` bit field). The `MSI_FIFOTH.DMAMSZ` is equal to 1 when the block size chosen in Step 1 is not a multiple of the FIFO width (in bytes). If DMAMSZ =1 is not acceptable and a higher burst size is desired-that is, a higher DMAMSZ. Return to Step 1 and recalculate the block size.
   - Internal DMA (IDMAC). The size of the data buffer (in bytes) for each descriptor must be a multiple of DMAMSZ × 4.
5. Choose the RX watermark using the `MSI_FIFOTH.RXWM` bit field.

   If DMAMSZ = 1, then the RXWM = 1 or RXWM = BlkSize – 1

   Additionally, for all DMA modes the RXWM must be a multiple of the chosen DMAMSZ.

MSI Programming Model, Boot Operation

This section discusses details on programming the MSI for boot operation.

Normal Boot Operation

Normal boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards. It is performed in push-pull mode. The *Normal Boot Timing* figure shows the timing for the transmit state machine in a normal boot mode.

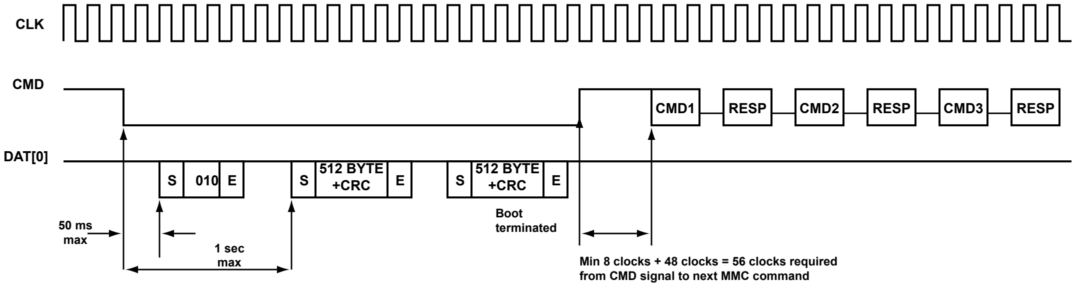


Figure 26-12: Normal Boot Timing

Normal Boot Operation; eMMC

Normal boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards. It must be performed in push-pull mode. See the *Normal Boot Timing* figure.

Software must adhere to the following steps when working with eMMC or removable MMC 4.3 cards for boot operation. Note that the software knows that the card supports boot operation (the BOOT_PARTITION_ENABLE bit set) in the card.

The software also knows the BOOT_SIZE_MULT value in the card and the data bus width to use during boot operation: cExtend CSD register byte [177] bit[0:1].

1. Set the masks for interrupts by clearing appropriate bits in the `MSI_IMSK` register.
2. Set the `MSI_CTL.INTEN` bit. Before this bit is set, programs should write 0xFFFF_FFFF to the `MSI_ISTAT` and `MSI_IDSTS` registers to clear any pending interrupts. For internal DMAC, unmask all the relevant fields in the `MSI_IDINTEN` register.
3. Set the and `MSI_CTL.INTEN` bits =1. In order to use the internal DMAC to transfer the boot data received, it must also set up the descriptors and program the `MSI_CTL.INTDMAC` bit =1.
4. Set the card frequency to 400 kHz using the clock-divider register.
5. Set Data Time Out = $(10 \times ((TAAC \times f_{OP}) + (100 \times NSAC))$; this is NAC.
6. Program the `MSI_BLKSIZ` register with 0x200 (512 bytes).
7. Program the `MSI_BYTCNT` register with multiples of 128K bytes, as indicated by the BOOT_SIZE_MULT value in the card.

8. Program the Rx FIFO threshold value in bytes in the `MSI_FIFOTH` register. Typically, the threshold value can be set to half the FIFO depth.
9. Program the following fields: `MSI_CMD.STARTCMD` =1'b1, `MSI_CMD.BOOTEN` =1'b1, `MSI_CMD.XPECTBOOTACK` (for start-acknowledge pattern ACK from the card) and `MSI_CMD.DXPECT` =1'b1.
10. If `MSI_CMD.XPECTBOOTACK` =1'b1, the software driver must start a timer after step 9; the terminal value is 50 ms.
    - Before this timer elapses, the BAR interrupt should be received from the MSI. If this action does not occur, program the CMD register as follows:

      `MSI_CMD.STARTCMD` =1'b1

      `MSI_CMD.BOOTDIS` =1'b1

      All other fields = 0
    - The MSI generates a Command Done (CD) interrupt after de-asserting the CMD line of the card. In IDMAC:

      Descriptor is closed

      `MSI_IDSTS.CES` =1, indicating BAR timeout

      `MSI_IDSTS.RI` =0
    - If the BAR interrupt is received, the software should clear this interrupt by writing a 1 to it. The software should then start another timer with a terminal value of 1 – 0.05 = 0.95 seconds. Before this timer elapses, the BDS interrupt should be received from the MSI. If this action does not occur, the software driver must program the CMD register as follows:

      `MSI_CMD.STARTCMD` =1'b1

      `MSI_CMD.BOOTDIS` =1'b1

      All other fields =0
    - The MSI generates a CD interrupt after de-asserting the CMD line of the card. In IDMAC:

      Descriptor is closed

      `MSI_IDSTS.CES` =1, indicating BDS timeout

      `MSI_IDSTS.RI` =0
    - If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed RX_WMark level is attained. At the end of a successful boot data transfer from the card, the following interrupts are generated:

      Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit

Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit

Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit

- If an error occurs in Boot Ack pattern (010) or an end bit error occurs

  Controller automatically aborts boot by pulling CMD line high

  Controller generates CD interrupt

  Controller does not generate BAR interrupt

  Application aborts boot transfer

- In IDMAC:

  If the software creates more descriptors than the boot data received, the MSI does not close the extra descriptors.

  If the software creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

- If between data block transfers NAC is violated, DRTO (Data Read Timeout) is asserted. If there are errors associated with start or end bits, SBE/EBE interrupts are also generated.

11. If `MSI_CMD.XPECTBOOTACK` =1'b0, the software should start a timer after the step 9 where the terminal value is 1 second.

    - Before this timer elapses, a BDS interrupt should be received from the MSI. If the interrupt is not received, the software must program their CMD register with the following fields:

      `MSI_CMD.STARTCMD` =1'b1

      `MSI_CMD.BOOTDIS` =1'b1

      All other fields =0

    ADDITIONAL INFORMATION: MSI generates a CD interrupt after de-asserting the CMD line of the card. In IDMAC mode, the descriptor is closed and the `MSI_IDSTS.CES` bit =1, indicating a BDS timeout.

    - If a BDS interrupt is received, it indicates that the boot data is being received from the card. At the end of a successful boot data transfer from card, the following interrupts are generated.

      Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit

      Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit

      Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit

Normal Boot Operation; Removable MMC4.3, MMC4.4, and MMC4.41 Cards

Removable MMC4.3, MMC4.4, and MMC4.41 cards are different than eMMC in that the software is not aware whether these cards support the boot mode of operation when plugged in. Thus, the software must:

- Enumerate these cards as it would enumerate MMC4.0/4.1/4.2 cards for the first time
- Know the card characteristics
- Decide whether to perform a boot operation or not

Use the following procedure when booting with these card types,

1. Enumerate the card.
2. Read the EXT_CSD register of the card and examine the following fields:
    - BOOT_PARTITION_ENABLE
    - BOOT_SIZE_MULT
    - BOOT_INFO
3. If necessary, the controller can manipulate the boot information in the card.
4. If the controller must perform a boot operation at the next power-up cycle, it can manipulate the EXT_CSD register contents by using a SWITCH (CMD6) command.
5. From this point, use the same steps as in Normal Boot Operation; eMMC.

Alternate Boot Operation; eMMC

The alternate boot operation differs from the normal boot operation in that CMD0 is used to boot the card rather than holding down the CMD-line of the card. The alternate boot operation can occur only if bit 0 in the extended CSD byte[228] (BOOT_INFO) is set to 1.

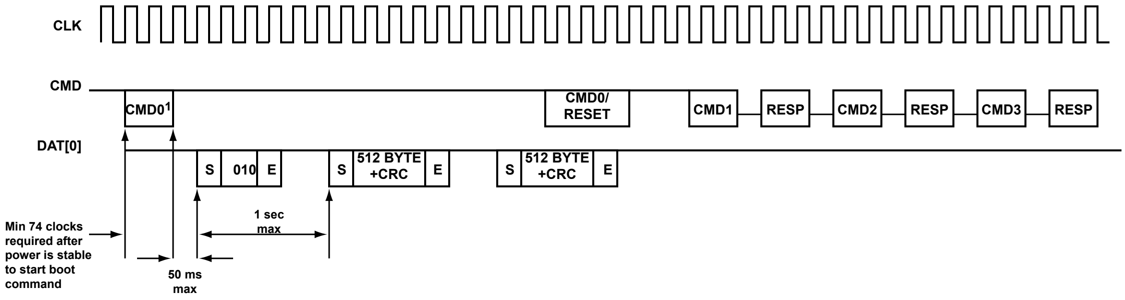

CLK
CMD
CMD0[1]
CMD0/
RESET
CMD1
RESP
CMD2
RESP
CMD3
RESP
DAT[0]
S
010
E
S
512 BYTE
+CRC
E
S
512 BYTE
+CRC
E
Min 74 clocks
required after
power is stable
to start boot
command
50 ms
max
1 sec
max

Figure 26-13: Alternate Boot Timing

Use the following procedure when working with eMMC or removable MMC 4.3 cards for the alternative boot operation. The software is aware that the card supports the Alternate Boot operation—the BOOT_INFO bit is set in the card. Also, the software is aware of the BOOT_SIZE_MULT value in the card and the data bus width to use during the boot operation-extend CSD register byte[177] bit[0:1].

1. Set the masks for interrupts by clearing appropriate bits in the Interrupt Mask register.
2. Set the `MSI_CTL.INTEN` bit =1.

ADDITIONAL INFORMATION: Set the `MSI_CTL.INTEN` bit. Before this bit is set, programs should write 0xFFFF_FFFF to the `MSI_ISTAT` and `MSI_IDSTS` registers to clear any pending interrupts. Software also must unmask all the relevant fields in `MSI_IDINTEN` register.

3. Configure the following bits in the control register.
    a. `MSI_CTL.INTEN` = 1'b1
    b. Other fields are 1'b0
    c. For IDMAC, set up the descriptors.
    d. Program the `MSI_CTL.INTDMAC` bit to 1
4. Set the card frequency to 400 kHz using the clock-divider register

    ADDITIONAL INFORMATION: Wait for a time that ensures that at least 74 card clock cycles have occurred on the card interface.
5. Set Data Time Out = (10 × ((TAAC × f_{OP}) + (100 × NSAC)); this is NAC
6. Program the `MSI_BLKSIZ` register with 0x200 (512 bytes).
7. Program the `MSI_BYTCNT` register with multiples of 128K bytes, as indicated by the BOOT_SIZE_MULT value in the card
8. Program the Rx FIFO threshold value in bytes in the `MSI_FIFOTH` register. Typically, the threshold value is set to half the FIFO depth; that is, `MSI_FIFOTH.RXWM` = (FIFO_DEPTH/2) – 1.
9. Program CMDARG = 0xFFFFFFFA.
10. Program the following fields.
    a. `MSI_CMD.STARTCMD` = 1'b1
    b. `MSI_CMD.BOOTMODE`= 1'b1
    c. `MSI_CMD.XPECTBOOTACK` depending on whether a start-acknowledge pattern is expected from the card.
    d. `MSI_CMD.DXPECT` = 1'b1
    e. `MSI_CMD.INDX` = 0
    f. Remainder of `MSI_CMD` register fields = 1'b0

    ADDITIONAL INFORMATION: Wait for the Command Done (CD) interrupt.
11. If `MSI_CMD.XPECTBOOTACK` = 1'b1 in step 10, the software driver must start a timer with a terminal value of 50 ms.
    a. Before this timer elapses, the BAR interrupt should be received from the MSI. If this action does not occur, the software must infer that the start-pattern has not been received and must discontinue the boot process and start with normal enumeration. In IDMAC:

  - Descriptor is closed
  - `MSI_IDSTS.CES` =1, indicating BAR timeout
  - `MSI_IDSTS.RI` =0

b. If the BAR interrupt is received, the software driver should clear this interrupt by writing a 1 to it. The software driver then starts another timer with a terminal value of 1 - 0.05 = 0.95 seconds. Before this timer elapses, the BDS interrupt should be received from the MSI. If this action does not occur, the software driver discontinues the boot process and start with normal enumeration.

  - Descriptor is closed
  - `MSI_IDSTS.CES` =1, indicating BDS timeout
  - `MSI_IDSTS.RI` =0

c. If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed `MSI_FIFOTH.RXWM` level is hit.

d. It is the responsibility of the software driver to terminate the boot operation by programming the MSI to send a CMD0 by programming the registers `MSI_CMDARG` =0 and the command register bits `MSI_CMD.STARTCMD` =1, `MSI_CMD.INDX` =0, all_other_fields = 0.

e. At the end of a successful boot data transfer from the card, the following status bits are set:

  - Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit
  - Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit
  - Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit

f. If an error occurs in Boot Ack pattern (010) or an end bit error occurs:

  - Controller does not generate BAR interrupt
  - Controller detects boot data start and generates BDS interrupt
  - Controller continues to receive boot data
  - Application must abort boot after receiving BDS interrupt

g. In IDMAC:

  - If the software driver creates more descriptors than the boot data received, the extra descriptors are not closed by the MSI.
  - If the software driver creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

h. If between data block transfers NAC is violated, DRTO (Data Read Timeout) is asserted. Apart from this, if there are errors associated with Start/End bits, SBE/EBE interrupts are also generated.

12. If `MSI_CMD.XPECTBOOTACK` =1'b0 in Step 10, the software should start a timer after Step 10with a terminal value of 1 second.
    a. Before this timer elapses, the BDS interrupt should be received from the MSI. If this does not occur, the software driver should discontinue the boot process and start with normal enumeration. In IDMAC:
        - Descriptor is closed
        - `MSI_IDSTS.CES` =1, indicating BDS timeout
        - `MSI_IDSTS.RI` =0
    b. If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed `MSI_FIFOTH.RXWM` level is hit.
    c. It is the responsibility of the software to terminate the boot operation. Software programs the MSI to send a CMD0 through the `MSI_CMDARG` register =0 and command register bits `MSI_CMD.STARTCMD` =1, `MSI_CMD.INDX` =0, and all_other_fields =0.
    d. At the end of a successful boot data transfer from card, the following interrupts are generated.
        - Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit
        - Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit
        - Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit
    e. In IDMAC
        - If the software driver creates more descriptors than the boot data received, the MSI does not close the extra descriptors.
        - If the software driver creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

Alternate Boot Operation; Removable MMC4.3 Card

Removable MMC4.3 cards are different than eMMC in that software is not aware whether these cards support the boot mode of operation. The software must:

- Enumerate these cards as it would enumerate MMC4.0/4.1/4.2 cards for the first time
- Know the card characteristics
- Decide whether to perform a boot operation

Use the following procedure when working with removable MMC 4.3 cards for the Alternative Boot operation.

1. Enumerate the card.
2. Read the EXT_CSD register of the card and examine the following fields:

   a. BOOT_PARTITION_ENABLE
   b. BOOT_SIZE_MULT
   c. BOOT_INFO

3. If necessary, the controller can also manipulate the boot information in the card.
4. If the host controller must perform a boot operation at the next power-up cycle, it can manipulate the EXT_CSD register contents by using a SWITCH (CMD6) command.
5. From this point, use the same steps as in Alternate Boot Operation; eMMC.

ADSP-SC58x Product Specific Information

The ADSP-SC589 MSI pins support programmable Pull-up control to prevent the pins from floating. The on-chip pulls to Data and CMD lines are programmable.

- The pullup for MSI_DATA[3:0] and MSI_CMD is controlled by the bit PUMSIDLC in PADS_PCFG0
- The pullup for MSI_DATA[7:4] is controlled by PUMSIDH bit in PADS_PCFG0

The PADS_PCFG0 register in the GPIO chapter has more details on bit assignments.

ADSP-SC58x MSI Register Descriptions

Mobile Storage Interface (MSI) contains the following registers.

Table 26-16: ADSP-SC58x MSI Register List

Name	Description
MSI_BLKSIZ	Block Size Register
MSI_BUFADDR	Current Buffer Descriptor Address Register
MSI_BUSMODE	Bus Mode Register
MSI_BYTCNT	Byte Count Register
MSI_CDETECT	Card Detect Register
MSI_CDTHRCTL	Card Threshold Control Register
MSI_CLKDIV	Clock Divider Register
MSI_CLKEN	Clock Enable Register
MSI_CMD	Command Register
MSI_CMDARG	Command Argument Register
MSI_CTL	Control Register
MSI_CTYPE	Card Type Register
MSI_DBADDR	Descriptor List Base Address Register

Table 26-16: ADSP-SC58x MSI Register List (Continued)

Name	Description
MSI_DEBNCE	Debounce Count Register
MSI_DSCADDR	Current Host Descriptor Address Register
MSI_ENSHIFT	Enable Phase Shift Register
MSI_FIFOTH	FIFO Threshold Watermark Register
MSI_IDINTEN	Internal DMA Interrupt Enable Register
MSI_IDSTS	Internal DMA Status Register
MSI_IMSK	Interrupt Mask Register
MSI_ISTAT	Raw Interrupt Status Register
MSI_MSKISTAT	Masked Interrupt Status Register
MSI_PLDMND	Poll Demand Register
MSI_RESP0	Response Register 0
MSI_RESP1	Response Register 1
MSI_RESP2	Response Register 2
MSI_RESP3	Response Register 3
MSI_STAT	Status Register
MSI_TBBCNT	Transferred Host to BIU-FIFO Byte Count Register
MSI_TCBCNT	Transferred CIU Card Byte Count Register
MSI_TMOUT	Timeout Register

Block Size Register

The MSI_BLKSIZ register provides bits that configure data block sizes. Sizes supported are 1 to 65,535 bytes.

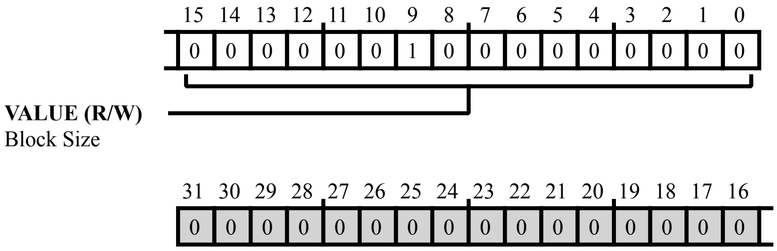

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Block Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-14: MSI_BLKSIZ Register Diagram

Table 26-17: MSI_BLKSIZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Block Size. The `MSI_BLKSIZ.VALUE` bit field configures data block size in bytes. Sizes supported are 1 to 65,535 bytes.

Current Buffer Descriptor Address Register

The `MSI_BUFADDR` register points to the data buffer address of the current descriptor read by the IDMAC.

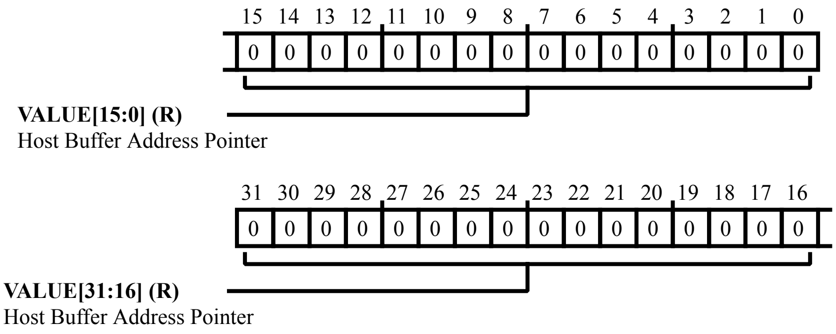

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Host Buffer Address Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Host Buffer Address Pointer

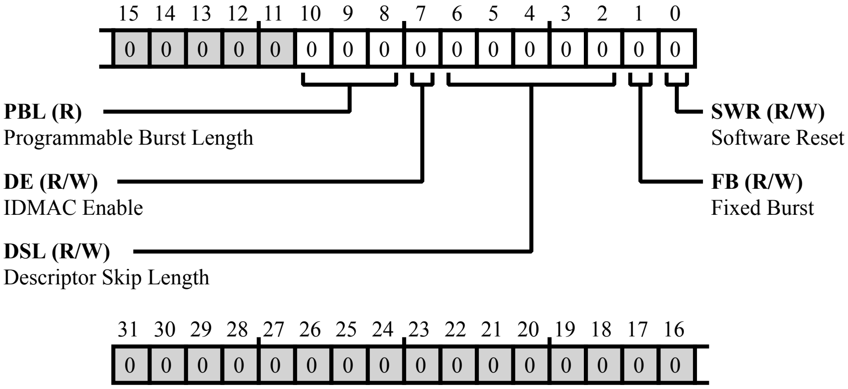
Figure 26-15: MSI_BUFADDR Register Diagram

Table 26-18: MSI_BUFADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Host Buffer Address Pointer. The `MSI_BUFADDR.VALUE` bit field points to the data buffer address of the current descriptor read by the IDMAC.

Bus Mode Register

The `MSI_BUSMODE` register provides bits that control the burst mode, descriptor skip length, and IDMAC. This register also provides a software reset bit.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PBL (R)
Programmable Burst Length
DE (R/W)
IDMAC Enable
DSL (R/W)
Descriptor Skip Length
SWR (R/W)
Software Reset
FB (R/W)
Fixed Burst
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-16: MSI_BUSMODE Register Diagram

Table 26-19: MSI_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/NW)	PBL	Programmable Burst Length. The `MSI_BUSMODE.PBL` bit field indicate the maximum number of beats to be performed in one IDMAC transaction. The IDMAC always attempts to burst as specified in PBL each time it starts a burst transfer on the host bus. This value is the mirror of the `MSI_FIFOTH.DMAMSZ` bits. In order to change this value, write the required value to `MSI_FIFOTH.DMAMSZ` bit field. The units for transfer are 32-bit.	
		0	1 transfer
		1	4 transfers
		2	8 transfers
		3	16 transfers
		4	32 transfers
		5	64 transfers
		6	128 transfers
		7	256 transfers
7 (R/W)	DE	IDMAC Enable. Setting the `MSI_BUSMODE.DE` bit enables the internal DMA interface.	
		0	Disable internal DMA
		1	Enable internal DMA

Table 26-19: MSI_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:2 (R/W)	DSL	Descriptor Skip Length. The `MSI_BUSMODE.DSL` bit field specifies the number of words to skip between two unchained descriptors. This is applicable only for dual buffer structures.	
1 (R/W)	FB	Fixed Burst. The `MSI_BUSMODE.FB` bit controls whether the peripheral bus master interface performs fixed burst transfers or not. When set, the peripheral bus uses only SINGLE, INCR4, INCR8 or INCR16 during the start of normal burst transfers. When reset, the peripheral bus uses SINGLE and INCR burst transfer operations.	
		0	Use SINGLE and INCR
		1	Use SINGLE, INCR4, INCR8 or INCR16
0 (R/W)	SWR	Software Reset. When the `MSI_BUSMODE.SWR` bit is set, the DMA controller resets all its internal registers. The `MSI_BUSMODE.SWR` bit is automatically cleared after 1 clock cycle.	
		0	No reset
		1	Reset internal registers

Byte Count Register

The MSI_BYTCNT register provides bits that configure the byte count to be transferred.

In SDIO mode, if a single transfer is greater than 4 bytes and non-DWORD-aligned, the transfer should be broken where only the last transfer is non-DWORD-aligned and less than 4 bytes. For example, if a transfer of 129 bytes must occur, then the driver should start at least two transfers; one with 128 bytes and the other with 1 byte.

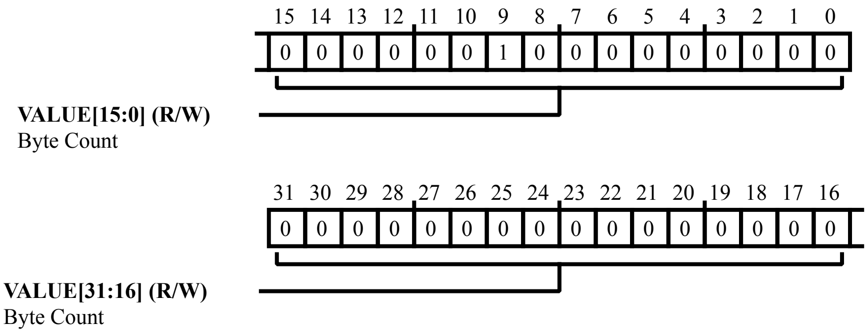

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Byte Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Byte Count

Figure 26-17: MSI_BYTCNT Register Diagram

Table 26-20: MSI_BYTCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Byte Count. The MSI_BYTCNT.VALUE bit field provides bits that configure the byte count to be transferred.

Card Detect Register

The `MSI_CDETECT` register configures the value on card_detect_n input ports (1 bit per card).

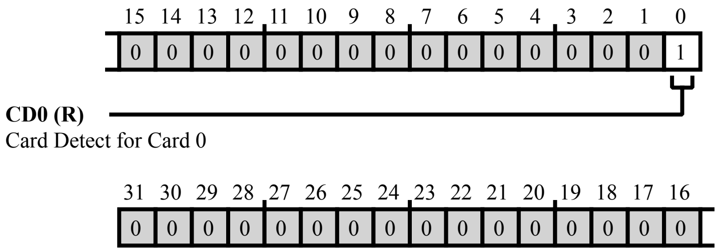

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
CD0 (R)
Card Detect for Card 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-18: MSI_CDETECT Register Diagram

Table 26-21: MSI_CDETECT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	CD0	Card Detect for Card 0. The `MSI_CDETECT.CD0` bit sets the value on card_detect_n input ports.

Card Threshold Control Register

The `MSI_CDTHRCTL` register sets the card read threshold size and enables card read threshold. This register also has a Busy Clear interrupt generation bit.

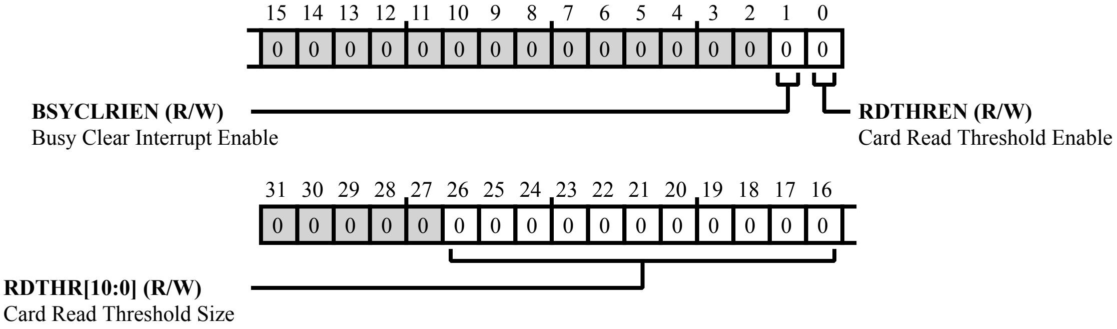

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BSYCLRIEN (R/W)
Busy Clear Interrupt Enable
RDTHREN (R/W)
Card Read Threshold Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDTHR[10:0] (R/W)
Card Read Threshold Size

Figure 26-19: MSI_CDTHRCTL Register Diagram

Table 26-22: MSI_CDTHRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26:16 (R/W)	RDTHR	Card Read Threshold Size. The `MSI_CDTHRCTL.RDTHR` bit configures the card read threshold size. For information on using this feature, see the "Card Read Threshold" section in the MSI chapter.	
1 (R/W)	BSYCLRIEN	Busy Clear Interrupt Enable. The `MSI_CDTHRCTL.BSYCLRIEN` bit indicates the completion of a busy driven by the card after a write data transfer.	
		0	Busy clear interrupt disabled
		1	Busy clear interrupt enabled
0 (R/W)	RDTHREN	Card Read Threshold Enable. The `MSI_CDTHRCTL.RDTHREN` bit enables the card read threshold size feature.	
		0	Card read threshold disabled
		1	Card read threshold enabled

Clock Divider Register

The `MSI_CLKDIV` register provides clock divider bit fields. The bit field value is: clock division = 2 x n.

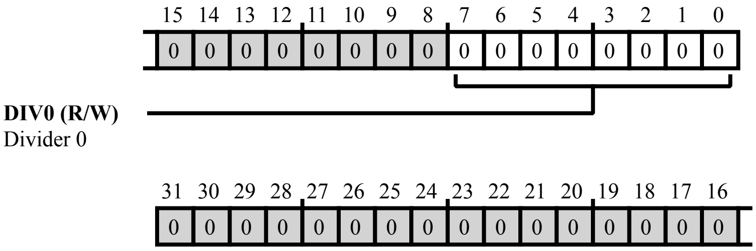

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIV0 (R/W)
Divider 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-20: MSI_CLKDIV Register Diagram

Table 26-23: MSI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	DIV0	Divider 0. The `MSI_CLKDIV.DIV0` bit field provides clock division. The value is 2 x n. For example, a value of 0 means divide by 2 x 0 = 0 (no division, bypass), a value of 1 means divide by 2 x 1 = 2, and a value of 0xFF means divide by 2 x 255 = 510, and so on.

Clock Enable Register

The `MSI_CLKEN` register enables clock control. This register also provides low-power control for the card clock.

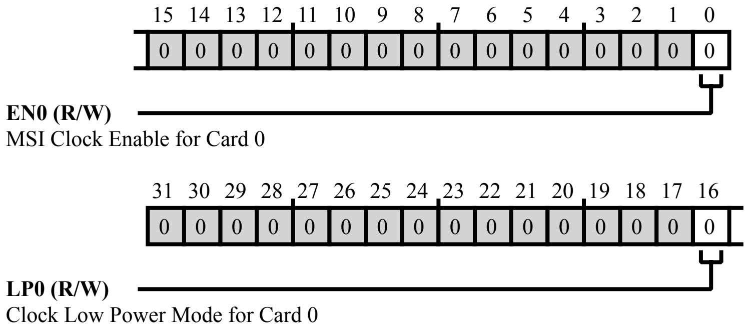

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN0 (R/W)
MSI Clock Enable for Card 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LP0 (R/W)
Clock Low Power Mode for Card 0

Figure 26-21: MSI_CLKEN Register Diagram

Table 26-24: MSI_CLKEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	LP0	Clock Low Power Mode for Card 0. Setting the `MSI_CLKEN.LP0` bit puts the card clock into low-power mode; it stops the clock when the card is in IDLE (it should be normally set to only MMC and SD memory cards. For SDIO cards, if interrupts must be detected, the clock should not be stopped). Clearing the `MSI_CLKEN.LP0` bit puts the cards into non-low power mode.
0 (R/W)	EN0	MSI Clock Enable for Card 0. The `MSI_CLKEN.EN0` bit is the clock-enable control for the card clock.

Command Register

The `MSI_CMD` register provides bits that configure various command parameters such as boot modes, data transfer modes, and command response settings.

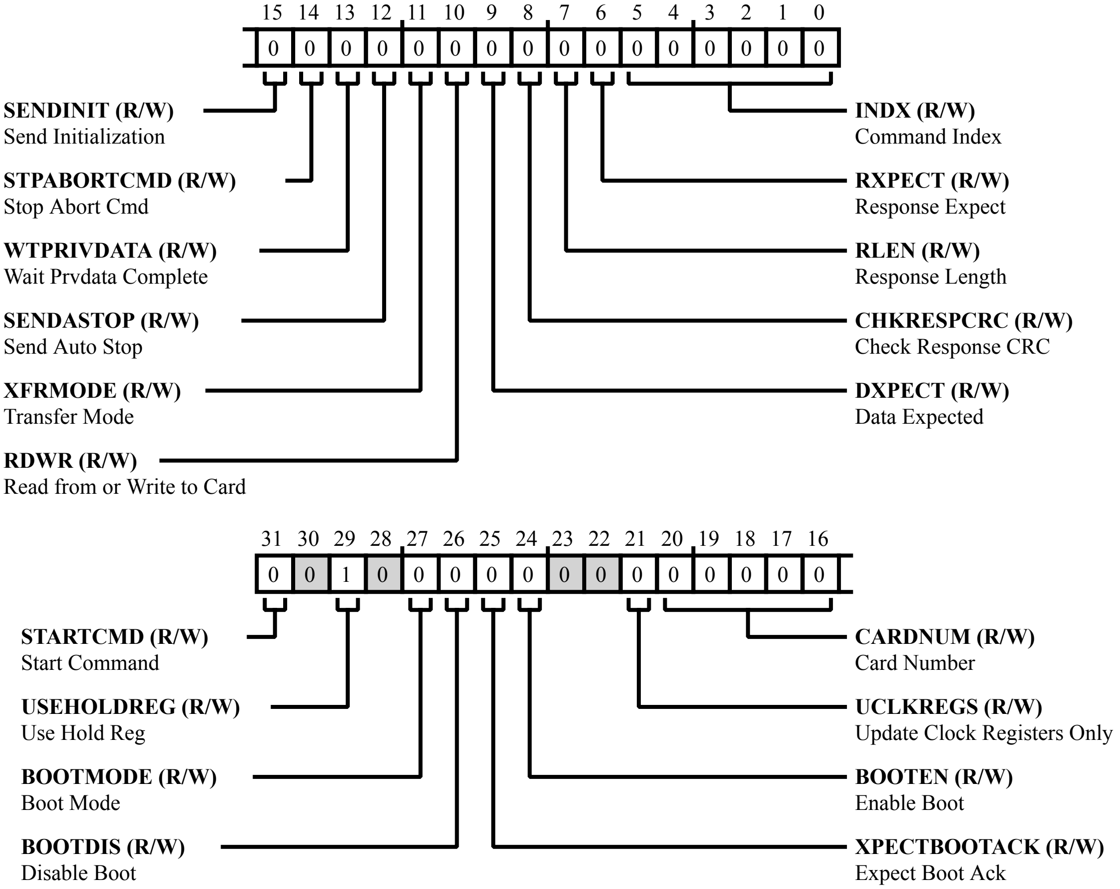

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SENDINIT (R/W)
Send Initialization
STPABORTCMD (R/W)
Stop Abort Cmd
WTPRIVDATA (R/W)
Wait Prvdata Complete
SENDASTOP (R/W)
Send Auto Stop
XFRMODE (R/W)
Transfer Mode
RDWR (R/W)
Read from or Write to Card
INDX (R/W)
Command Index
RXPECT (R/W)
Response Expect
RLEN (R/W)
Response Length
CHKRESPCRC (R/W)
Check Response CRC
DXPECT (R/W)
Data Expected
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
STARTCMD (R/W)
Start Command
USEHOLDREG (R/W)
Use Hold Reg
BOOTMODE (R/W)
Boot Mode
BOOTDIS (R/W)
Disable Boot
CARDNUM (R/W)
Card Number
UCLKREGS (R/W)
Update Clock Registers Only
BOOTEN (R/W)
Enable Boot
XPECTBOOTACK (R/W)
Expect Boot Ack

Figure 26-22: MSI_CMD Register Diagram

Table 26-25: MSI_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	STARTCMD	Start Command. Once the command is taken by CIU, the `MSI_CMD.STARTCMD` bit is cleared. When this bit is set, the host should not attempt to write to any command registers. If a write is attempted, a hardware lock error is set in the raw interrupt register. Once the command is sent and a response is received from the SD_MMC cards, the Command Done bit is set in the raw interrupt register.

Table 26-25: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	USEHOLDREG	Use Hold Reg. When the `MSI_CMD.USEHOLDREG` bit is set, CMD and DATA are sent to the card through the HOLD register. When the `MSI_CMD.USEHOLDREG` bit is cleared, CMD and DATA are sent to the card bypassing the HOLD register.	
		0	CMD and DATA sent to card bypassing the HOLD register
		1	CMD and DATA sent to card through the HOLD register
27 (R/W)	BOOTMODE	Boot Mode. When the `MSI_CMD.BOOTMODE` bit is set, the MSI has an alternate boot operation. When the `MSI_CMD.BOOTMODE` bit is cleared, the MSI has a mandatory boot operation.	
		0	Mandatory Boot
		1	Alternate Boot
26 (R/W)	BOOTDIS	Disable Boot. When the `MSI_CMD.BOOTDIS` bit is set with the `MSI_CMD.STARTCMD` bit, the CIU terminates the boot operation. Do NOT set `MSI_CMD.BOOTDIS` and `MSI_CMD.BOOTEN` together.	
		0	No action
		1	Terminate boot
25 (R/W)	XPECTBOOTACK	Expect Boot Ack. When the `MSI_CMD.XPECTBOOTACK` bit is set with with the `MSI_CMD.BOOTEN` bit, the CIU expects a boot acknowledge start pattern of 0-1-0 from the selected card.	
		0	No ACK expected
		1	ACK expected
24 (R/W)	BOOTEN	Enable Boot. The `MSI_CMD.BOOTEN` bit should be set only for mandatory boot mode. When software sets this bit along with the `MSI_CMD.STARTCMD` bit, CIU starts the boot sequence for the corresponding card by asserting the CMD line low. Do NOT set the `MSI_CMD.BOOTEN` bit and the `MSI_CMD.BOOTDIS` bit together.	

Table 26-25: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	UCLKREGS	Update Clock Registers Only. When the `MSI_CMD.UCLKREGS` bit is set, do not send commands. Update the clock register value into card clock domain. When cleared, the normal command sequence is used. The following register values are transferred into card clock domain: CLKDIV, and CLKENA. Changes card clocks (change frequency, truncate off or on, and set low-frequency mode); provided in order to change clock frequency or stop clock without having to send command to cards. During the normal command sequence, when update_clock_registers_only = 0, the following control registers are transferred from BIU to CIU: CMD, CMDARG, TMOUT, CTYPE, BLKSIZ, BYTCNT. CIU uses new register values for new command sequence to card(s). When the `MSI_CMD.UCLKREGS` bit is set, there are no Command Done interrupts because no command is sent to SD_MMC cards.	
		0	Normal command sequence
		1	Do not send commands, just update clock register value
20:16 (R/W)	CARDNUM	Card Number. The `MSI_CMD.CARDNUM` bit represents the physical slot number of the card being accessed.	
15 (R/W)	SENDINIT	Send Initialization. When the `MSI_CMD.SENDINIT` bit is set, send an initialization sequence before sending this command. When the `MSI_CMD.SENDINIT` bit is cleared, do not send an initialization sequence (80 clocks of 1) before sending this command. After power-on, 80 clocks must be sent to the card for initialization before sending any commands to the card. The bit should be set while sending the first command to the card, so that the controller will initialize clocks before sending a command to the card. This bit should not be set for either of the boot modes (alternate or mandatory).	
		0	Do not send initialization sequence
		1	Send initialization sequence

Table 26-25: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	STPABORTCMD	Stop Abort Cmd. When the `MSI_CMD.STPABORTCMD` bit is set, the stop or abort command is intended to stop the current data transfer in progress. When an open-ended or predefined data transfer is in progress, and the host issues a stop or abort command to stop the data transfer, the `MSI_CMD.STPABORTCMD` bit should be set, so that the command/data state-machines of CIU can return correctly to an idle state. This is also applicable for boot mode transfers. To abort boot mode, this bit should be set along with CMD[26] = disable_boot. When cleared, neither stop nor abort command to stop current data transfer in progress. If abort is sent to function-number currently selected or not in data-transfer mode, then the bit should be set to 0.	
		0	Neither stop nor abort command
		1	Stop or abort command
13 (R/W)	WTPRIVDATA	Wait Prvdata Complete. When the `MSI_CMD.WTPRIVDATA` bit is set, wait for the previous data transfer to complete before sending a command. When the `MSI_CMD.WTPRIVDATA` bit is cleared, send the command at once, even if the previous data transfer has not completed. The `MSI_CMD.WTPRIVDATA` = 0 option is typically used to query the status of the card during data transfer or to stop the current data transfer; the card number should be the same as in the previous command.	
		0	Send command at once
		1	Wait for previous data transfer completion
12 (R/W)	SENDASTOP	Send Auto Stop. When the `MSI_CMD.SENDASTOP` bit is set, MSI sends the stop command to SD_MMC cards at the end of the data transfer. Refer to the Auto Stop section in the chapter to determine: • When the `MSI_CMD.SENDASTOP` bit should be set, since some data transfers do not need explicit stop commands • When software should explicitly send a stop command during open-ended transfers Additionally, when resuming suspended memory access of SD-Combo card, the `MSI_CMD.SENDASTOP` bit should be set correctly to send a stop command. The bit does not need to be set if no data is expected from the card.	
		0	No stop command sent at end of data transfer
		1	Send stop command at end of data transfer

Table 26-25: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	XFRMODE	Transfer Mode. When the `MSI_CMD.XFRMODE` bit is set, stream data transfer command. If no data expected, do-not-care.	
		0	Block data transfer command
		1	Stream data transfer command
10 (R/W)	RDWR	Read from or Write to Card. When the `MSI_CMD.RDWR` bit is set, write to the card. When the `MSI_CMD.RDWR` bit is cleared, read from the card.	
		0	Read from card
		1	Write to card
9 (R/W)	DXPECT	Data Expected. When the `MSI_CMD.DXPECT` bit is set, data transfer is expected. When the `MSI_CMD.DXPECT` bit is cleared, no data transfer is expected.	
		0	No data transfer expected
		1	Data transfer expected
8 (R/W)	CHKRESPCRC	Check Response CRC. When the `MSI_CMD.CHKRESPCRC` bit is set, check the response CRC. When cleared, do not check the response. Some of the command responses do not return valid CRC bits. Software should disable CRC checks for those commands in order to disable CRC checking by controller.	
		0	Do not check response
		1	Check response
7 (R/W)	RLEN	Response Length. When the `MSI_CMD.RLEN` bit is set, a long response is expected from the card. When cleared, a short response is expected.	
		0	Short response expected
		1	Long response expected
6 (R/W)	RXPECT	Response Expect. When the `MSI_CMD.RXPECT` bit is set, a response is expected from the card. When cleared, no response is expected.	
		0	No response expected
		1	Response expected

Table 26-25: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	INDX	Command Index.

Command Argument Register

The MSI_CMDARG register provides bits that specify the command argument to be passed to the card.

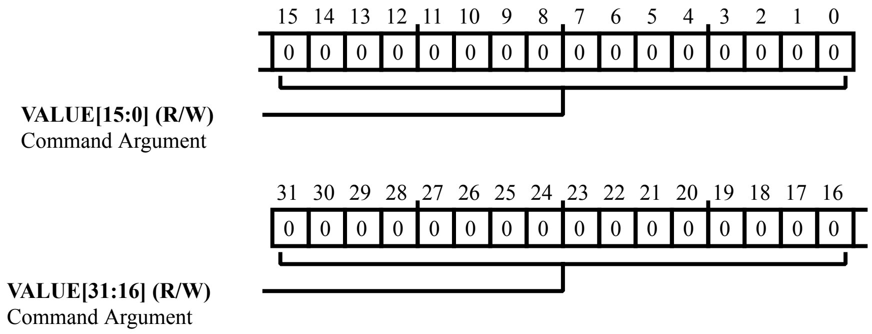

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Command Argument
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Command Argument

Figure 26-23: MSI_CMDARG Register Diagram

Table 26-26: MSI_CMDARG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Argument. The MSI_CMDARG.VALUE bit field specifies the command argument to be passed to the card.

Control Register

The MSI_CTL register controls the various settings used by the MSI module.

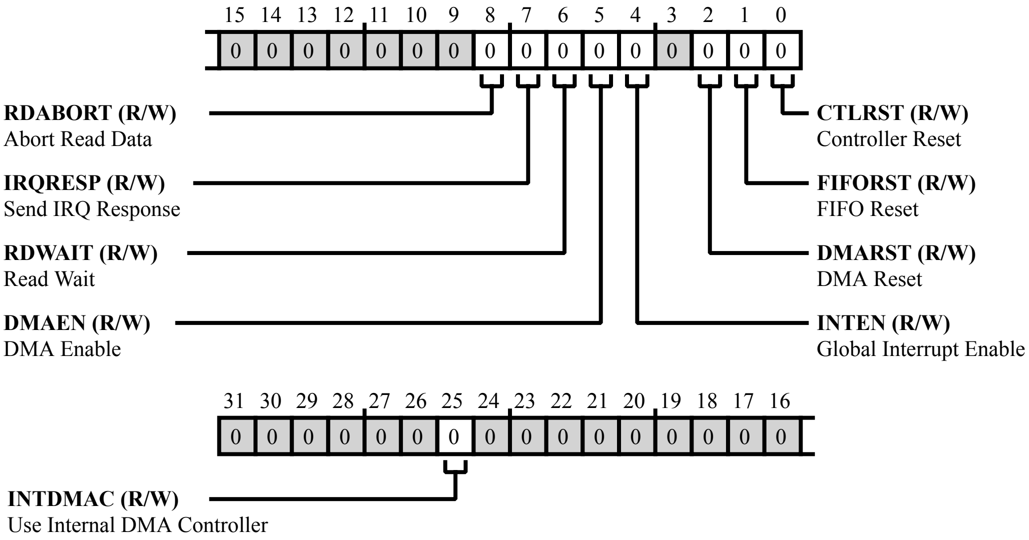

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDABORT (R/W)
Abort Read Data
IRQRESP (R/W)
Send IRQ Response
RDWAIT (R/W)
Read Wait
DMAEN (R/W)
DMA Enable
CTLRST (R/W)
Controller Reset
FIFORST (R/W)
FIFO Reset
DMARST (R/W)
DMA Reset
INTEN (R/W)
Global Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INTDMAC (R/W)
Use Internal DMA Controller

Figure 26-24: MSI_CTL Register Diagram

Table 26-27: MSI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	INTDMAC	Use Internal DMA Controller. The MSI_CTL.INTDMAC bit is present only for the internal DMAC configuration and determines whether the host or DMA is used for data transfers.	
		0	Host performs transfers
		1	DMA performs transfers
8 (R/W)	RDABORT	Abort Read Data. Setting the MSI_CTL.RDABORT bit after the suspend command is issued during a read-transfer operation, software polls the card to find out when the suspend happened. Once the suspend occurs, software sets the bit to reset data state-machine, which is waiting for next block of data. The bit automatically clears once the data state machine resets to idle. It is used in the SDIO card suspend sequence.	
		0	No change
		1	Reset data state machine

Table 26-27: MSI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	IRQRESP	Send IRQ Response. Setting the `MSI_CTL.IRQRESP` bit sends an auto IRQ response. It is cleared once the response is sent. To wait for MMC card interrupts, the host issues CMD40, and the MSI waits for an interrupt response from the MMC card(s). In the meantime, if the host wants the MSI to exit waiting for interrupt state, it can set the `MSI_CTL.IRQRESP` bit, at which time the MSI command state-machine sends the CMD40 response on the bus and returns to the idle state.	
		0	No change
		1	Send auto IRQ response
6 (R/W)	RDWAIT	Read Wait. Setting the `MSI_CTL.RDWAIT` bit allows sending read-wait to SDIO cards.	
		0	Clear read wait
		1	Assert read wait
5 (R/W)	DMAEN	DMA Enable. Setting the `MSI_CTL.DMAEN` bit enables DMA transfer mode.	
		0	Disable
		1	Enable
4 (R/W)	INTEN	Global Interrupt Enable. Setting the `MSI_CTL.INTEN` bit enables global interrupts. The int port is 1 only when this bit is 1 and one or more unmasked interrupts are set.	
		0	Disable
		1	Enable
2 (R/W)	DMARST	DMA Reset. Setting the `MSI_CTL.DMARST` bit resets internal DMA interface control logic. To reset the DMA interface, firmware should set this bit to 1. This bit is auto-cleared after two peripheral clocks.	
		0	No change
		1	Reset

Table 26-27: MSI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FIFORST	FIFO Reset. Setting the `MSI_CTL.FIFORST` bit provides a reset to data FIFO to reset FIFO pointers. To reset FIFO, firmware should set bit to 1. This bit is auto-cleared after completion of reset operation.	
		0	No change
		1	Reset
0 (R/W)	CTLRST	Controller Reset. Setting the `MSI_CTL.CTLRST` bit resets the MSI. To reset the controller, firmware should set the bit to 1. This bit is auto-cleared after two peripheral bus and two cclk_in clock cycles. This resets the: • BIU/CIU interface • CIU and state machines • `MSI_CTL.RDABORT`, `MSI_CTL.IRQRESP`, and `MSI_CTL.RDWAIT` bits • `MSI_CMD.STARTCMD` bit It does not affect any registers or DMA interface, or FIFO or host interrupts.	
		0	No change
		1	Reset

Card Type Register

The `MSI_CTYPE` register provides bits that configure card widths.

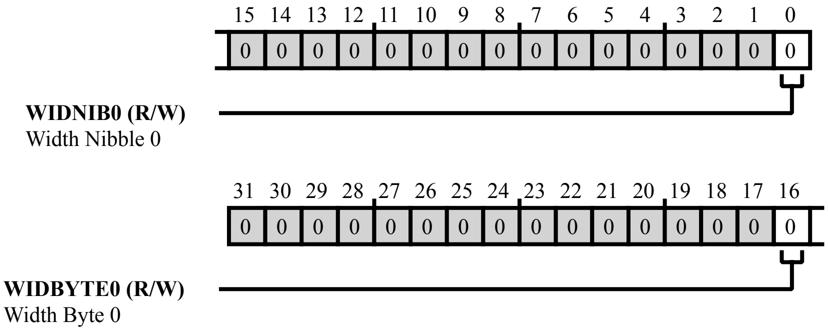

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WIDNIB0 (R/W)
Width Nibble 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WIDBYTE0 (R/W)
Width Byte 0

Figure 26-25: MSI_CTYPE Register Diagram

Table 26-28: MSI_CTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	WIDBYTE0	Width Byte 0. The `MSI_CTYPE.WIDBYTE0` bit enables 8-bit mode for card 0.
0 (R/W)	WIDNIB0	Width Nibble 0. The `MSI_CTYPE.WIDNIB0` bit enables 4-bit mode for card 0.

Descriptor List Base Address Register

The `MSI_DBADDR` register contains the base address of the first descriptor. The LSB bits [1:0] bits are ignored and taken as all-zero by the IDMAC internally and these LSB bits are read-only.

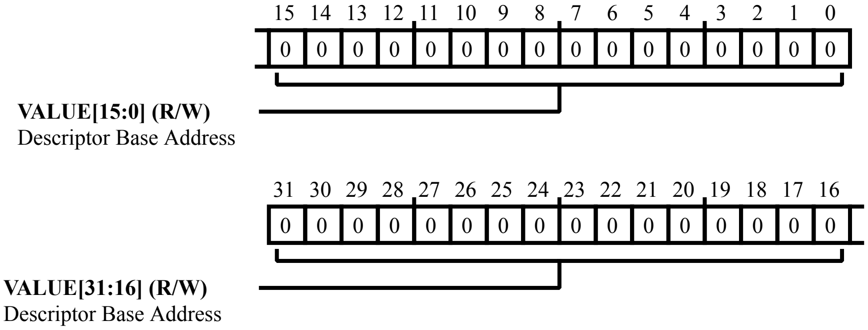

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Descriptor Base Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Descriptor Base Address

Figure 26-26: MSI_DBADDR Register Diagram

Table 26-29: MSI_DBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Descriptor Base Address. The `MSI_DBADDR.VALUE` bit field contains the base address of the first descriptor.

Debounce Count Register

The `MSI_DEBNCE` register provides the number of host clocks (clk) used by debounce filter logic; typical debounce time is 5-25 ms.

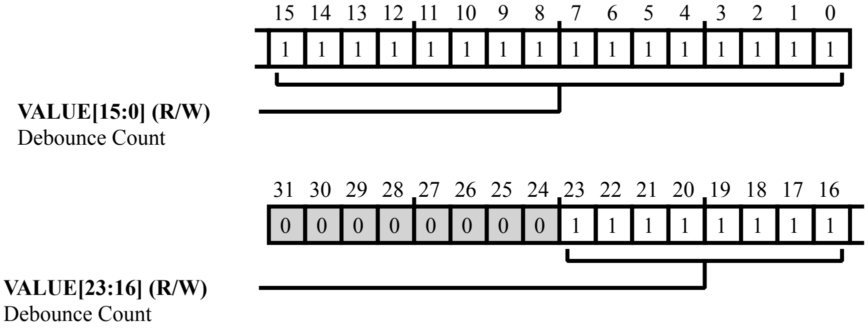

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
VALUE[15:0] (R/W)
Debounce Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
VALUE[23:16] (R/W)
Debounce Count

Figure 26-27: MSI_DEBNCE Register Diagram

Table 26-30: MSI_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Debounce Count. The `MSI_DEBNCE.VALUE` bit field provides the number of host clocks (clk) used by debounce filter logic.

Current Host Descriptor Address Register

The MSI_DSCADDR register points to the start address of the current descriptor read by the IDMAC.

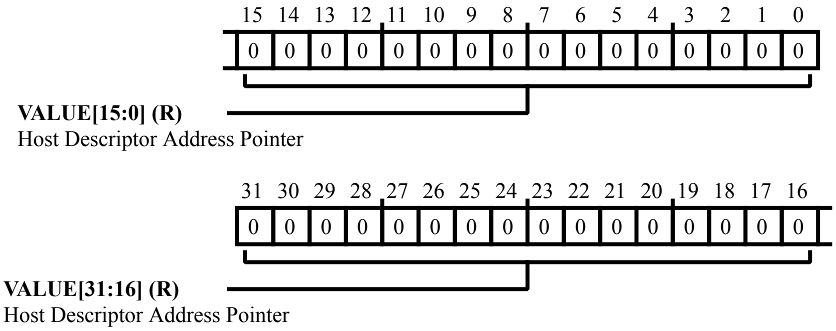

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Host Descriptor Address Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Host Descriptor Address Pointer

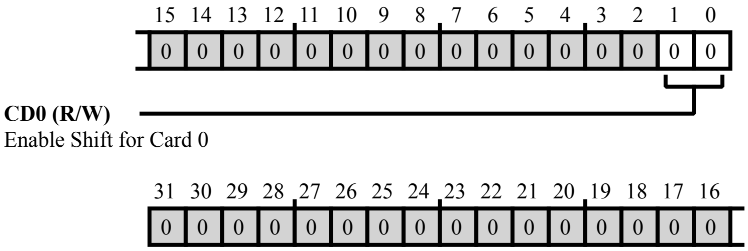
Figure 26-28: MSI_DSCADDR Register Diagram

Table 26-31: MSI_DSCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Host Descriptor Address Pointer. The `MSI_DSCADDR.VALUE` bit field points to the start address of the current descriptor read by the IDMAC.

Enable Phase Shift Register

The `MSI_ENSHIFT` register provides control for the amount of phase shift provided on the default enables in the design. Two bits are assigned for each card/slot.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CD0 (R/W)
Enable Shift for Card 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-29: MSI_ENSHIFT Register Diagram

Table 26-32: MSI_ENSHIFT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	CD0	Enable Shift for Card 0. The `MSI_ENSHIFT.CD0` bit field provides control for the amount of phase shift provided on the default enables in the design.

FIFO Threshold Watermark Register

The MSI_FIFOTH register provides bits that manage the FIFO transactions.

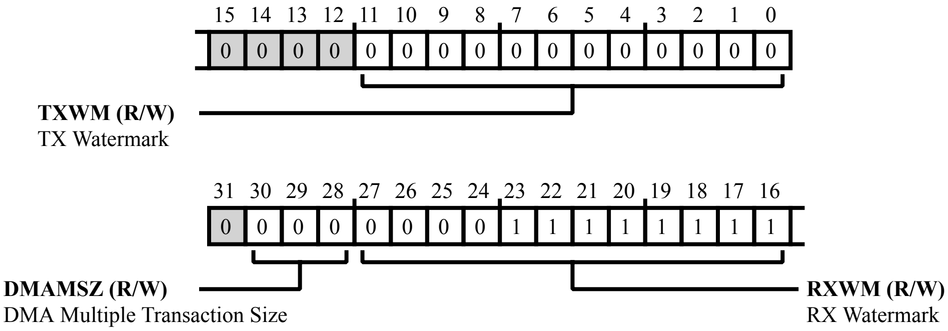

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TXWM (R/W)
TX Watermark
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
DMAMSZ (R/W)
DMA Multiple Transaction Size
RXWM (R/W)
RX Watermark

Figure 26-30: MSI_FIFOTH Register Diagram

Table 26-33: MSI_FIFOTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30:28 (R/W)	DMAMSZ	DMA Multiple Transaction Size. The MSI_FIFOTH.DMAMSZ bit field configures the burst size of a multiple transaction. The units for transfer are 32-bit.	
		0	1 transfer
		1	4 transfers
		2	8 transfers
		3	16 transfers
		4	32 transfers
		5	64 transfers
		6	128 transfers
		7	256 transfers
27:16 (R/W)	RXWM	RX Watermark. The MSI_FIFOTH.RXWM bit field sets the FIFO threshold watermark level when receiving data to card. When the FIFO data count is greater than this number, a DMA/FIFO request is raised.	
11:0 (R/W)	TXWM	TX Watermark. The MSI_FIFOTH.TXWM bit field sets the FIFO threshold watermark level when transmitting data to card. When the FIFO data count is less than or equal to this number, a DMA/FIFO request is raised.	

Internal DMA Interrupt Enable Register

The `MSI_IDINTEN` register provides bits for setting various interrupts in DMA mode.

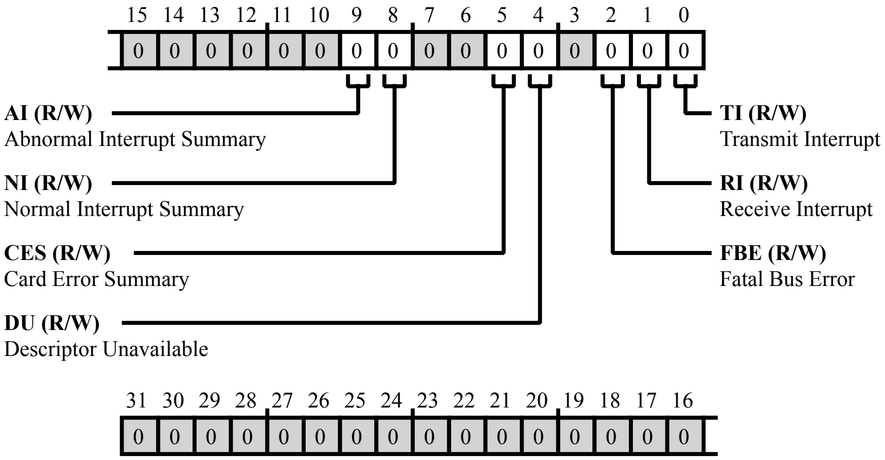

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AI (R/W)
Abnormal Interrupt Summary
NI (R/W)
Normal Interrupt Summary
CES (R/W)
Card Error Summary
DU (R/W)
Descriptor Unavailable
TI (R/W)
Transmit Interrupt
RI (R/W)
Receive Interrupt
FBE (R/W)
Fatal Bus Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 26-31: MSI_IDINTEN Register Diagram

Table 26-34: MSI_IDINTEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	AI	Abnormal Interrupt Summary. Setting the `MSI_IDINTEN.AI` bit enables the abnormal interrupt summary. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
8 (R/W)	NI	Normal Interrupt Summary. Setting the `MSI_IDINTEN.NI` bit enables the normal interrupt summary. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
5 (R/W)	CES	Card Error Summary. Setting the `MSI_IDINTEN.CES` bit enables the card error summary interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled

Table 26-34: MSI_IDINTEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DU	Descriptor Unavailable. Setting the `MSI_IDINTEN.DU` bit enables the descriptor unavailable interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
2 (R/W)	FBE	Fatal Bus Error. Setting the `MSI_IDINTEN.FBE` bit enables the FBE interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
1 (R/W)	RI	Receive Interrupt. Setting the `MSI_IDINTEN.RI` bit enables the receive interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
0 (R/W)	TI	Transmit Interrupt. Setting the `MSI_IDINTEN.TI` bit enables the transmit interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled

Internal DMA Status Register

The `MSI_IDSTS` register provides DMA status information. This register is updated only when the DMA is active.

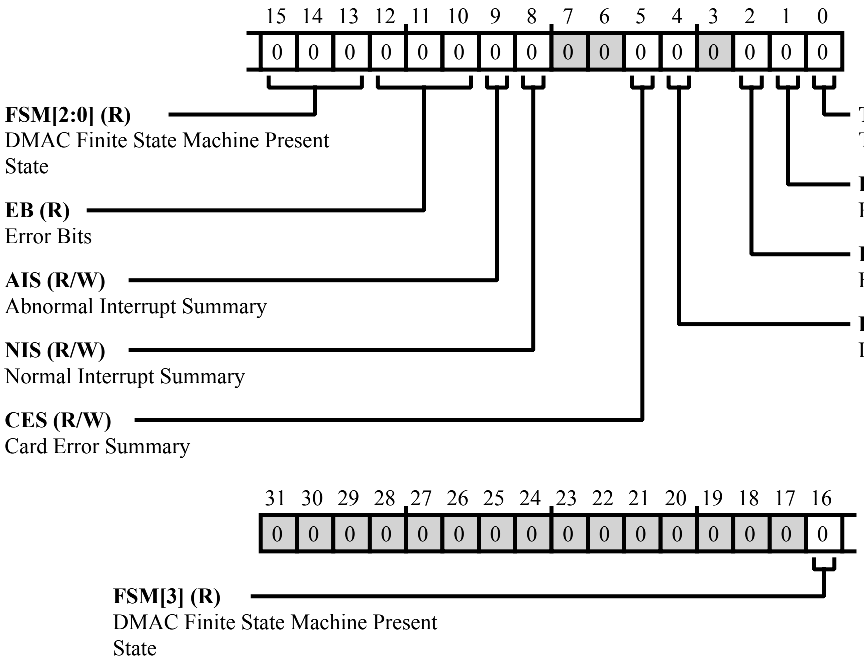

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSM[2:0] (R)
DMAC Finite State Machine Present State
EB (R)
Error Bits
AIS (R/W)
Abnormal Interrupt Summary
NIS (R/W)
Normal Interrupt Summary
CES (R/W)
Card Error Summary
TI (R/W)
Transmit Interrupt
RI (R/W)
Receive Interrupt
FBE (R/W)
Fatal Bus Error Interrupt
DU (R/W)
Descriptor Unavailable Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSM[3] (R)
DMAC Finite State Machine Present State

Figure 26-32: MSI_IDSTS Register Diagram

Table 26-35: MSI_IDSTS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:13 (R/NW)	FSM	DMAC Finite State Machine Present State. The `MSI_IDSTS.FSM` bit field indicates the present state of the finite state machine IDMAC (DMA controller).	
		0	DMA Idle
		1	DMA suspend
		2	DESC_RD
		3	DESC_CHK
		4	DMA_RD_REQ_WAIT
		5	DMA_WR_REQ_WAIT
		6	DMA_RD
		7	DMA_WR
		8	DESC_CLOSE

Table 26-35: MSI_IDSTS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12:10 (R/NW)	EB	Error Bits. The `MSI_IDSTS.EB` bit field indicates the type of error that caused a bus error. It is valid only with `MSI_IDSTS.FBE` set. This field does not generate an interrupt. The `MSI_IDSTS.EB` bit field is read-only.	
		0	3'b001 - Host abort received during transmission
		1	3'b010 - Host abort received during reception
9 (R/W)	AIS	Abnormal Interrupt Summary. The `MSI_IDSTS.AIS` bit field is a logical OR of the following: IDSTS[2] - Fatal Bus Interrupt IDSTS[4] - DU bit Interrupt Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes `MSI_IDSTS.AIS` to be set is cleared. Writing a 1 clears this bit.	
8 (R/W)	NIS	Normal Interrupt Summary. The `MSI_IDSTS.NIS` bit field is a logical OR of the following: IDSTS[0] - Transmit Interrupt IDSTS[1] - Receive Interrupt Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes `MSI_IDSTS.NIS` to be set is cleared. Writing a 1 clears this bit.	
5 (R/W)	CES	Card Error Summary. The `MSI_IDSTS.CES` bit indicates the logical OR of the following bits: • EBE - End Bit Error • RTO - Response Timeout/Boot Ack Timeout • RCRC - Response CRC • SBE - Start Bit Error • DRTO - Data Read Timeout/BDS timeout • DCRC - Data CRC for Receive • RE - Response Error Writing a 1 clears this bit.	
		0	No error occurred
		1	Error occurred

Table 26-35: MSI_IDSTS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DU	Descriptor Unavailable Interrupt. The `MSI_IDSTS.DU` bit indicates a descriptor unavailable error. If software creates less descriptors than the boot data received, the MSI generates this interrupt and does not transfer any further data to system memory.	
		0	No error occurred
		1	Error occurred
2 (R/W)	FBE	Fatal Bus Error Interrupt. The `MSI_IDSTS.FBE` bit indicates a fatal bus error error. An FBE occurs due to an error response from the SCB. Because this is a system error, the software driver should not perform any further programming of the MSI. The only recovery mechanism from such an error is to issue a hard reset or to perform a controller reset by writing to the `MSI_CTL.CTLRST` bit.	
		0	No error occurred
		1	Error occurred
1 (R/W)	RI	Receive Interrupt. The `MSI_IDSTS.RI` bit indicates that data reception is finished for a descriptor.	
		0	No event occurred
		1	Event occurred
0 (R/W)	TI	Transmit Interrupt. The `MSI_IDSTS.TI` bit indicates that data transmission is finished for a descriptor.	
		0	No event occurred
		1	Event occurred

Interrupt Mask Register

The MSI_IMSK register provides bits that allow the masking of unwanted interrupts.

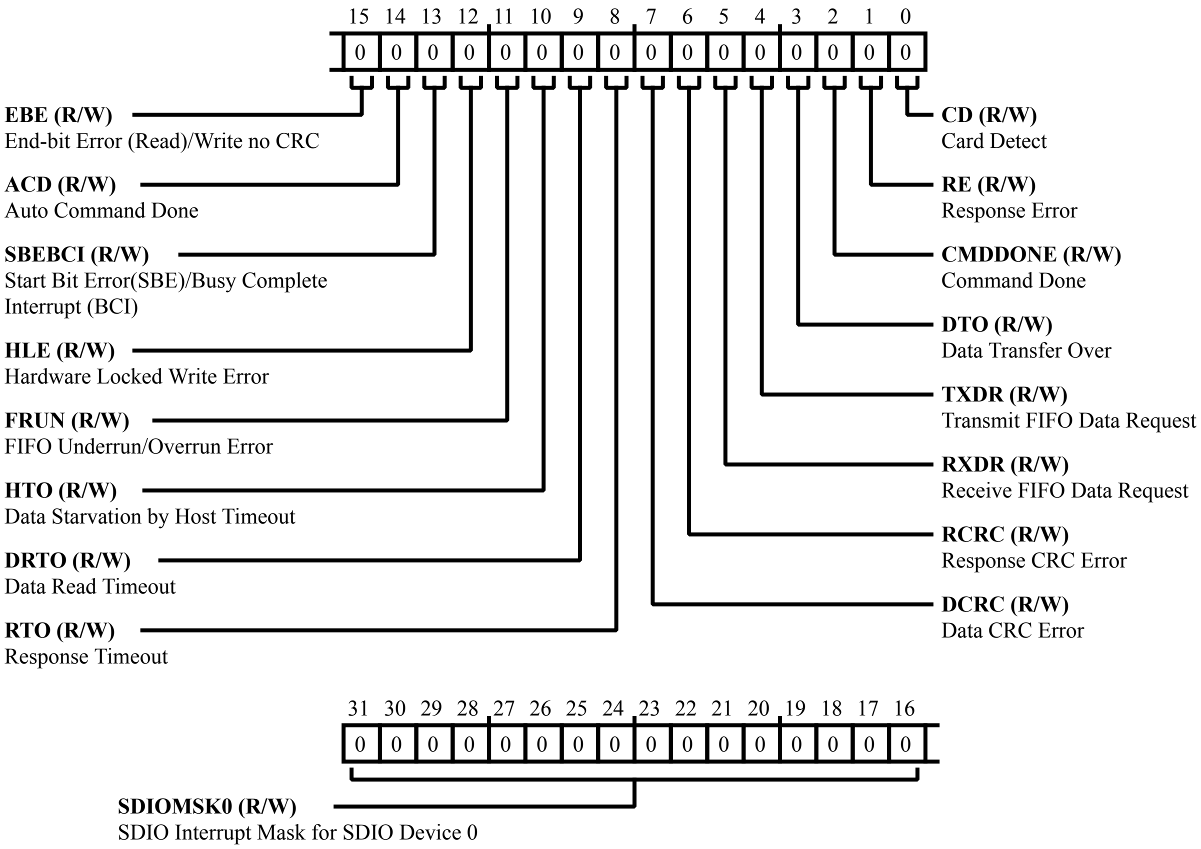

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EBE (R/W)
End-bit Error (Read)/Write no CRC
ACD (R/W)
Auto Command Done
SBEBCI (R/W)
Start Bit Error(SBE)/Busy Complete Interrupt (BCI)
HLE (R/W)
Hardware Locked Write Error
FRUN (R/W)
FIFO Underrun/Overrun Error
HTO (R/W)
Data Starvation by Host Timeout
DRTO (R/W)
Data Read Timeout
RTO (R/W)
Response Timeout
CD (R/W)
Card Detect
RE (R/W)
Response Error
CMDDONE (R/W)
Command Done
DTO (R/W)
Data Transfer Over
TXDR (R/W)
Transmit FIFO Data Request
RXDR (R/W)
Receive FIFO Data Request
RCRC (R/W)
Response CRC Error
DCRC (R/W)
Data CRC Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SDIOMSK0 (R/W)
SDIO Interrupt Mask for SDIO Device 0

Figure 26-33: MSI_IMSK Register Diagram

Table 26-36: MSI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16 (R/W)	SDIOMSK0	SDIO Interrupt Mask for SDIO Device 0. The MSI_IMSK.SDIOMSK0 bits mask SDIO interrupts, one bit for one card. When masked, SDIO interrupt detection for that card is disabled. A 0 masks an interrupt, and 1 enables an interrupt.	
15 (R/W)	EBE	End-bit Error (Read)/Write no CRC. The MSI_IMSK.EBE bit masks the end bit read/write no CRC error.	
		0	Masked
		1	Enabled

Table 26-36: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ACD	Auto Command Done. The `MSI_IMSK.ACD` bit masks the auto command done error.	
		0	Masked
		1	Enabled
13 (R/W)	SBEBCI	Start Bit Error(SBE)/Busy Complete Interrupt (BCI). The `MSI_IMSK.SBEBCI` bit masks the start bit/busy complete error.	
		0	Masked
		1	Enabled
12 (R/W)	HLE	Hardware Locked Write Error. The `MSI_IMSK.HLE` bit masks the hardware locked write error.	
		0	Masked
		1	Enabled
11 (R/W)	FRUN	FIFO Underrun/Overrun Error. The `MSI_IMSK.FRUN` bit masks the FIFO overrun or underrun error.	
		0	Masked
		1	Enabled
10 (R/W)	HTO	Data Starvation by Host Timeout. The `MSI_IMSK.HTO` bit masks the host timeout error.	
		0	Masked
		1	Enabled
9 (R/W)	DRTO	Data Read Timeout. The `MSI_IMSK.DRTO` bit masks the data read timeout error.	
		0	Masked
		1	Enabled
8 (R/W)	RTO	Response Timeout. The `MSI_IMSK.RTO` bit masks the response timeout error.	
		0	Masked
		1	Enabled

Table 26-36: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	DCRC	Data CRC Error. The `MSI_IMSK.DCRC` bit masks the data CRC error where the received data CRC does not match with locally-generated CRC in CIU. The error can also occur if the write CRC status is incorrectly sampled by the host.	
		0	Masked
		1	Enabled
6 (R/W)	RCRC	Response CRC Error. The `MSI_IMSK.RCRC` bit masks the response CRC error which is set when the response CRC does not match with the locally-generated CRC in the CIU.	
		0	Masked
		1	Enabled
5 (R/W)	RXDR	Receive FIFO Data Request. The `MSI_IMSK.RXDR` bit masks the receive FIFO data request interrupt which is set during write operation to card when FIFO level reaches less than or equal to transmit-threshold level.	
		0	Masked
		1	Enabled
4 (R/W)	TXDR	Transmit FIFO Data Request. The `MSI_IMSK.TXDR` bit masks the transmit FIFO data request interrupt which is set during write operation to card when FIFO level reaches less than or equal to transmit-threshold level.	
		0	Masked
		1	Enabled
3 (R/W)	DTO	Data Transfer Over. The `MSI_IMSK.DTO` bit masks the data transfer over interrupt which indicates the data transfer completed. Though on detection of errors such as the start bit error, the data CRC error, and so on, DTO may or may not be set; the application must issue CMD12, which ensures that DTO is set.	
		0	Masked
		1	Enabled

Table 26-36: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CMDDONE	Command Done. The `MSI_IMSK.CMDDONE` bit masks the command done interrupt where a command is sent to a card and received a response from the card, even if a response error or a CRC error occurs.	
		0	Masked
		1	Enabled
1 (R/W)	RE	Response Error. The `MSI_IMSK.RE` masks a response error. This is an error in received response set if one of following occurs: transmission bit != 0, command index mismatch, End-bit != 1.	
		0	Masked
		1	Enabled
0 (R/W)	CD	Card Detect. The `MSI_IMSK.CD` bit masks the card detect interrupt. On power-on, the controller should read in the card_detect port and store the value in the memory. Upon receiving a card-detect interrupt, it should again read the card_detect port and XOR with the previous card-detect status to find out which card has interrupted.	
		0	Masked
		1	Enabled

Raw Interrupt Status Register

The `MSI_ISTAT` register provides bits that clear interrupts. Conditions 6 through 9 indicate that the received data may have errors. If there was a response timeout, then no data transfer occurred.

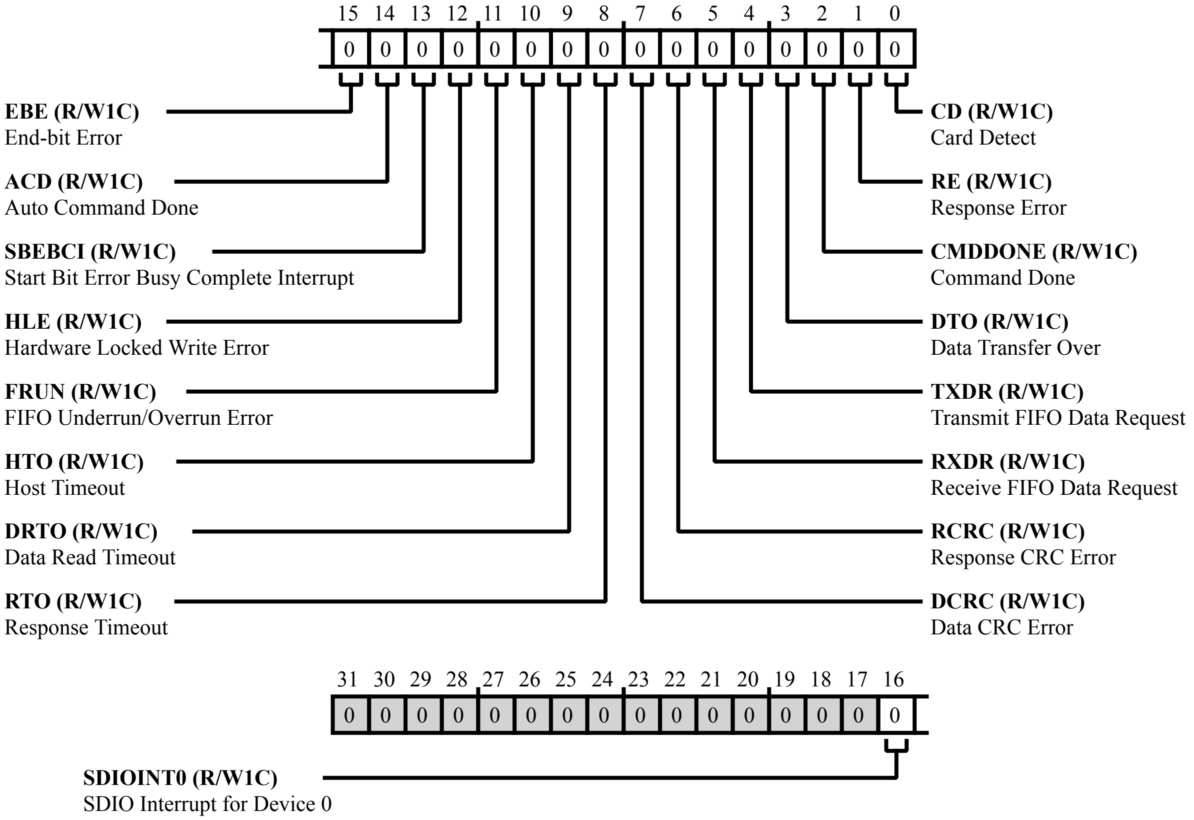

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EBE (R/W1C)
End-bit Error
ACD (R/W1C)
Auto Command Done
SBEBCI (R/W1C)
Start Bit Error Busy Complete Interrupt
HLE (R/W1C)
Hardware Locked Write Error
FRUN (R/W1C)
FIFO Underrun/Overrun Error
HTO (R/W1C)
Host Timeout
DRTO (R/W1C)
Data Read Timeout
RTO (R/W1C)
Response Timeout
CD (R/W1C)
Card Detect
RE (R/W1C)
Response Error
CMDDONE (R/W1C)
Command Done
DTO (R/W1C)
Data Transfer Over
TXDR (R/W1C)
Transmit FIFO Data Request
RXDR (R/W1C)
Receive FIFO Data Request
RCRC (R/W1C)
Response CRC Error
DCRC (R/W1C)
Data CRC Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SDIOINT0 (R/W1C)
SDIO Interrupt for Device 0

Figure 26-34: MSI_ISTAT Register Diagram

Table 26-37: MSI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	SDIOINT0	SDIO Interrupt for Device 0. The `MSI_ISTAT.SDIOINT0` bit indicates that an interrupt occurred on an SDIO card.	
15 (R/W1C)	EBE	End-bit Error. The `MSI_ISTAT.EBE` bit indicates that the start bit of the CRC status was not received by two clocks after the end of the data block. A CRC error is indicated by the card.	
		0	No error
		1	Error occurred

Table 26-37: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	ACD	Auto Command Done.	
		0	No error
		1	Error occurred
13 (R/W1C)	SBEBCI	Start Bit Error Busy Complete Interrupt. The `MSI_ISTAT.SBEBCI` bit indicates the completion of a busy signal driven by the card after a write data transfer.	
		0	No error
		1	Error occurred
12 (R/W1C)	HLE	Hardware Locked Write Error.	
		0	No error
		1	Error occurred
11 (R/W1C)	FRUN	FIFO Underrun/Overrun Error.	
		0	No error
		1	Error occurred
10 (R/W1C)	HTO	Host Timeout. The `MSI_ISTAT.HTO` bit indicates that the FIFO is empty during transmission or is full during reception. Unless software/DMA writes data for an empty condition or reads data for a full condition, the MSI cannot continue with data transfer. The clock to the card is stopped.	
		0	No error
		1	Error occurred
9 (R/W1C)	DRTO	Data Read Timeout. The `MSI_ISTAT.DRTO` bit indicates that the card has not sent data within the timeout period.	
		0	No error
		1	Error occurred
8 (R/W1C)	RTO	Response Timeout.	
		0	No error
		1	Error occurred

Table 26-37: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	DCRC	Data CRC Error. The `MSI_ISTAT.DCRC` bit indicates that a data CRC error where the received data CRC does not match with the locally-generated CRC in the CIU. This error can also occur if the write CRC status is incorrectly sampled by the host.	
		0	No error
		1	Error occurred
6 (R/W1C)	RCRC	Response CRC Error. The `MSI_ISTAT.RCRC` bit indicates that the response CRC error which is caused when the response CRC does not match with the locally-generated CRC in the CIU.	
		0	No error
		1	Error occurred
5 (R/W1C)	RXDR	Receive FIFO Data Request. The `MSI_ISTAT.RXDR` bit indicates that the FIFO threshold for receiving data was reached and software/DMA is expected to read data from the FIFO.	
		0	No error
		1	Error occurred
4 (R/W1C)	TXDR	Transmit FIFO Data Request. The `MSI_ISTAT.TXDR` bit indicates that the FIFO threshold for transmitting data was reached and is less than or equal to transmit-threshold level. Software/DMA is expected to write data, if available, in the FIFO. This interrupt is set during a write operation to the card.	
		0	No error
		1	Error occurred
3 (R/W1C)	DTO	Data Transfer Over. The `MSI_ISTAT.DTO` bit indicates the data transfer completed. If there is a response timeout error, then the MSI does not attempt any data transfer and this bit is never set.	
		0	No error
		1	Error occurred
2 (R/W1C)	CMDDONE	Command Done. The `MSI_ISTAT.CMDDONE` bit indicates that a command that was sent to a card received a response from the card, even a response error or CRC error occurred.	
		0	No error
		1	Error occurred

Table 26-37: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RE	Response Error. The `MSI_ISTAT.RE` bit indicates that an error was received during response reception. In this case, the response that copied in the response registers is invalid. Software can retry the command.	
		0	No error
		1	Error occurred
0 (R/W1C)	CD	Card Detect. The `MSI_ISTAT.CD` bit indicates a card detect interrupt.	
		0	No error
		1	Error occurred

Masked Interrupt Status Register

The `MSI_MSKISTAT` register indicates the status for the bits which are unmasked in the `MSI_IMSK` register. In other words, `MSI_MSKISTAT` = `MSI_ISTAT` and `MSI_IMSK`.

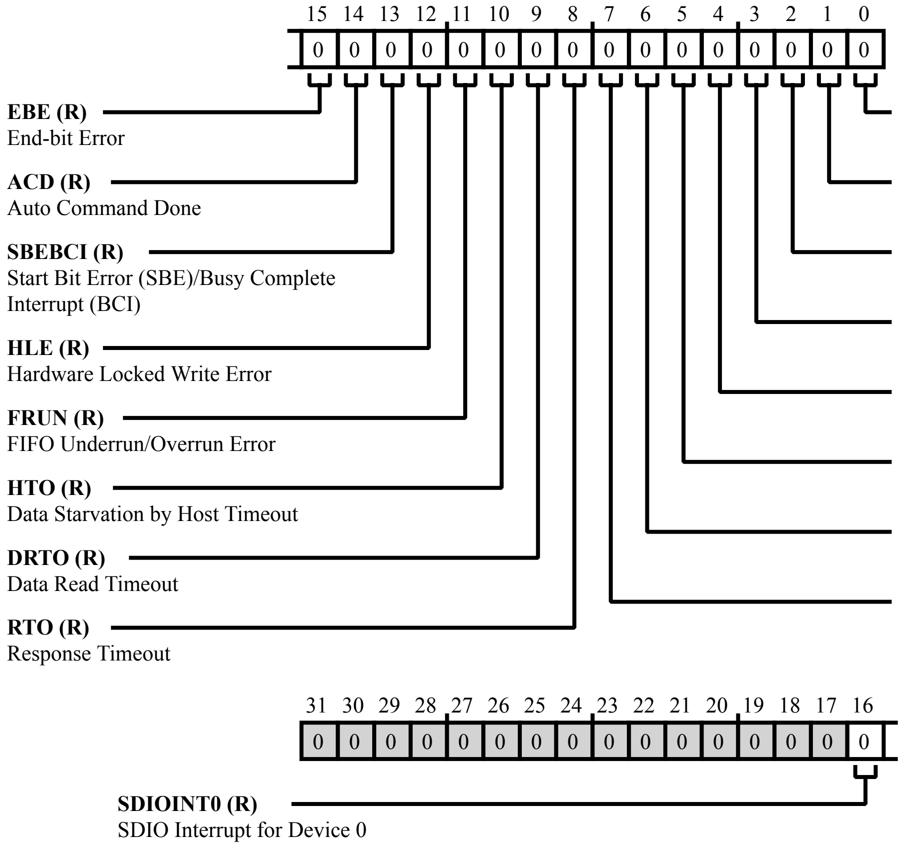

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EBE (R)
End-bit Error
ACD (R)
Auto Command Done
SBEBCI (R)
Start Bit Error (SBE)/Busy Complete Interrupt (BCI)
HLE (R)
Hardware Locked Write Error
FRUN (R)
FIFO Underrun/Overrun Error
HTO (R)
Data Starvation by Host Timeout
DRTO (R)
Data Read Timeout
RTO (R)
Response Timeout
CD (R)
Card Detect
RE (R)
Response Error
CMDDONE (R)
Command Done
DTO (R)
Data Transfer Over
TXDR (R)
Transmit FIFO Data Request
RXDR (R)
Receive FIFO Data Request
RCRC (R)
Response CRC Error
DCRC (R)
Data CRC Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SDIOINT0 (R)
SDIO Interrupt for Device 0

Figure 26-35: MSI_MSKISTAT Register Diagram

Table 26-38: MSI_MSKISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	SDIOINT0	SDIO Interrupt for Device 0. The `MSI_MSKISTAT.SDIOINT0` bit indicates an interrupt occurred on the SDIO interrupt for device 0 masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 26-38: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	EBE	End-bit Error. The `MSI_MSKISTAT.EBE` bit indicates an interrupt occurred on the end-bit error (read)/write no CRC masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
14 (R/NW)	ACD	Auto Command Done. The `MSI_MSKISTAT.ACD` bit indicates an interrupt occurred on the auto command done masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
13 (R/NW)	SBEBCI	Start Bit Error (SBE)/Busy Complete Interrupt (BCI). The `MSI_MSKISTAT.SBEBCI` bit indicates an interrupt occurred on the start bit error/busy complete masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
12 (R/NW)	HLE	Hardware Locked Write Error. The `MSI_MSKISTAT.HLE` bit indicates an interrupt occurred on the hardware locked write error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
11 (R/NW)	FRUN	FIFO Underrun/Overrun Error. The `MSI_MSKISTAT.FRUN` bit indicates an interrupt occurred on the FIFO underrun/overrun error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
10 (R/NW)	HTO	Data Starvation by Host Timeout. The `MSI_MSKISTAT.HTO` bit indicates an interrupt occurred on the data starvation by host timeout masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 26-38: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	DRTO	Data Read Timeout. The `MSI_MSKISTAT.DRTO` bit indicates an interrupt occurred on the data read timeout masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
8 (R/NW)	RTO	Response Timeout. The `MSI_MSKISTAT.RTO` bit indicates an interrupt occurred on the response timeout masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
7 (R/NW)	DCRC	Data CRC Error. The `MSI_MSKISTAT.DCRC` bit indicates an interrupt occurred on the data CRC error request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
6 (R/NW)	RCRC	Response CRC Error. The `MSI_MSKISTAT.RCRC` bit indicates an interrupt occurred on the response CRC error data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
5 (R/NW)	RXDR	Receive FIFO Data Request. The `MSI_MSKISTAT.RXDR` bit indicates an interrupt occurred on the receive FIFO data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
4 (R/NW)	TXDR	Transmit FIFO Data Request. The `MSI_MSKISTAT.TXDR` bit indicates an interrupt occurred on the transmit FIFO data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 26-38: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	DTO	Data Transfer Over. The `MSI_MSKISTAT.DTO` bit indicates an interrupt occurred on the data transfer over masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
2 (R/NW)	CMDDONE	Command Done. The `MSI_MSKISTAT.CMDDONE` bit indicates an interrupt occurred on the command done masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
1 (R/NW)	RE	Response Error. The `MSI_MSKISTAT.RE` bit indicates an interrupt occurred on the response error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
0 (R/NW)	CD	Card Detect. The `MSI_MSKISTAT.CD` bit indicates an interrupt occurred on the card detect masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Poll Demand Register

If the OWN bit of a descriptor is not set, the FSM goes into the suspend state. The host needs to write any value into the `MSI_PLDMND` register for the IDMAC FSM to resume normal descriptor fetch operation. This is a write-only register.

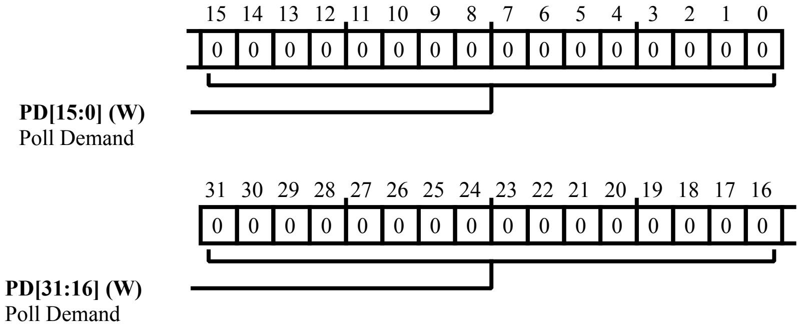

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PD[15:0] (W)
Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PD[31:16] (W)
Poll Demand

Figure 26-36: MSI_PLDMND Register Diagram

Table 26-39: MSI_PLDMND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	PD	Poll Demand. The `MSI_PLDMND.PD` bit field needs to be written so that the IDMAC FSM can resume normal descriptor fetch operation.

Response Register 0

The `MSI_RESP0` register represents bits[31:0] of a long response.

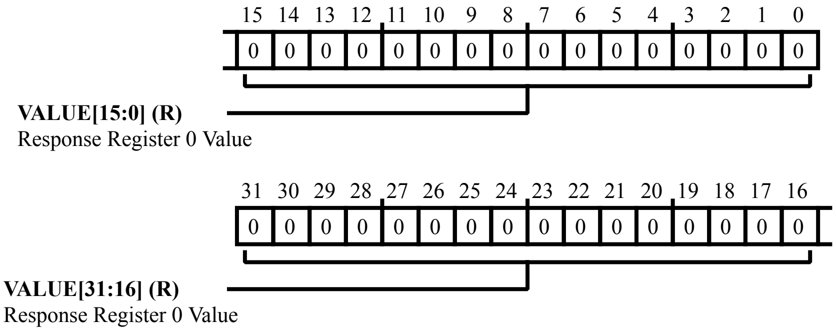

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Response Register 0 Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Response Register 0 Value

Figure 26-37: MSI_RESP0 Register Diagram

Table 26-40: MSI_RESP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 0 Value. The `MSI_RESP0.VALUE` bit field represents bits[31:0] of a long response.

Response Register 1

The `MSI_RESP1` register represents bits[63:32] of a long response.

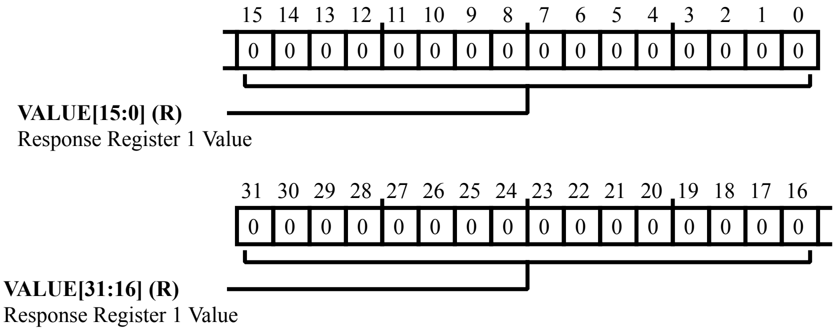

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Response Register 1 Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Response Register 1 Value

Figure 26-38: MSI_RESP1 Register Diagram

Table 26-41: MSI_RESP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 1 Value. The `MSI_RESP1.VALUE` bit field represents bits[63:32] of a long response.

Response Register 2

The `MSI_RESP2` register represents bits[95:64] of a long response.

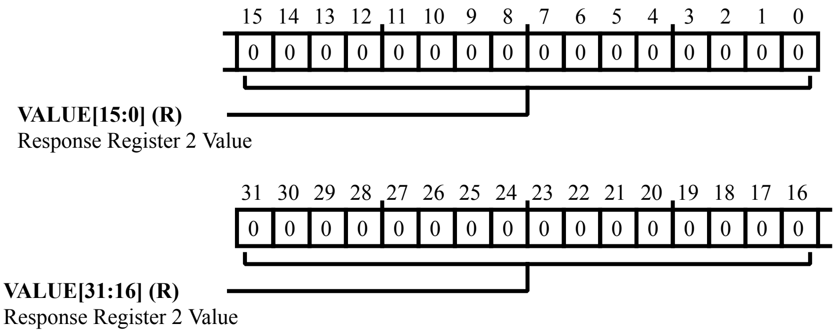

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Response Register 2 Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Response Register 2 Value

Figure 26-39: MSI_RESP2 Register Diagram

Table 26-42: MSI_RESP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 2 Value. The `MSI_RESP2.VALUE` bit field represents bits[95:64] of a long response.

Response Register 3

The `MSI_RESP3` register represents bits[127:96] of a long response.

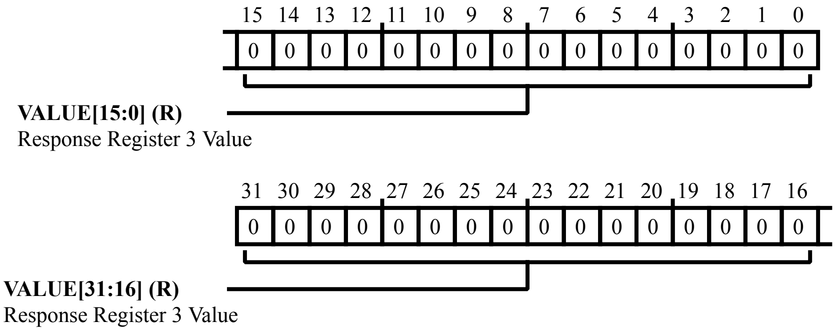

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Response Register 3 Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Response Register 3 Value

Figure 26-40: MSI_RESP3 Register Diagram

Table 26-43: MSI_RESP3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 3 Value. The `MSI_RESP3.VALUE` bit field represents bits[127:96] of a long response.

Status Register

The `MSI_STAT` register provides MSI DMA and data transfer status and information.

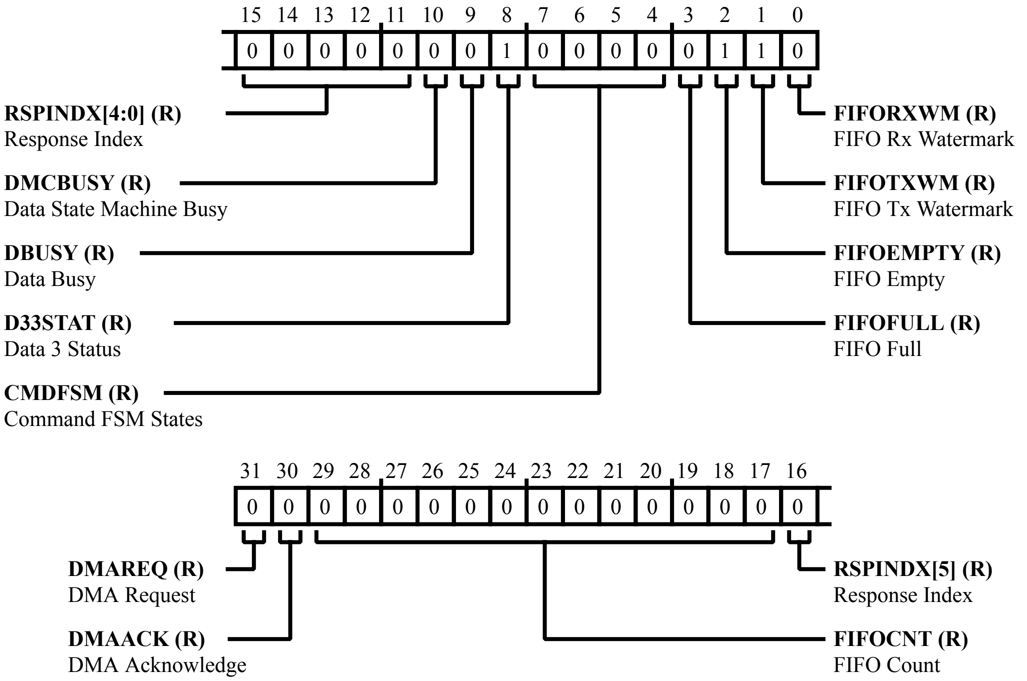

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
RSPINDX[4:0] (R)
Response Index
DMCBUSY (R)
Data State Machine Busy
DBUSY (R)
Data Busy
D33STAT (R)
Data 3 Status
CMDFSM (R)
Command FSM States
FIFORXWM (R)
FIFO Rx Watermark
FIFOTXWM (R)
FIFO Tx Watermark
FIFOEMPTY (R)
FIFO Empty
FIFOFULL (R)
FIFO Full
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DMAREQ (R)
DMA Request
DMAACK (R)
DMA Acknowledge
RSPINDX[5] (R)
Response Index
FIFOCNT (R)
FIFO Count

Figure 26-41: MSI_STAT Register Diagram

Table 26-44: MSI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	DMAREQ	DMA Request. The `MSI_STAT.DMAREQ` bit indicates the DMA request signal state.
30 (R/NW)	DMAACK	DMA Acknowledge. The `MSI_STAT.DMAACK` bit indicates the DMA acknowledge signal state.
29:17 (R/NW)	FIFOCNT	FIFO Count. The `MSI_STAT.FIFOCNT` bit field indicates the number of filled locations in the FIFO.
16:11 (R/NW)	RSPINDX	Response Index. The `MSI_STAT.RSPINDX` bit field indicates the index of the previous response, including any auto-stop sent by core.

Table 26-44: MSI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	DMCBUSY	Data State Machine Busy. The `MSI_STAT.DMCBUSY` bit indicates that the data transmit or receive state-machine is busy.	
		0	DMC idle
		1	DMC busy
9 (R/NW)	DBUSY	Data Busy. The `MSI_STAT.DBUSY` bit indicates the inverted version of the raw selected card_data[0].	
		0	Card data not busy
		1	Card data busy
8 (R/NW)	D33STAT	Data 3 Status.	
		0	Card not present
		1	Card present
7:4 (R/NW)	CMDFSM	Command FSM States. The `MSI_STAT.CMDFSM` bit field indicates the state machine status of CIU (card interface unit) or in general command FSM (not IDMA).	
		0	Idle
		1	Send init sequence
		2	Tx cmd start bit
		3	Tx cmd tx bit
		4	Tx cmd index + arg
		5	Tx cmd CRC7
		6	Tx cmd end bit
		7	Rx resp start bit
		8	Rx resp IRQ response
		9	Rx resp tx bit
		10	Rx resp cmd idx
		11	Rx resp data
		12	Rx resp CRC7
		13	Rx resp end bit
		14	Cmd path wait NCC
		15	Wait; CMD-to-response turnaround

Table 26-44: MSI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	FIFOFULL	FIFO Full. The `MSI_STAT.FIFOFULL` bit indicates that the FIFO is full.	
2 (R/NW)	FIFOEMPTY	FIFO Empty. The `MSI_STAT.FIFOEMPTY` bit indicates that the FIFO is empty.	
1 (R/NW)	FIFOTXWM	FIFO Tx Watermark. The `MSI_STAT.FIFOTXWM` bit indicates that the FIFO reached the transmit watermark level and is not qualified with data transfer.	
		0	Did not reach watermark level
		1	Reached watermark level
0 (R/NW)	FIFORXWM	FIFO Rx Watermark. The `MSI_STAT.FIFORXWM` bit indicates that the FIFO reached the receive watermark level and is not qualified with data transfer.	
		0	Did not reach watermark level
		1	Reached watermark level

Transferred Host to BIU-FIFO Byte Count Register

The MSI_TBBCNT register provides the number of bytes transferred between host/DMA memory and BIU FIFO. The register should be accessed in full to avoid read-coherency problems.

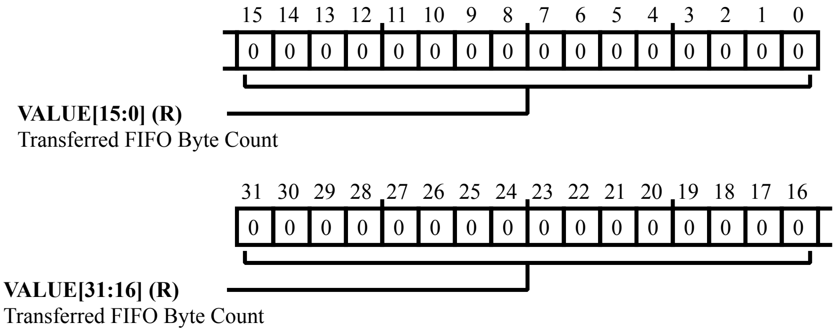

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Transferred FIFO Byte Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Transferred FIFO Byte Count

Figure 26-42: MSI_TBBCNT Register Diagram

Table 26-45: MSI_TBBCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Transferred FIFO Byte Count. The `MSI_TBBCNT.VALUE` bit field provides the number of bytes transferred between the host/DMA memory and the BIU FIFO.

Transferred CIU Card Byte Count Register

The `MSI_TCBCNT` register provides the number of bytes transferred by the CIU unit to a card. This register should be accessed in full to avoid read-coherency problems. Both the `MSI_TCBCNT` and `MSI_TBBCNT` registers share the same coherency register.

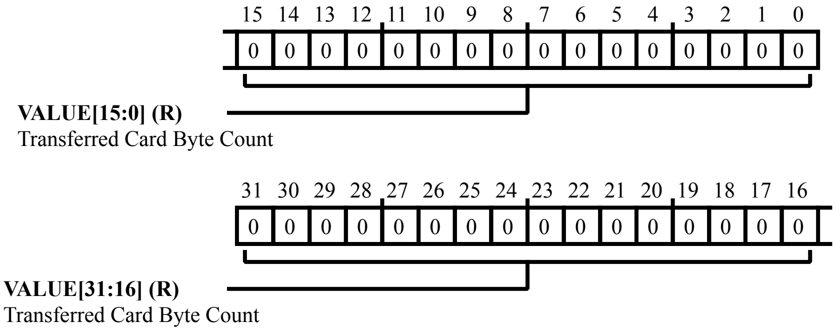

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Transferred Card Byte Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Transferred Card Byte Count

Figure 26-43: MSI_TCBCNT Register Diagram

Table 26-46: MSI_TCBCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Transferred Card Byte Count. The `MSI_TCBCNT.VALUE` bit field provides the number of bytes transferred by the CIU unit to a card.

Timeout Register

The `MSI_TMOUT` register provides bits that configure card data read and response timeout values.

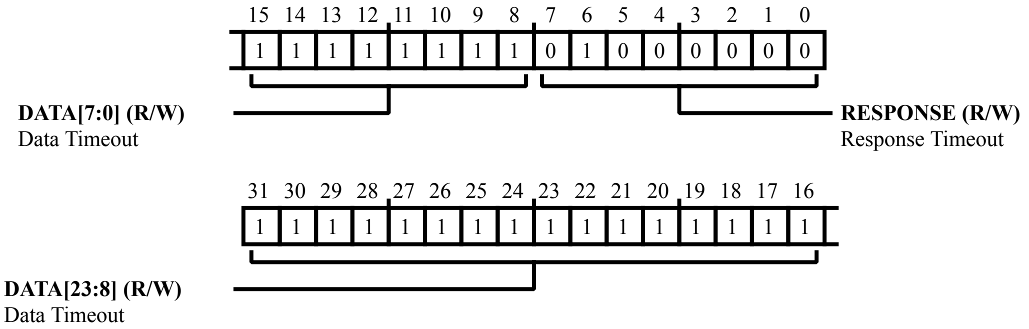

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0
DATA[7:0] (R/W)
Data Timeout
RESPONSE (R/W)
Response Timeout
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
DATA[23:8] (R/W)
Data Timeout

Figure 26-44: MSI_TMOUT Register Diagram

Table 26-47: MSI_TMOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (R/W)	DATA	Data Timeout. The `MSI_TMOUT.DATA` bit field configures the value for the card data read timeout; the same value is also used for the data starvation by host timeout. The timeout counter is started only after the card clock is stopped. The value is in number of card output clocks cclk_out of a selected card.
7:0 (R/W)	RESPONSE	Response Timeout. The `MSI_TMOUT.RESPONSE` bit field configures the response timeout in number of card output clocks.

27 Universal Serial Bus (USB)

The USB OTG controller provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras, and MP3 players. It allows these devices to transfer data using a point-to-point USB connection without the need for a personal computer host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification.

NOTE: See the On-The-Go Supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF and the Universal Serial Bus Specification 2.0.

The USB module supports:

- Host mode transfers at high-speed (480 Mbp/sec) rate
- Host mode transfers at full-speed (12 Mbp/sec) rate
- Host mode transfers at low-speed (1.5 Mbp/sec) rates. The connection to low-speed devices is only possible through a full-speed hub.
- Peripheral mode transfers at high-speed (480 Mbp/sec) rate
- Peripheral mode transfers at full-speed (12 Mbp/sec) rate

The USB controller uses a peripheral bus slave interface to access its control and status registers as well as read and write to the endpoint packet buffers. Data transfers to and from the USB controller through the 11 transmit and 11 receive endpoint FIFOs (EP1 – EP11), providing a total of 22 data endpoints.

USB Features

The USB controller provides the following features:

- Low-speed, full-speed, and high-speed rates supported
- One bidirectional control endpoint
- 11 transmit and 11 receive unidirectional endpoints
- 16 KB dynamically configured FIFO RAM

- Eight DMA master channels
- Two top-level maskable general-purpose interrupts
- Low-power wake-up on activity
- VBUS control interrupts for external analog VBUS control
- Session request protocol (SRP) and host negotiation protocol (HNP) capability
- Host transaction scheduling in hardware
- Soft connect or disconnect feature
- Full- and high-speed physical layer UTMI+ level 3 interface for on-chip PHY
- Backwards compatible with existing USB 1.1 hosts
- Support for Battery Charging Specification Revision 1.1

Only device requirements or system bandwidth limit the number of active endpoints at one time because each endpoint operates independently from the next. Software determines the type of transfer for each endpoint individually and also the manner in which it is transferred between the USB controller and memory (DMA or interrupt-based). The USB uses endpoint zero solely for receive and transmit control transfer. These transfers are used for device configuration and information gathering.

USB Functional Description

The following sections describe the function of the USB OTG interface.

USB Architectural Concepts

The USB controller operates in either of two USB operation modes (peripheral or host mode) at a given time.

In peripheral mode, the USB controller encodes, decodes, checks, and directs all USB packets sent and received, responding appropriately to host requests. Data is transferred from the processor core memory into the Tx FIFOs of the device onto USB as IN packets. In the other direction, USB OUT packets are received into the Rx FIFOs (having been sent from the host) and transferred to system memory for processing or storage. In peripheral mode, the USB controller acts as a slave device to another USB host; either a personal computer or another OTG host controller.

When operating in host mode, the USB controller uses simple hosting capabilities to master point-to-point connections with another USB peripheral, initiating transfers on the bus for the peripheral to respond. USB IN packets are received into the Rx FIFOs for transfer into the processor core memory. Data written into Tx FIFOs is transmitted onto the bus as USB OUT packets. In this mode, the USB controller encodes, decodes, and checks USB packets sent and received. The controller automatically schedules isochronous and interrupt transfers from the endpoint buffers. It performs one transaction every *n* frames, where n represents the polling interval programmed for the endpoint.

Any of the endpoints can be programmed to be written to or read from using the DMA master channels. This configuration provides the most efficient means of transferring data between the controller and on-chip memory.

USB endpoints 0 through 11 have DMA interrupt lines (`USB_DMA_IRQ`) providing a total of eight DMA request lines.

The USB provides two top-level maskable interrupts, each of which can be sourced from any or all of transmit endpoint status, receive endpoint status or global USB status. See Interrupt Signals for details.

The RAM interface of the USB controller supports a single block of synchronous single-port RAM used to buffer the USB packets.

16K bytes of SRAM are available.

The UTMI+ level 3 PHY interface provides a means of connecting a selection of high- or full-speed PHYs to the controller, from device-only PHYs through full OTG-compliant PHYs.

For details of the PHY interface, See UTMI Interface.

ATTENTION: Check the processor data sheet for requirements regarding minimum system clock frequency needed for proper USB operation.

The USB controller is configured as either a USB OTG A device or B device depending on the type of plug inserted into its USB receptacle. The state of the `USB_ID` (connector ID) pin determines this configuration.

The USB controller uses an asynchronous wake-up circuit to detect when another B device is asserting its D+ pull-up. This activity initiates the SRP (session request protocol) when all other clocks are off.

This slow clock is derived from SCLK0_0 and enabled using the `USB_PHY_CTL.EN` bit.

Use of the controller for OTG functionality requires the capability to:

- Drive VBUS (as a default A device powering the bus)
- Discharge VBUS (speeding up the time for VBUS to fall below the *SessionEnd* threshold as a B device checking initial conditions)
- Charge VBUS to 2.1 V (when initiating SRP as a B device).

The UTMI interface drives these controls, but the controller also provides a separate interrupt register, `USB_VBUS_CTL`, which represents the drive VBUS, discharge VBUS, and charge VBUS signaling. See the register section for more information on these controls.

Multi-Point Support

The USB controller has the facility, when operating in host mode, to act as the host to a range of USB peripheral devices.

High-speed, full-speed, or low-speed devices connect to the USB controller through a USB hub.

The USB controller, as part of its support for multiple devices, permits individual allocation of the functions of the target to the different Rx and Tx endpoints implemented. Furthermore, the USB controller can make this allocation dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The numbers of Tx and Rx endpoints implemented in the controller limit the combinations of peripheral devices that can be used together. Devices can only be added where the required endpoints remain available.

On-Chip Bus Interfaces

The USB controller uses two 32-bit wide independent bus interfaces, a master and a slave, to communicate with a processor-based subsystem. The slave interface allows the processor core to access the control and status registers (including DMA master registers) and the endpoint FIFOs. The integrated DMA uses the master interface to drive data into or out of the endpoint FIFOs with minimal processor core interaction. For more information, see USB Block Diagram.

FIFO Configuration

Each bidirectional endpoint (provided as two unidirectional endpoints) has its own endpoint number (0 for control, 1 on up for data transfer). Although two endpoints could use the same number, the endpoints can support different transfer types. Each of these bidirectional endpoints has a fixed region of the SRAM in the USB controller to which it has access. This feature dictates to some extent the types of transfers that can be used for that particular endpoint. This restriction follows from the maximum size of USB packets, which varies with each transfer type. The ***FIFO Sizes and Transfer Types*** table lists the endpoint FIFO configuration, with an indication of the transfer types possible for that particular buffer size.

Table 27-1: FIFO Sizes and Transfer Types

Bidirectional Endpoint (Rx and Tx)	FIFO Size (each direction)	USB Transfer Types
0	64 bytes	Size fixed for control transfers
1–4	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous
1–11	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous

Each endpoint FIFO can buffer one or two packets (in double-buffered mode). Double-buffering is recommended for most applications to improve efficiency by reducing the frequency of servicing for each endpoint.

Double-buffering bulk transactions means that data transfers over the USB are not slowed when packets are loaded or unloaded from the FIFO in the time it takes to transfer a packet. Double-buffering isochronous transactions allows more time to load or unload the FIFO. It also allows the usage of the SOF interrupt to service the endpoint rather than the endpoint interrupt. This functionality has the following advantages:

- Easy detection of lost packets
- Regular interrupt timing (making it easier to source or sink the data)
- If the USB controller uses more than one isochronous endpoint, one interrupt can service all the endpoints.

The USB controller uses the transmit or receive FIFO address registers to specify the address of each endpoint FIFO.

Clocking

The USB controller uses the system clock SCLK0_0 to generate an internal clock (CLK) used to clock the USB registers.

For proper operation, refer to device datasheet for minimum system clock SCLK0_0 value.

NOTE: For best performance (best signal integrity), follow the guidelines in the data sheet for selecting an input clock frequency.

When the controller is in the SUSPEND state and when no session is active, the clock and much of the USB controller is stopped to reduce power consumption. The clock becomes operational again when RESUME signaling is detected on the USB lines.

UTMI Interface

The interface to the on-chip PHY uses the industry-standard UTMI+ (universal transceiver macro interface) level 3.

This interface provides full- and high-speed device and OTG functionality and supports communication to a hub.

The PHY is a mixed-signal block and includes the following:

- Full-speed and high-speed drivers and receivers (single-ended and differential)
- Full-speed and high-speed CDR
- Full-speed or high-speed shift registers, NRZI encode or decode and bit-stuff encode or decode
- Data line pull-up and pull-down resistors
- VBUS and `USB_ID` level detection
- Host disconnect detection

Although the UTMI specification indicates that VBUS charging, driving and discharging happen inside the PHY, for process-restricting and power reasons, implement these functions off-chip in a separate USB charge-pump chip.

ADSP-SC58x USB Register List

The Universal Serial Bus controller (USB) is a multi-point high-speed dual-role USB 2.0-compliant controller. The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF. A set of registers governs USB controller operations. For more information on USB controller functionality, see the USB controller register descriptions.

Table 27-2: ADSP-SC58x USB Register List

Name	Description
`USB_BAT_CHG`	Battery Charging Control Register
`USB_CT_HHSRTN`	Host High-Speed Return to Normal Register

Table 27-2: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_CT_HSBT	High-Speed Timeout Register
USB_CT_UCH	Chirp Timeout Register
USB_DEV_CTL	Device Control Register
USB_DMA[n]_ADDR	DMA Channel n Address Register
USB_DMA[n]_CNT	DMA Channel n Count Register
USB_DMA[n]_CTL	DMA Channel n Control Register
USB_DMA_IRQ	DMA Interrupt Register
USB_EP0I_CFGDATA[N]	EP0 Configuration Information Register
USB_EP0I_CNT[N]	EP0 Number of Received Bytes Register
USB_EP0I_CSR[N]_H	EP0 Configuration and Status (Host) Register
USB_EP0I_CSR[N]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0I_NAKLIMIT[N]	EP0 NAK Limit Register
USB_EP0I_TYPE[N]	EP0 Connection Type Register
USB_EP0_CFGDATA[n]	EP0 Configuration Information Register
USB_EP0_CNT[n]	EP0 Number of Received Bytes Register
USB_EP0_CSR[n]_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSR[n]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0_NAKLIMIT[n]	EP0 NAK Limit Register
USB_EP0_TYPE[n]	EP0 Connection Type Register
USB_EPINFO	Endpoint Information Register
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register

Table 27-2: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_IDCTL	ID Control
USB_IEN	Common Interrupts Enable Register
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register
USB_LPM_IEN	LPM Interrupt Enable Register

Table 27-2: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_VBUS_CTL	VBUS Control Register
USB_VPLEN	VBUS Pulse Length Register

ADSP-SC58x USB Interrupt List

Table 27-3: ADSP-SC58x USB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
132	USB0_STAT	USB0 Status/FIFO Data Ready	Level	
133	USB0_DATA	USB0 DMA Status/Transfer Complete	Level	
134	USB1_STAT	USB1 Status/FIFO Data Ready	Level	
135	USB1_DATA	USB1 DMA Status/Transfer Complete	Level	

ADSP-SC58x USB Trigger List

Table 27-4: ADSP-SC58x USB Trigger List Masters

Trigger ID	Name	Description	Sensitivity
72	`USB0_DATA`	USB0 DMA Status/Transfer Complete	Level
73	`USB1_DATA`	USB1 DMA Status/Transfer Complete	Level

Table 27-5: ADSP-SC58x USB Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

USB Block Diagram

The ***USB OTG Controller Block Diagram*** shows the functional blocks within the USB. For more information about the blocks, see the USB Functional Description.

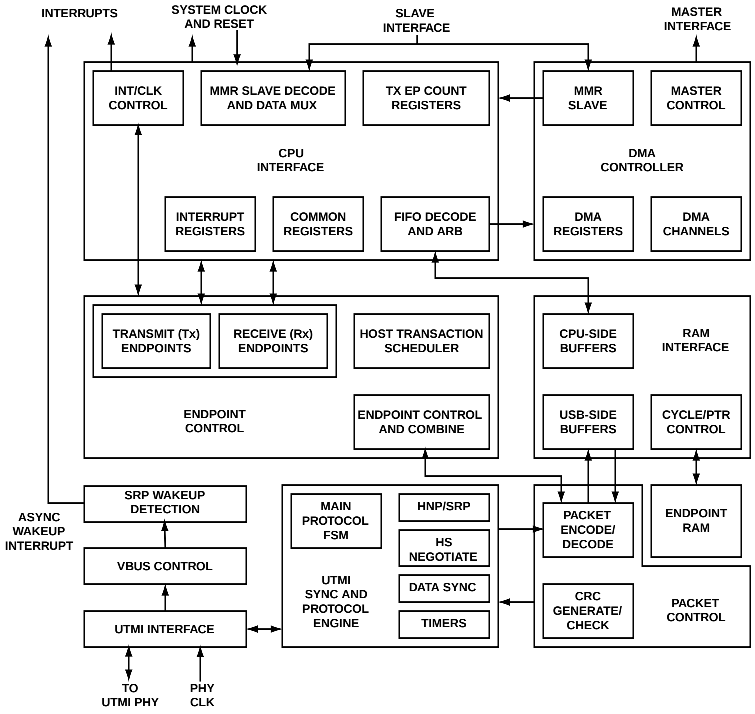

INTERRUPTS
SYSTEM CLOCK AND RESET
SLAVE INTERFACE
MASTER INTERFACE
INT/CLK CONTROL
MMR SLAVE DECODE AND DATA MUX
TX EP COUNT REGISTERS
MMR SLAVE
MASTER CONTROL
CPU INTERFACE
DMA CONTROLLER
INTERRUPT REGISTERS
COMMON REGISTERS
FIFO DECODE AND ARB
DMA REGISTERS
DMA CHANNELS
TRANSMIT (Tx) ENDPOINTS
RECEIVE (Rx) ENDPOINTS
HOST TRANSACTION SCHEDULER
CPU-SIDE BUFFERS
RAM INTERFACE
ENDPOINT CONTROL
ENDPOINT CONTROL AND COMBINE
USB-SIDE BUFFERS
CYCLE/PTR CONTROL
SRP WAKEUP DETECTION
ASYNC WAKEUP INTERRUPT
VBUS CONTROL
UTMI INTERFACE
MAIN PROTOCOL FSM
HNP/SRP
HS NEGOTIATE
UTMI SYNC AND PROTOCOL ENGINE
DATA SYNC
TIMERS
PACKET ENCODE/ DECODE
ENDPOINT RAM
CRC GENERATE/ CHECK
PACKET CONTROL
TO UTMI PHY
PHY CLK

Figure 27-1: USB OTG Controller Block Diagram

USB Definitions

A list of common USB terms and their definitions as used in this specification and based on the USB controller follows:

'A' Device

The USB device with a mini-A plug inserted into its receptacle. The A device always supplies power to VBUS.

'B' Device

The USB device with a standard-B or mini-B plug inserted into its receptacle. The B device starts a session as the peripheral.

Bidirectional endpoint

An endpoint that can concurrently support both receive and transfer packets.

Control endpoint

An endpoint used only for transfer of USB control packets for setup and configuration. In all USB devices, the control endpoint refers to the bidirectional endpoint 0.

Dual role device

A USB device that can operate either as the USB host in an OTG session or as a traditional USB peripheral.

Endpoint

A single physical communication channel for USB, implemented as a FIFO and control logic for that endpoint. Each endpoint has an associated USB transfer type, maximum packet size, bandwidth requirement, endpoint number, and (often) a fixed transfer direction.

Frame

A regular, fixed 1 ms timeslot that can contain several transactions. The transfer type determines the permissible transactions for a given endpoint.

HNP

Host negotiation protocol. Part of the USB OTG supplement that allows the host function to be transferred between two connected dual role devices.

Packet

The lowest level of data exchange on USB. The transfer type and buffer size of the USB peripheral determine the size.

PHY

The PHY is a transceiver circuit that implements the physical layer of USB. For full speed USB OTG, this circuit includes line drivers and receivers, pull-up, or pull-down resistors as well as device ID and VBUS level detection.

Session

A period during which USB transfers take place within an OTG connection. The A device (drives VBUS) or B device (initiates SRP) can initiate this period. VBUS is powered during a session.

SRP

Session request protocol. Part of the USB OTG supplement that allows a B device to turn on VBUS and initiate a USB session.

Transaction

Collection of one or more packets in sequence

Transfer

Collection of one or more transfers in sequence

Unidirectional endpoint

Endpoint with its direction fixed in a single direction (for example, it can only receive packets from the USB) in both host and peripheral modes.

USB References

The following references provide further information regarding the USB.

- *On-The-Go Supplement to the USB 2.0 Specification*, Rev 1.0a, June 24, 2003, USB-IF
- *Universal Serial Bus Specification 2.0*

USB Operating Modes

The USB OTG interface can operate in peripheral mode or host mode.

When the USB controller operates in peripheral mode, the controller can be attached to a conventional host (such as a personal computer) or another OTG device operating in host mode. The second device can be high speed or full speed. When linked to another peripheral device, the USB controller can also act as the host. If the other device is also a dual role controller, the two devices can switch roles, as needed.

The role the USB controller takes depends on the way the devices are cabled together. Each USB cable has an A and a B device end. If the A end of the cable is plugged into the device containing the USB controller, the USB controller takes the role of the host device. It goes into host mode (in this case, the `USB_DEV_CTL.HOSTMODE` bit is set to 1). If the B of the cable is plugged in, the USB controller goes instead into peripheral mode (and the `USB_DEV_CTL.HOSTMODE` bit remains at 0).

When both devices contain dual role controllers, signaling can be used to switch the roles of the two devices, without switching the cable connecting the two devices. See Host Negotiation Protocol for details on the conditions under which the USB controller can switch between peripheral and host mode.

NOTE: The multi-point capability of the USB controller is associated with a range of registers recording the allocation of device functions to individual endpoints and device function characteristics. These characteristics include endpoint number, operating speed, and transaction type on an endpoint-by-endpoint basis. These registers are principally associated with the use of the USB controller as the host to a number of devices. However, set the registers when the core is used as the host for a single target device.

To enable the USB:

1. Configure the USB PLL multiplier settings in the USB PLL control register. Check the processor data sheet for the requirements for input clock frequency.
2. Enable the USB PHY by setting the `USB_PHY_CTL.EN` bit.
3. Poll the bit in the USB PLL control register to ensure that the USB PLL has locked to the new frequency.

Peripheral Mode

USB OTG interface operations for the peripheral mode differ from host mode in a number of ways. The following sections describe peripheral mode operations.

Endpoint Setup

In peripheral mode, the USB uses a few endpoint-specific configuration bits when setting up an endpoint for transfer for all types of peripheral transfer. The configuration determines how the processor core interacts with the endpoint FIFO.

One key parameter required before a transfer can occur through an endpoint is the maximum USB packet size that the endpoint can support. The software sets this value. It depends on various system constraints. These constraints include the size of hardware FIFO available and system latencies as well as the USB transfer type and class used. The `USB_EP[n]_TXMAXP` or `USB_EP[n]_RXMAXP` registers define the maximum amount of data that can be transferred to the selected endpoint in a single frame. The value must match the programmed maximum individual packet size (*MaxPktSize*) of the standard endpoint descriptor for the endpoint.

For transmit endpoints, program the maximum packet size using the `USB_EP[n]_TXMAXP`. For receive endpoints, the USB uses the `USB_EP[n]_RXMAXP` register. The maximum packet size must not exceed the actual hardware endpoint FIFO size.

The settings in the USB_RXFIFOSZ or USB_TXFIFOSZ register determine the corresponding sizes of the transmit or receive FIFOs, as well as, single or double buffered mode for endpoints 1 to 11.

Because the USB controller uses a 32-bit interface, choose an even number for the value of *MaxPktSize*. This selection simplifies transferring data between FIFOs and the processor core.

Configure more setup parameters using the USB_EP[n]_TXCSR_H or USB_EP[n]_RXCSR_H register (depending on whether the endpoint in question is for receive or transmit). The USB uses the USB_EP[n]_RXCSR_H.DMAREQEN bit in this register to enable the assertion of the appropriate DMA request whenever the endpoint is able to receive or transmit another packet. The USB uses the USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits to set the FIFO ready triggers (USB_EP[n]_RXCSR_H.RXPKTRDY and USB_EP[n]_TXCSR_H.TXPKTRDY) automatically whenever a packet is transferred to streamline DMA operation for transfers that span multiple packets. Note, however, that the USB cannot use USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits with high-bandwidth endpoints. Refer to the following register sections for more information: USB_EP[n]_TXCSR_H,USB_EPI[N]_RXCSR_P,USB_EPI[N]_RXCSR_H,USB_EP[n]_TXCSR_P.

IN Transactions as a Peripheral

When the USB controller operates in peripheral mode, the transmit FIFOs handle data for IN transactions. The maximum size of data packet that can be placed in a FIFO for a transmit endpoint is programmable. When applicable, the value written to the USB_EP[n]_TXMAXP register for that endpoint determines the size (maximum payload multiplied by the number of transactions per micro-frame).

The maximum packet size set for any endpoint must not exceed the FIFO size. (See FIFO Configuration.)

ATTENTION: Do not write to the USB_EP[n]_TXMAXP register while there is data in the FIFO, as unexpected results can occur.

The following sections describe the two types of packet buffering used for IN transactions.

Single packet buffering. Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for transmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, and where auto-set cannot be used (high-bandwidth isochronous or interrupt transactions), always set the USB_EP[n]_TXCSR_P.TXPKTRDY bit manually (for example by the processor core).

When the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EP[n]_TXCSR_P.NEFIFO bit is also set and the packet is ready to be sent. When the packet is successfully sent, both the USB_EP[n]_TXCSR_P.TXPKTRDY and USB_EP[n]_TXCSR_P.NEFIFO bits are cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled). The next packet can then be loaded into the FIFO.

Double packet buffering. Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for transmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the

FIFO. For packet sizes less than the maximum, and where auto-set cannot be used (high-bandwidth isochronous or interrupt transactions), always set the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit manually (for example by the processor core).

When the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set, either manually or automatically, the `USB_EP[n]_TXCSR_P.NEFIFO` bit also is set. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is then immediately cleared (and an interrupt generated, if enabled). A second packet can now be loaded into the transmit FIFO and the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set again (either manually or automatically if the packet is the maximum size). Both packets are now ready for transmission.

When the first packet is successfully sent, the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled) to signal that another packet can now be loaded into the transmit FIFO. The state of the `USB_EP[n]_TXCSR_P.NEFIFO` bit indicates how many packets can be loaded. If the `USB_EP[n]_TXCSR_P.NEFIFO` bit is set, then there is another packet in the FIFO and only one more packet can be loaded. If the `USB_EP[n]_TXCSR_P.NEFIFO` bit is cleared, then there are no packets in the FIFO and two more packets can be loaded.

OUT Transactions as a Peripheral

When the USB controller operates in peripheral mode, the receive FIFOs handle data for OUT transactions.

The value written to the `USB_EP[n]_RXMAXP` register for an endpoint determines the maximum amount of data received by a receive endpoint in any frame. The value is programmable. The maximum packet size must not exceed the FIFO size.

The value written to the `USB_EP[n]_RXMAXP` register for an endpoint determines maximum amount of data received by a receive endpoint in any micro-frame (in high-speed mode). The value is programmable. It is the maximum payload multiplied by the number of transactions per micro-frame. The maximum packet size must not exceed the FIFO size.

If the size of the receive endpoint FIFO is less than twice the maximum packet size for this endpoint, only one data packet can be buffered in the FIFO. Single buffering is selected. (The size is set in the `USB_EP[n]_RXMAXP` register.) When a packet is received and placed in the receive FIFO, the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit and the `USB_EP[n]_RXCSR_P.FIFOFULL` bit are set. The USB controller generates the appropriate receive endpoint interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. After the packet is unloaded, clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit to allow reception of more packets. If the `USB_EP[n]_RXCSR_P.AUTOCLR` bit is set and a maximum-sized packet is unloaded from the FIFO, the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is cleared automatically. The `USB_EP[n]_RXCSR_P.FIFOFULL` bit is also cleared. For packet sizes less than the maximum, clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit manually (for example by the processor core).

If double packet buffering is enabled, then two data packets can be buffered. When the first packet for reception is loaded into the receive FIFO, the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is set. The USB controller generates the appropriate receive endpoint interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. The `USB_EP[n]_RXCSR_P.FIFOFULL` bit is not set. This bit is only set if a second packet is received and loaded into the receive FIFO.

After the first packet is unloaded, clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit to allow reception of further packets. If the `USB_EP[n]_RXCSR_P.AUTOCLR` bit is set and a maximum-sized packet is unloaded from the FIFO, the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is cleared automatically. For packet sizes less than the maximum, clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit manually (for example by the processor core).

If the `USB_EP[n]_RXCSR_P.FIFOFULL` bit is set to 1 when `USB_EP[n]_RXCSR_P.RXPKTRDY` is cleared, the USB controller first clears the `USB_EP[n]_RXCSR_P.FIFOFULL` bit. The controller then sets the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit again, indicating that there is another packet waiting in the FIFO for unloading.

High-Bandwidth Isochronous or Interrupt Transactions

High-bandwidth isochronous or interrupt transactions use much the same protocol as other isochronous or interrupt transactions. There are, however, some special features to conducting high-bandwidth transactions.

- When setting the maximum packet size handled by the endpoint in the `USB_EP[n]_TXMAXP`/ `USB_EP[n]_RXMAXP` registers, set the maximum number of transactions per micro-frame using the `USB_EP[n]_TXMAXP.MULTM1` and `USB_EP[n]_RXMAXP.MULTM1` bits.

  The maximum number of transactions (2 or 3) also represents the maximum number of sections in which any single high-bandwidth packet can be transferred. The configuration sets the maximum size of the packet to 2 or 3 times the maximum payload specified for the endpoint in the same register.

  NOTE: The maximum payload that can be sent in any transaction is 1K byte.

- When sending packets, set the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit using the application software. Similarly, when unloading packets from the receive endpoint FIFO, clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit using the application software.

  CAUTION: The AutoSet and AutoClear functions cannot be used to set and clear these bits in high-bandwidth transactions.

- The transmission of packets as a number of sections introduces a further type of error – the transmission of incomplete packets.

  For transmit endpoints, transmitting incomplete packets principally applies when the interface is in peripheral mode. It occurs when the transmission fails to receive enough IN tokens from the host to send all the parts of the data packet. It can also apply to high-bandwidth interrupt transactions in host mode where the core does not receive any response from the device to which the packet is transmitted. In both cases, the `USB_EP[n]_TXCSR_P.INCOMPTX` bit is set.

  For receive endpoints, an incomplete packet issue can occur. The PIDs of the received parts of the data packet show that one or more parts of the data packet have not been received. When this event happens, the `USB_EP[n]_RXCSR_P.INCOMPRX` bit is set. Usually this bit is set in peripheral mode. However, it can also be set in host mode (using the `USB_EP[n]_RXCSR_H.INCOMPRX` bit). This event occurs when the USB communicates with a device that fails to respond in accordance with the USB protocol.

High Bandwidth Isochronous or Interrupt IN Endpoints

In high-speed mode, transmit endpoints configured for high-bandwidth isochronous or interrupt transactions can transmit up to three USB packets in any micro-frame. The transmission occurs with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame.

The ***High Bandwidth IN Endpoints*** figure provides an overview of high-bandwidth IN endpoints in USB.

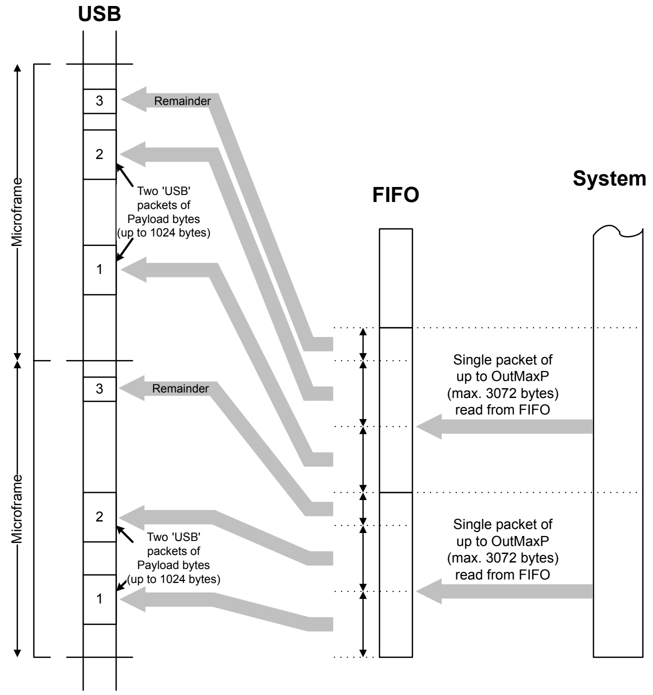

USB
FIFO
System
Microframe
Microframe
3
Remainder
2
Two 'USB' packets of Payload bytes (up to 1024 bytes)
1
3
Remainder
2
Two 'USB' packets of Payload bytes (up to 1024 bytes)
1
Single packet of up to OutMaxP (max. 3072 bytes) read from FIFO
Single packet of up to OutMaxP (max. 3072 bytes) read from FIFO

Figure 27-2: High Bandwidth IN Endpoints

The USB controller supports these transfers by permitting the loading of data packets with up to three times the normal packet size into the associated FIFO in a single transaction. From the software viewpoint in the processor core, the ***High Bandwidth IN Endpoints*** figure describes the operation for single or double packet buffering (as appropriate). One exception is that the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit must always be set manually (for example by the processor core) as the auto set feature does not operate with high-bandwidth isochronous transfers.

The USB controller automatically splits any data packet loaded into the FIFO that is larger than the maximum into USB packets of the maximum payload, or smaller, for transmission. The following settings define the number of USB packets transmitted per micro-frame and the maximum payload in each packet:

- Use the `USB_EP[n]_TXMAXP.MAXPAY` bits to set the maximum payload in any USB packet
- Use the `USB_EP[n]_TXMAXP.MULTM1` bits to set the maximum number of such packets for transmission in one micro-frame (2 or 3)

Together, these settings define the maximum size of packet that can be loaded into the FIFO.

At least one USB packet always is sent. The number of further USB packets sent in the same micro-frame depends on the amount of data loaded into the FIFO. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is cleared and an interrupt is generated only when all the packets have been sent. Each USB packet is sent in response to an IN token. If, at the end of a micro-frame, the USB controller has not received enough IN tokens to send all the USB packets, the remaining data is flushed from the FIFO. (For example, one of the IN tokens received is corrupt). The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is cleared and the `USB_EP[n]_TXCSR_P.INCOMPTX` bit is set to indicate that not all of the data loaded into the FIFO transmitted.

High-Bandwidth Isochronous or Interrupt OUT Endpoints

In high-speed mode, isochronous receive endpoints can receive up to three USB packets in any micro-frame. The reception occurs with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame. Similarly, the USB controller can receive high-bandwidth interrupt transactions in host mode, but there is no support for high-bandwidth interrupt transactions in peripheral mode.

The ***High-Bandwidth OUT Endpoints*** figure shows an overview of high-bandwidth OUT endpoints.

The USB controller supports this rate by automatically combining all the USB packets received during a micro-frame into a single packet of up to 3 normal packets within the receive FIFO. From the software viewpoint in the processor core, the ***High Bandwidth IN Endpoints*** figure describes the operation for single or double packet buffering (as appropriate). One exception is that the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit always must be cleared manually (for example by the processor core) because the auto-clear function does not operate with high-bandwidth isochronous transfers.

The maximum number of USB packets that can be received in any micro-frame and the maximum payload of these packets are configured as follows:

- Use the `USB_EP[n]_RXMAXP.MAXPAY` bits to set the maximum payload in any USB packet
- Use the `USB_EP[n]_RXMAXP.MULTM1` bits to set the maximum number of these packets that can be received in a micro-frame (2 or 3)

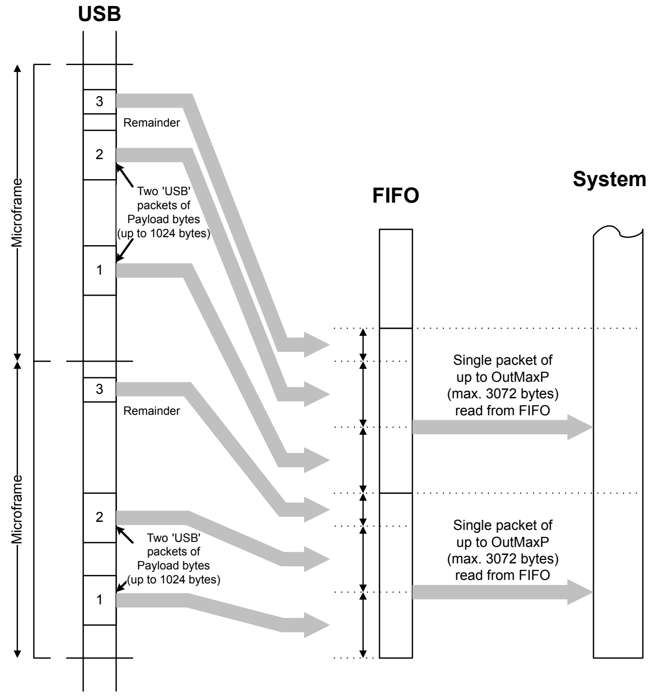

USB
FIFO
System
Microframe
Microframe
3
Remainder
2
Two 'USB' packets of Payload bytes (up to 1024 bytes)
1
3
Remainder
2
Two 'USB' packets of Payload bytes (up to 1024 bytes)
1
Single packet of up to OutMaxP (max. 3072 bytes) read from FIFO
Single packet of up to OutMaxP (max. 3072 bytes) read from FIFO

Figure 27-3: High-Bandwidth OUT Endpoints

The number of USB packets sent in any micro-frame depends on the amount of data for transfer, and is indicated through the PIDs used for the individual packets. If the indicated number of packets have not been received by the end of a micro-frame, the `USB_EP[n]_RXCSR_P.INCOMPRX` bit is set to indicate that the data in the FIFO is incomplete. An interrupt is still generated to allow the data that has been received to be read from the FIFO.

Peripheral Transfer Work Flows

The USB transfer types (control, bulk, isochronous, and interrupt transfers) have different system requirements as well as individual USB transfer-specific features. Software handles each type differently. There is no uniform way of doing transfers across all transfer types using the USB controller.

The following sections provide some guidelines for peripheral mode transfer flows for each of the transfer types, in both IN (transmit) and OUT (receive) directions. For bulk endpoints, the optimal transfer flow depends on whether the final size of the transfer is known or unknown. The USB driver class in use determines whether the transfer size is known or not. Some drivers define the complete transfer size, and others operate on a packet-by-packet basis using a short packet to denote the end of a transfer. (A short packet is less than the value configured in the `USB_EP[n]_TXMAXP` register or less than the value configured in the `USB_EP[n]_RXMAXP` register.)

Each of the work flows uses the following common steps.

1. Configure the endpoint control and status registers and the `USB_EP[n]_TXMAXP` or `USB_EP[n]_RXMAXP` value.
2. Configure the appropriate data transfer mechanism (DMA or interrupt setup).
3. Data transfer occurs.

The work flows do not describe the actions of the USB controller immediately preceding the endpoint setup. (For example, the reception of an IN/OUT token from the host, token validity checking, or NAK generation, among others.) Note also that there is no error-handling contained in the work flows (for example, checking the `USB_EP[n]_RXCSR_P.FIFOFULL` bit before writing data).

The proceeding sections use terms packets, frames, and transfers with their strict USB definitions (see USB Definitions).

Control Transactions as a Peripheral

Endpoint 0 is the main control endpoint of the USB controller. As such, the routines required to service endpoint 0 are more complicated than the routines required to service other endpoints.

The software is required to handle all the standard device requests that the USB controller sends or receives through endpoint 0. The *Universal Serial Bus Specification*, Revision 2.0, Chapter 9 describes the requirements. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this functionality, the processor must take a state machine approach to command decoding and handling.

The standard device requests a USB peripheral receives fits into three categories:

- Zero data requests (in which the command includes all the information)
- Write requests (in which extra data follows the command)
- Read requests (in which the device sends data back to the host)

The following sections describe the sequence of actions that the software must perform to process these different types of device request.

Write Requests

The host sends an 8-byte command followed by a write request that contains an extra packet (or packets) of data. An example of a write standard device request is `SET_DESCRIPTOR`.

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO.

As with a zero data request, write to the `USB_EP0_CSR[n]_P` register to set the `USB_EP0_CSR[n]_P.SPKTRDY` bit. (The event indicates that the host read the command from the FIFO.) But, in this case, do not set the `USB_EP0_CSR[n]_P.DATAEND` bit (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the `USB_EP0_CSR[n]_P` register is read to check the endpoint status. The `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set to indicate that a data packet is received. Read the `USB_EP0_CNT[n]` register to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request is greater than the maximum packet size for endpoint 0, the host sends more data packets. (The `WLENGTH` field in the command indicates the length of the data.) In this case, the `USB_EP0_CSR[n]_P.SPKTRDY` bit is set, but do not set the `USB_EP0_CSR[n]_P.DATAEND` bit.

When all the expected data packets have been received, software writes to the `USB_EP0_CSR[n]_P` register to set the `USB_EP0_CSR[n]_P.SPKTRDY` bit and to set the `USB_EP0_CSR[n]_P.DATAEND` bit (indicating that no more data is expected).

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or cannot be executed, then when the host decodes it, software must write to the `USB_EP0_CSR[n]_P` register. This operation sets the `USB_EP0_CSR[n]_P.SPKTRDY` bit and the `USB_EP0_CSR[n]_P.SENDSTALL` bit. When the host sends more data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the `USB_EP0_CSR[n]_P.SENTSTALL` bit is set.

If the host sends more data after the `USB_EP0_CSR[n]_P.DATAEND` has been set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the `USB_EP0_CSR[n]_P.SENTSTALL` bit is set.

Read Requests

The function sends the 8-byte command followed by read requests containing a packet (or packets) of data to the host. Examples of standard device requests for read are:

- `GET_CONFIGURATION`
- `GET_INTERFACE`
- `GET_DESCRIPTOR`
- `GET_STATUS`
- `SYNCH_FRAME`

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO. Write the `USB_EP0_CSR[n]_P.SPKTRDY` bit (indicating that the command has been read from the FIFO).

The data to transmit to the host is written to the endpoint 0 FIFO. If the size of the transmit data is greater than the maximum packet size for endpoint 0, only the maximum packet size is written to the FIFO. The `USB_EP0_CSR[n]_P.TXPKTRDY` bit is then set (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, software generates another endpoint 0 interrupt. The next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, software writes to the `USB_EP0_CSR[n]_P` register to set the `USB_EP0_CSR[n]_P.TXPKTRDY` bit and to set the `USB_EP0_CSR[n]_P.DATAEND` bit. (This activity indicates that there is no more data after this packet.)

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, software must write to the `USB_EP0_CSR[n]_P` register. This operation sets the `USB_EP0_CSR[n]_P.SPKTRDY` bit and the `USB_EP0_CSR[n]_P.SENDSTALL` bit. When the host requests data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the `USB_EP0_CSR[n]_P.SENTSTALL` bit is set.

If the host requests more data after `USB_EP0_CSR[n]_P.DATAEND` is set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the `USB_EP0_CSR[n]_P.SENTSTALL` bit is set.

Zero Data Requests

Zero data requests have all their information included in the 8-byte command and do not require transfer of extra data.

Examples of standard device requests for zero data are:

- `SET_FEATURE`
- `CLEAR_FEATURE`
- `SET_ADDRESS`
- `SET_CONFIGURATION`
- `SET_INTERFACE`

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The `USB_EP0_CSR[n]_P.RXPKTRDY` bit is also set. The host must then read and decode the 8-byte command from the endpoint 0 FIFO, and take appropriate action. For example, if the command is `SET_FEATURE`, the host writes 7-bit address value contained in the command to the `USB_FADDR` register.

Software must set the `USB_EP0_CSR[n]_P.SPKTRDY` bit (indicating that the command has been read from the FIFO) and the `USB_EP0_CSR[n]_P.DATAEND` bit (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, the USB controller generates a second endpoint 0 interrupt, indicating that the request has completed. No further action is required from the software; the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, the `USB_EP0_CSR[n]_P.SPKTRDY` bit is set which sets the `USB_EP0_CSR[n]_P.SENDSTALL` bit. When the host moves to the status stage of the request, the USB controller sends a stall to tell the host that the request was not executed. The USB controller generates a second endpoint 0 interrupt and sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit.

If the host sends more data after the `USB_EP0_CSR[n]_P.DATAEND` bit is set, then the USB controller sends a stall. It generates an endpoint 0 interrupt and sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit.

Endpoint 0 States

When the USB operates as a peripheral, the endpoint 0 control has three modes (IDLE, Tx, and Rx). The modes correspond to the phases of the control transfer and the state that endpoint 0 enters during phases of the transfer. (See Endpoint 0 Service Routine as Peripheral.)

IDLE is the default mode on power-up or reset. The processor sets `RxPktRdy` bit when endpoint 0 is in IDLE state, indicating a new device request. Once the processor unloads the device request from the FIFO, the USB decodes the descriptor. It determines whether there is a data phase and, if so, the direction of the data phase of the control transfer (to set the FIFO direction).

Depending on the direction of the data phase, endpoint 0 goes into either Tx state or Rx state. If there is no data phase, endpoint 0 remains in IDLE state to accept the next device request.

The processor must take different actions at the different phases of the possible transfers (for example, loading the FIFO, setting `TxPktRdy`). The ***Endpoint 0 Control States*** figure shows the actions for the phase. The USB changes the FIFO direction depending on the direction of the data phase, independently of the processor.

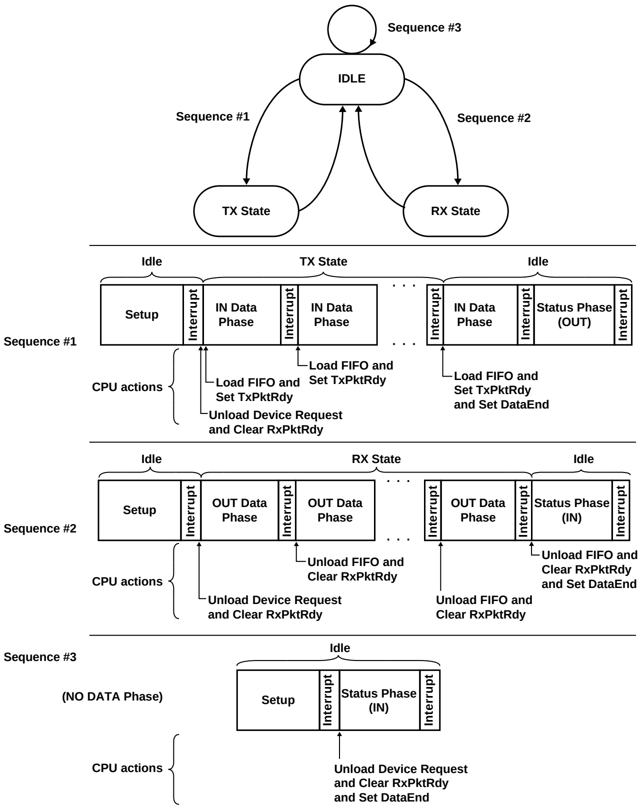

Sequence #3
IDLE
Sequence #1
Sequence #2
TX State
RX State
Idle
TX State
Idle
Setup
Interrupt
IN Data Phase
Interrupt
IN Data Phase
Interrupt
IN Data Phase
Interrupt
Status Phase (OUT)
Interrupt
Sequence #1
CPU actions
Load FIFO and Set TxPktRdy
Load FIFO and Set TxPktRdy
Unload Device Request and Clear RxPktRdy
Load FIFO and Set TxPktRdy and Set DataEnd
Idle
RX State
Idle
Setup
Interrupt
OUT Data Phase
Interrupt
OUT Data Phase
Interrupt
OUT Data Phase
Interrupt
Status Phase (IN)
Interrupt
Sequence #2
CPU actions
Unload FIFO and Clear RxPktRdy
Unload FIFO and Clear RxPktRdy and Set DataEnd
Unload Device Request and Clear RxPktRdy
Unload FIFO and Clear RxPktRdy
Sequence #3
(NO DATA Phase)
Idle
Setup
Interrupt
Status Phase (IN)
Interrupt
CPU actions
Unload Device Request and Clear RxPktRdy and Set DataEnd

Figure 27-4: Endpoint 0 Control States

Endpoint 0 Service Routine as Peripheral

The USB controller generates an endpoint 0 interrupt when:

- The USB controller sets the `USB_EP0_CSR[n]_P.RXPKTRDY` bit after a valid token has been received and data has been written to the FIFO.

- The USB controller clears the `USB_EP0_CSR[n]_P.TXPKTRDY` bit after the data packet in the FIFO has been successfully transmitted to the host.
- The USB controller sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit after a control transaction is ended due to a protocol violation.
- The USB controller sets the `USB_EP0_CSR[n]_P.SETUPEND` bit because a control transfer has ended before `USB_EP0_CSR[n]_P.DATAEND` is set.

Whenever the endpoint 0 service routine is entered, the firmware must first check whether the current control transfer has been ended. The transfer can end due to either a stall condition or a premature end-of-control transfer. If the control transfer ends due to a stall condition, the USB controller sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit. If the control transfer ends due to a premature end-of-control transfer, the USB controller sets `USB_EP0_CSR[n]_P.SETUPEND`. In either case, the firmware must abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that an illegal bus state did not generate the interrupt, the next action depends on the endpoint state.

If endpoint 0 is in IDLE state, the only valid reason the USB controller can generate an interrupt is due to the core receiving data from the USB bus. The service routine must check for this state by testing the `USB_EP0_CSR[n]_P.RXPKTRDY` bit. If the USB controller sets the bit, then the core has received a setup packet. The processor unloads this packet from the FIFO and decodes it to determine the next action. Depending on the command contained within the setup packet, endpoint 0 enters one of the following three states.

- If the command is a single packet transaction (`SET_ADDRESS`, `SET_INTERFACE` and the others) without a data phase, the endpoint remains in the IDLE state.
- If the command has an OUT data phase (`SET_DESCRIPTOR` and others), the endpoint enters the Rx state.
- If the command has an IN data phase (`GET_DESCRIPTOR` and others), the endpoint enters the Tx state.

NOTE: Command transactions all include a field that indicates the amount of data the host expects to receive or send.

If the endpoint is in Tx state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this event by:

- Placing more data in the FIFO when the host still expects more data
- Setting the `USB_EP0_CSR[n]_P.DATAEND` bit to indicate that the data phase is complete

Once the data phase of the transaction completes, endpoint 0 returns to the IDLE state to await the next control transaction.

If the endpoint is in the Rx state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data. If it has, the firmware must set the `USB_EP0_CSR[n]_P.DATAEND` bit and return endpoint 0 to IDLE state. If more data is expected, the firmware must set the `USB_EP0_CSR[n]_P.SPKTRDY` bit to indicate that it has read the data in the FIFO and leave the endpoint in the Rx state.

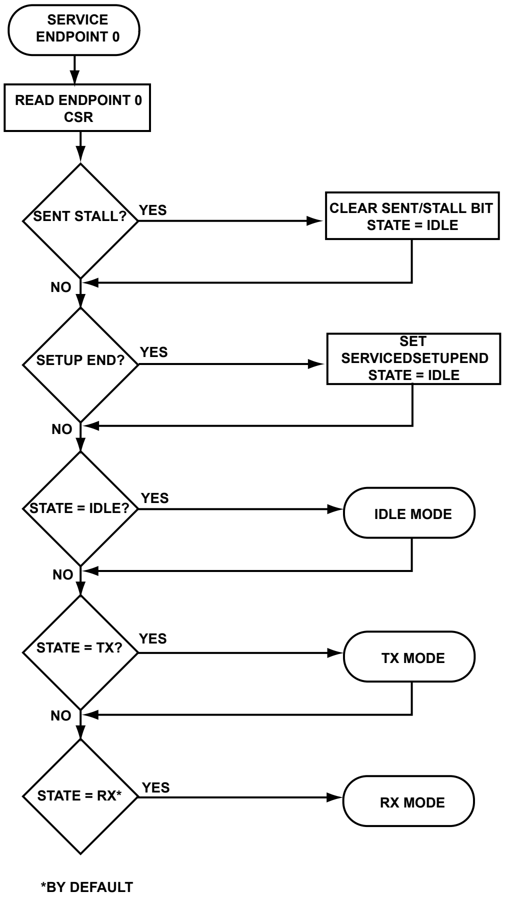

SERVICE ENDPOINT 0
READ ENDPOINT 0 CSR
SENT STALL?
YES
CLEAR SENT/STALL BIT
STATE = IDLE
NO
SETUP END?
YES
SET SERVICEDSETUPEND
STATE = IDLE
NO
STATE = IDLE?
YES
IDLE MODE
NO
STATE = TX?
YES
TX MODE
NO
STATE = RX*
YES
RX MODE
*BY DEFAULT

Figure 27-5: Endpoint 0 Service Routine

IDLE Mode

The endpoint 0 control must select IDLE mode at power-on or reset. The endpoint 0 control returns to this mode when the Rx and Tx modes terminate.

As shown in the ***Endpoint 0 Idle Mode (SETUP Phase)*** figure, the SETUP phase of control transfer is handled in IDLE mode.

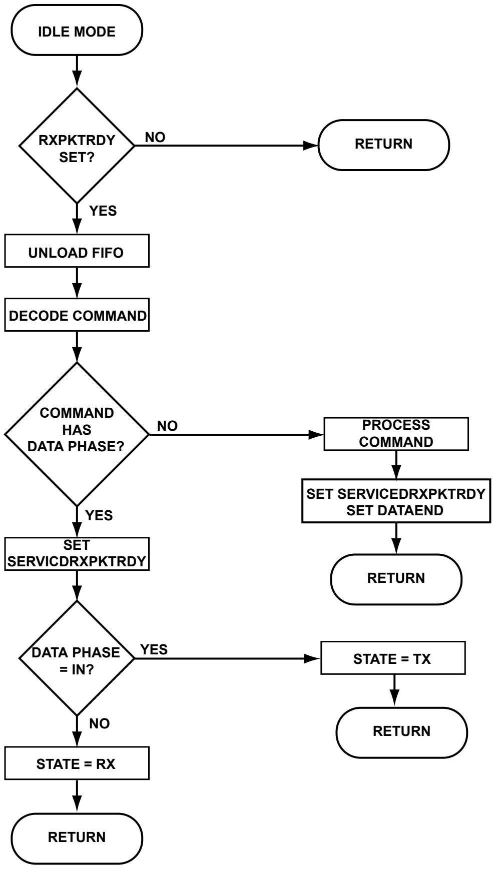

IDLE MODE
RXPKTRDY SET?
NO
RETURN
YES
UNLOAD FIFO
DECODE COMMAND
COMMAND HAS DATA PHASE?
NO
PROCESS COMMAND
SET SERVICEDRXPKTRDY
SET DATAEND
RETURN
YES
SET SERVICDRXPKTRDY
DATA PHASE = IN?
YES
STATE = TX
RETURN
NO
STATE = RX
RETURN

Figure 27-6: Endpoint 0 Idle Mode (SETUP Phase)

Tx Mode

Refer to the ***Endpoint 0 Tx Mode*** figure. When the endpoint is in Tx state, all arriving IN tokens must be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the Tx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs. The core expects only IN tokens.

Three events can cause the Tx mode to terminate before the expected amount of data has been sent:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The firmware sends a packet containing less than the maximum packet size for endpoint 0.
- The firmware sends an empty data packet.

Until the transaction is terminated, when the firmware receives an interrupt which indicates that a packet has been sent from the FIFO, it simply loads the FIFO. An interrupt is generated when the USB controller clears `USB_EP0_CSR[n]_P.TXPKTRDY`.

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

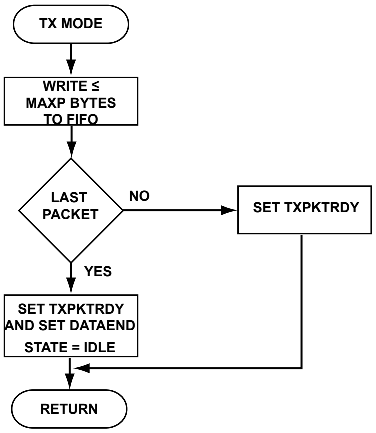

TX MODE
WRITE ≤ MAXP BYTES TO FIFO
LAST PACKET
NO
SET TXPKTRDY
YES
SET TXPKTRDY AND SET DATAEND STATE = IDLE
RETURN

Figure 27-7: Endpoint 0 Tx Mode

Rx Mode

Refer to the ***Endpoint 0 Rx Mode*** figure. In Rx mode, all arriving data must be treated as part of a data phase until the expected amount of data is received. If either a SETUP or an IN token is received while the endpoint is in Rx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs since the core expects only OUT tokens.

Three events can cause the Rx mode to terminate before the expected amount of data is received:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The host sends a packet which contains less than the maximum packet size for endpoint 0.
- The host sends an empty data packet.

The transaction terminates when the firmware receives an interrupt which indicates that new data has arrived (`USB_EP0_CSR[n]_P.RXPKTRDY` bit is set). Until the transaction terminates, firmware must unload the FIFO and clear `USB_EP0_CSR[n]_P.RXPKTRDY` by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.

When the firmware detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

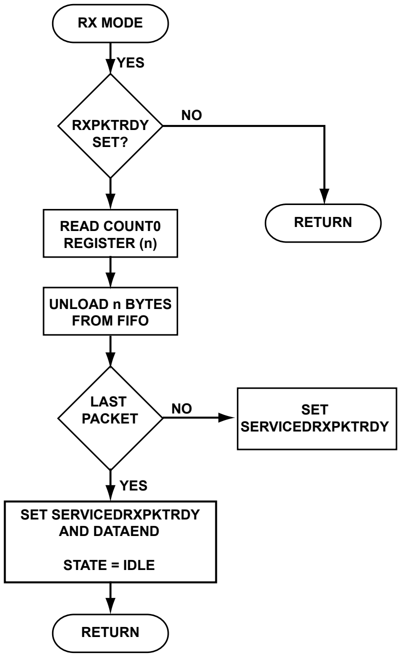

RX MODE
YES
RXPKTRDY SET?
NO
RETURN
READ COUNT0 REGISTER (n)
UNLOAD n BYTES FROM FIFO
LAST PACKET
NO
SET SERVICEDRXPKTRDY
YES
SET SERVICEDRXPKTRDY AND DATAEND
STATE = IDLE
RETURN

Figure 27-8: Endpoint 0 Rx Mode

Peripheral Mode, Bulk IN, Transfer Size Known

For this process, the maximum size of an individual packet (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes, must be known.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the following bits: `USB_EP[n]_TXCSR_P.DMAREQEN` = 1, `USB_EP[n]_TXCSR_P.AUTOSET` = 1, `USB_EP[n]_TXCSR_P.ISO` = 0, `USB_EP[n]_TXCSR_P.FRCDATATGL`= 0.
3. Load the *TxferSize* value into the `USB_DMA[n]_CNT` register.
4. Configure the DMA controller to write the data into the corresponding Tx FIFO address.
5. On each `USB_DMA[n]_CNT` transition, the DMA controller writes a new packet into the FIFO. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is automatically set when each new packet is written.

   ADDITIONAL INFORMATION: Repeat Step 5 for each full packet of the transfer. Even if the final packet is a short packet, the USB controller automatically detects the packet because the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set.

Peripheral Mode, Bulk IN, Transfer Size Unknown

For this process, assume the maximum individual packet size (*MaxPktSize*) in bytes is an even number of bytes.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the following bits: `USB_EP[n]_TXCSR_P.DMAREQEN` = 1, `USB_EP[n]_TXCSR_P.AUTOSET` = 1, `USB_EP[n]_TXCSR_P.ISO` = 0, `USB_EP[n]_TXCSR_P.FRCDATATGL`= 0.

3. Configure the DMA controller to write *MaxPktSize*/2 half words into the corresponding Tx FIFO address on each `USB_DMA[n]_CNT`.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt, that writes a remaining short packet into the Tx FIFO using processor core DMA. Then, set the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit or toggle the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit to send a zero-length packet.
5. On each `USB_DMA[n]_CNT` transition, the DMA controller writes a new packet into the FIFO. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set automatically when each new packet is written.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. The ISR from step 4 manages the final short or zero-length packet.

Peripheral Mode, ISO IN, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the following bits: `USB_EP[n]_TXCSR_P.ISO` = 1.
3. Preload the first two packets into the endpoint Tx FIFO and set the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit.
4. Set up an ISR, sensitive to the `USB_IRQ.SOF` interrupt, which writes a new packet into the Tx FIFO and sets the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit.
5. Set the `USB_IEN.SOF` bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode, ISO IN, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the `USB_EP[n]_TXCSR_P.ISO` bit = 1.
3. Set the `USB_POWER.ISOUPDT` bit = 1 to prevent the initial packet loaded into the FIFO from transmitting on the USB until the next 1ms frame.
4. Load the total number of bytes for the first two packets into the `USB_DMA[n]_CNT` register.
5. Configure the DMA controller to pre-load the two packets into the corresponding Tx FIFO address and set the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit.

6. Set up an ISR, sensitive to the `USB_IRQ.SOF` interrupt, which writes a new packet into the Tx FIFO by configuring the DMA controller to load the packet.
7. Set the `USB_IEN.SOF` bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 7 for each ISO packet.

Peripheral Mode, Bulk OUT, Transfer Size Known

For this process, the maximum individual packet size (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes must be known.

1. Load *MaxPktSize* into `USB_EP[n]_RXMAXP`.
2. Set the following bits: `USB_EP[n]_RXCSR_P.DMAREQEN` = 1, `USB_EP[n]_RXCSR_P.AUTOCLR`= 1, `USB_EP[n]_RXCSR_P.ISO` = 0, `USB_EP[n]_RXCSR_P.CLRDATATGL` = 0, `USB_EP[n]_RXCSR_P.DMAREQMODE` = 0.
3. Configure the DMA controller to read the full *TxferSize*/2 half words from the corresponding Rx FIFO address.
4. On each `USB_DMA[n]_CNT` transition, the DMA controller reads another packet from the FIFO. The USB controller automatically clears the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit when each new packet is read.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. If *TxferSize* is not an exact multiple of *MaxPktSize*, the final `USB_DMA[n]_CNT` transition causes the DMA controller to read out only the short packet that remains.

Peripheral Mode, Bulk OUT, Transfer Size Unknown

For this process, the maximum individual packet size (*MaxPktSize*) in bytes must be known.

1. Load *MaxPktSize* into `USB_EP[n]_RXMAXP`.
2. Set the following bits: `USB_EP[n]_RXCSR_P.DMAREQEN`= 1, `USB_EP[n]_RXCSR_P.AUTOCLR` = 1, `USB_EP[n]_RXCSR_P.ISO` = 0, `USB_EP[n]_RXCSR_P.CLRDATATGL`= 0, `USB_EP[n]_RXCSR_P.DMAREQMODE` = 1.
3. Set the appropriate bit in the `USB_INTRRXE` register.
4. Configure the DMA controller to read *MaxPktSize*/2 half words from the corresponding Rx FIFO address on each `USB_DMA[n]_CNT` transition.
5. Set up an ISR, sensitive to the Rx interrupt, which reads the `USB_EP[n]_RXCNT` register and then transfers `USB_EP[n]_RXCNT` bytes (in half words) from the Rx FIFO to the processor core.

   ADDITIONAL INFORMATION: Depending on the number of bytes in the FIFO, configure the DMA to read the data, or read it with the processor core.

ADDITIONAL INFORMATION: On each `USB_DMA[n]_CNT` transition, the DMA controller reads a packet from the FIFO. The USB controller automatically clears the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit when each new packet is read.

ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer.

6. If a packet is received that is less than *MaxPktSize*, the Rx interrupt goes high, and the ISR from step 5 reads out the remaining short packet.

Peripheral Mode, ISO OUT, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes. Assume that double buffering is enabled.

1. Load the *MaxPktSize* value into the `USB_EP[n]_RXMAXP` register.
2. Set the `USB_EP[n]_RXCSR_P.ISO` bit = 1.
3. Set up an ISR, sensitive to the `USB_IRQ.SOF` interrupt, that reads the `USB_EP[n]_RXCSR_P.FIFOFULL` bit and then reads the `USB_EP0_CNT[n].RXCNT` status register. The ISR removes one or two packets (equal to the `USB_EP0_CNT[n].RXCNT` number of bytes) from the FIFO. It then clears the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit.
4. Set the `USB_IEN.SOF` bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 4 for each ISO packet.

Peripheral Mode, ISO OUT, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes. Assume that double buffering is enabled.

1. Load *MaxPktSize* into the `USB_EP[n]_RXMAXP` register.
2. Set the `USB_EP[n]_RXCSR_P.ISO` bit = 1.
3. Set up an ISR (sensitive to the `USB_IRQ.SOF` interrupt), that reads the `USB_EP[n]_RXCSR_P.FIFOFULL` bit, and then reads the `USB_EP[n]_RXCNT` status register. The ISR configures the DMA controller to remove one or two packets (equal to the `USB_EP[n]_RXCNT` number of bytes) from the FIFO.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt to clear the `USB_EP[n]_RXCSR_P.RXPKTRDY`.
5. Set the `USB_IEN.SOF` bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode Suspend

When no activity has occurred on the USB for 3 ms, the USB controller enters suspend mode. If the suspend interrupt (`USB_IRQ.SUSPEND`) is enabled, an interrupt is generated.

When resume signaling is detected, the USB controller exits suspend mode. If the `USB_IRQ.RESUME` interrupt is enabled, an interrupt is generated. The processor core can also force the USB controller to exit suspend mode by setting the `USB_POWER.RESUME` bit. This event initiates a remote wake-up. When this bit is set, the USB controller exits suspend mode and drives resume signaling onto the bus. The processor core must clear this bit after 10 ms (a maximum of 15 ms) to end resume-signaling.

NOTE: The `USB_IRQ.RESUME` interrupt is not generated when the processor core exits suspend mode. This interrupt is not generated when the software initiates remote wake-up.

Start of Frame (SOF) Packets

When the USB controller operates in peripheral mode, it receives a start of frame packet from the host every millisecond when in full-speed mode, or every 125 microseconds when in high-speed mode.

When the USB controller receives a SOF packet, it writes the 11-bit frame number contained in the packet into the `USB_FRAME` register. An output pulse, lasting one USB clock bit period, is generated. A start of frame interrupt is also generated (if enabled by the `USB_IRQ.SOF` bit).

After the USB controller has started to receive SOF packets, the controller expects one every millisecond (faster in high-speed mode). If the USB controller does not receive an SOF packet after 1.00358 ms (faster in high-speed mode), it is assumed that the packet is lost. A start of frame pulse (together with a `USB_IRQ.SOF` interrupt) is still generated even though the `USB_FRAME` register is not updated. The USB controller continues to generate an SOF pulse every millisecond (faster in high-speed mode). It resynchronizes these pulses to the received SOF packets when it successfully receives these packets again.

Soft Connect/Soft Disconnect

In peripheral mode, the USB controller sets or clears the `USB_POWER.SOFTCONN` bit to switch between normal mode and non-driving mode. When `USB_POWER.SOFTCONN`=1, the USB controller is in normal mode and the D+/D– lines of the USB bus are enabled. When the `USB_POWER.SOFTCONN`=0, the PHY is put into non-driving mode and D+ and D– are three-stated. The USB controller appears to have been disconnected from the USB bus.

After system reset, `USB_POWER.SOFTCONN`=0. From that point, the USB controller appears disconnected until the software has set `USB_POWER.SOFTCONN` =1. The application software can then choose when to set the PHY to its normal mode. Systems with a lengthy initialization procedure can use this functionality to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB. Once the `USB_POWER.SOFTCONN` bit is set to 1, the software can also clear this bit to 0 to simulate a disconnect.

Error Handling As a Peripheral

The host can abort a control transfer due to a protocol error on the USB. The function controller software can also abort the transfer (for example, because it cannot process the command).

The USB controller automatically detects protocol errors and sends a stall packet to the host under the following conditions.

1. The host sends more data during the OUT data phase of a write request than specified in the command. This condition is detected when the host sends an OUT token after the `USB_EP0_CSR[n]_P.DATAEND` bit is set.
2. The host requests more data during the IN data phase of a read request than specified in the command. This condition is detected when the host sends an IN token after the `USB_EP0_CSR[n]_P.DATAEND` bit is set.
3. The host sends more than *MaxPktSize* data bytes in an OUT data packet.
4. The host sends a non-zero length DATA1 packet during the status phase of a read request.

When the USB controller has sent the stall packet, it sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit and generates an interrupt. When the software receives an endpoint 0 interrupt with the `USB_EP0_CSR[n]_P.SENTSTALL` bit set, it aborts the current transfer, clears the `USB_EP0_CSR[n]_P.SENTSTALL` bit, and returns to the IDLE state.

If the host enters the status phase before all the data for the request transfers, or sends a new SETUP packet before completing the current transfer, then it prematurely ends the transfer. The `USB_EP0_CSR[n]_P.SETUPEND` bit is set and an endpoint 0 interrupt is generated. When the software receives an endpoint 0 interrupt with the `USB_EP0_CSR[n]_P.SETUPEND` bit set, it aborts the current transfer, sets the `USB_EP0_CSR[n]_P.SSETUPEND` bit, and returns to the IDLE state. If the `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set, it indicates that the host has sent another SETUP packet and the software must then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it must set the `USB_EP0_CSR[n]_P.SENTSTALL` bit. The USB controller then sends a stall packet to the host, sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit, and generates an endpoint 0 interrupt.

Stalls Issued to Control Transfers

In peripheral mode, the USB controller automatically issues a stall handshake to a control transfer under the following conditions:

1. The host sends more data during an OUT data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an OUT token (instead of an IN token) after the processor core unloads the last OUT packet and sets the `USB_EP0_CSR[n]_P.DATAEND` bit.
2. The host requests more data during an IN data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an IN token (instead of an OUT token) after the processor core clears `USB_EP[n]_TXCSR_P.TXPKTRDY` and sets `USB_EP0_CSR[n]_P.DATAEND`. The processor sets `USB_EP0_CSR[n]_P.DATAEND` in response to the host-issued ACK for the last packet.
3. The host sends more than *MaxPktSize* data with an OUT data token.

4. The host sends the wrong PID (packet identifier) for the OUT status phase of a control transfer.
5. The host sends more than a zero length data packet for the OUT status phase.

Zero Length OUT Data Packets in Control Transfers

The USB controller uses a zero-length OUT data packet to indicate the end of a control transfer. In normal operation, such packets must only be received after the entire length of the device request transfers (for example, after the processor core has set the `USB_EP0_CSR[n]_P.DATAEND` bit). If the host sends a zero-length OUT data packet before the entire length of device request transfers, this packet signals the premature end of the transfer. In this case, the USB controller automatically flushes any IN token the processor core has loaded for the data phase from the FIFO and sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.

Host Mode

USB OTG interface operations in host mode differ from peripheral mode in a number of ways. The following sections describe host mode operations.

Transaction Scheduling

When operating as a host, the USB controller maintains a frame counter.

If the target function is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame or micro-frame.

If the target function is a low-speed device, a K state is transmitted on the bus to act as a *keep-alive*. It stops the low-speed device from going into suspend mode.

After the SOF packet is transmitted, the USB controller cycles through all the endpoints looking for active transactions. An active transaction is defined as an Rx endpoint for which the `USB_EP[n]_RXCSR_H.REQPKT` bit is set or a Tx endpoint for which the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set.

An active isochronous or interrupt transaction only starts if:

- It is found on the first transaction scheduler cycle of a frame.
- The interval counter for that endpoint has counted down to zero.

This functionality ensures that only one interrupt or isochronous transaction occurs per endpoint per *n* frames or micro frames (or, up to three, if high-bandwidth support is selected). *n* is the interval set in the `USB_EP[n]_TXINTERVAL` or `USB_EP[n]_RXINTERVAL` register for that endpoint.

An active bulk transaction starts immediately, provided there is sufficient time left in the frame to complete the transaction before the next SOF packet is due. If the transaction must be retried, then it is not retried until the transaction scheduler has checked all the other endpoints for active transactions first. (For example, the transaction is retried because a NAK was received or the target function did not respond.) This check ensures that an endpoint that is sending many NAKs does not block other transactions on the bus. The USB controller permits specifying a limit (`USB_EP[n]_TXINTERVAL` or `USB_EP[n]_RXINTERVAL` registers) to the length of time in which NAKs can be received from a particular target before the endpoint is timed out.

Endpoint Setup and Data Transfer

When the `HOST_MODE` bit is set to 1, the USB controller operates as a host for point-to-point communications with another USB device. Or, when attached to a hub, the USB controller operates as a host for communication with a range of devices in a multi-point set-up.

The USB controller supports high-speed, full-speed, and low-speed USB functions, both for point-to-point communication and for operation through a hub.

Where necessary, the core automatically carries out the necessary transaction translation to allow usage of a low-speed or full-speed device with a USB 2.0 hub.

The USB controller supports control, bulk, isochronous, or interrupt transactions.

Transfers between the subsystem and endpoint FIFOs in host mode are similar to peripheral mode. See the descriptions of processor core-to-FIFO data transfer in Peripheral Mode.

Control Transaction as a Host

Host control transactions are conducted through endpoint 0. The software handles all the standard device requests that are sent or received through endpoint 0 (as described in *Universal Serial Bus Specification*, Revision 2.0, Chapter 9).

For a USB peripheral, there are three categories of standard device requests:

- ***Zero data requests.*** Comprised of a SETUP command followed by an IN status phase. The command includes all the information.
- ***Write requests.*** Comprised of a SETUP command, followed by an OUT data phase followed by an IN status phase. Extra data follows the command.
- ***Read requests*** Comprised of a SETUP command, followed by an IN data phase followed by an OUT status phase. The device is required to send data back to the host.

A timeout can be set to limit the length of time during which the USB controller retries a transaction that the target continually NAKs. This limit can be between 2 and 2^{15} frames or micro frames and is set through the `USB_EP0_NAKLIMIT[n]` register.

The following sections describe the steps taken in different phases of a control transaction and the actions of the core when issuing standard device requests.

Set up Phase as a Host

The processor core driving the host device performs the following actions for the SETUP phase of a control transaction.

1. Load the 8 bytes of the required device request command into the endpoint 0 FIFO.
2. Set the `USB_EP0_CSR[n]_H.SETUPPKT` bit and `USB_EP0_CSR[n]_H.TXPKTRDY` bit. These bits must be set together.

The USB controller then sends a SETUP token followed by the 8-byte command to endpoint 0 of the addressed device, retrying as necessary.

3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example, set `USB_INTRTXE.EP0`). The processor core then reads the `USB_EP0_CSR[n]_H` register to establish whether the `USB_EP0_CSR[n]_H.RXSTALL`, `USB_EP0_CSR[n]_H.TOERR`, or the `USB_EP0_CSR[n]_H.NAKTO` bits are set.

   If `USB_EP0_CSR[n]_H.RXSTALL`=1, the target did not accept the command (for example, because the target device does not support it) and issues a stall response.

   If `USB_EP0_CSR[n]_H.TOERR`=1, the USB controller tried to send the SETUP packet and the following data packet three times without getting a response.

   If `USB_EP0_CSR[n]_H.NAKTO`=1, the USB controller received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the `USB_EP0_NAKLIMIT[n]` register. Direct the USB controller to either clear the `USB_EP0_CSR[n]_H.NAKTO` bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the `USB_EP0_CSR[n]_H.NAKTO` bit.

4. If none of `USB_EP0_CSR[n]_H.RXSTALL`, `USB_EP0_CSR[n]_H.TOERR` or `USB_EP0_CSR[n]_H.NAKTO` bits are set, the SETUP phase is correctly acknowledged. The processor core can proceed to the following IN data phase, OUT data phase or IN status phase specified for the particular standard device request.

IN Data Phase as a Host

The processor core driving the host device performs the following actions for the IN data phase of a control transaction.

1. Set the `USB_EP0_CSR[n]_H.REQPKT` bit.

2. Wait while the USB controller sends the IN token and then receives the required data back.

3. When the USB controller generates the endpoint 0 interrupt (for example, by setting the `USB_INTRTXE.EP0` bit), read the `USB_EP0_CSR[n]_H` register. Determine whether the `USB_EP0_CSR[n]_H.RXSTALL` bit, the `USB_EP0_CSR[n]_H.TOERR` bit, the `USB_EP0_CSR[n]_H.NAKTO` bit, or the `USB_EP0_CSR[n]_H.RXPKTRDY` bit is set.

   If `USB_EP0_CSR[n]_H.RXSTALL`=1, the target has issued a stall response.

   If `USB_EP0_CSR[n]_H.TOERR`=1, the USB controller has tried to send the required IN token three times without getting a response.

   If `USB_EP0_CSR[n]_H.NAKTO`=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the `USB_EP0_NAKLIMIT[n]` register. Direct the USB controller to either clear the `USB_EP0_CSR[n]_H.NAKTO` bit to continue trying this transaction (until it times out again) or clear `USB_EP0_CSR[n]_H.REQPKT` before clearing the `USB_EP0_CSR[n]_H.NAKTO` bit to abort the transaction.

4. If the `USB_EP0_CSR[n]_H.RXPKTRDY` bit is set, the processor core reads the data from the endpoint 0 FIFO, then clears `USB_EP0_CSR[n]_H.RXPKTRDY`.
5. If further data is expected, the processor core must repeat the previous steps.

   When all the data is successfully received, the processor core can proceed to the OUT status phase of the control transaction.

OUT Data as a Host (Control)

The processor core driving the host device performs the following actions for the OUT data phase of a control transaction.

1. Load the data to be sent into the endpoint 0 FIFO.
2. Set the `USB_EP0_CSR[n]_H.TXPKTRDY` bit.

   The USB controller sends an OUT token followed by the data from the FIFO to endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example by setting the `USB_INTRTX.EP0` bit). The processor core can then read the `USB_EP0_CSR[n]_H` to establish whether the `USB_EP0_CSR[n]_H.RXSTALL` bit, the `USB_EP0_CSR[n]_H.TOERR` bit, or the `USB_EP0_CSR[n]_H.NAKTO` bit is set.

   If `USB_EP0_CSR[n]_H.RXSTALL`=1, the target has issued a stall response.

   If `USB_EP0_CSR[n]_H.TOERR`=1 the USB controller has tried to send the OUT token and the following data packet three times without getting a response.

   If `USB_EP0_CSR[n]_H.NAKTO`=1, the USB controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the `USB_EP0_NAKLIMIT[n]` register. Direct the USB controller to either clear the `USB_EP0_CSR[n]_H.NAKTO` bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the `USB_EP0_CSR[n]_H.NAKTO` bit.

   If none of the `USB_EP0_CSR[n]_H.RXSTALL`, `USB_EP0_CSR[n]_H.TOERR`, or `USB_EP0_CSR[n]_H.NAKTO` bits are set, the OUT data is correctly acknowledged.
4. If further data must be sent, the processor core must repeat the previous steps.

   When all the data is successfully sent, the processor core proceeds to the IN status phase of the control transaction.

IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase)

The processor core driving the host device performs the following actions for the IN status phase of a control transaction.

1. Set the `USB_EP0_CSR[n]_H.STATUSPKT` and `USB_EP0_CSR[n]_H.REQPKT` bits. These bits must be set together.

2. Wait while the USB controller both sends an IN token and receives a response from the USB peripheral.
3. When the USB controller generates the endpoint 0 interrupt (for example, it sets the USB_INTRTX.EP0 bit), read the USB_EP0_CSR[n]_H register to establish whether the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, USB_EP0_CSR[n]_H.NAKTO, or the USB_EP0_CSR[n]_H.RXPKTRDY bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or clear USB_EP0_CSR[n]_H.REQPKT before clearing the USB_EP0_CSR[n]_H.NAKTO bit to abort the transaction.
4. If the USB_EP0_CSR[n]_H.RXPKTRDY bit is set, the processor core typically clears it.

OUT Status Phase as a Host (Following IN Data Phase)

The processor core driving the host device performs the following actions for the OUT status phase of a control transaction.

1. Set USB_EP0_CSR[n]_H.STATUSPKT and USB_EP0_CSR[n]_H.TXPKTRDY bits. These bits must be set together.
2. Wait while the USB controller both sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt. The processor core then reads the USB_EP0_CSR[n]_H register to discover when the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the STATUS packet and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.

4. If none of the `USB_EP0_CSR[n]_H.RXSTALL`, `USB_EP0_CSR[n]_H.TOERR`, or `USB_EP0_CSR[n]_H.NAKTO` bits are set, the status phase is correctly acknowledged.

Host IN Transactions

When the USB controller operates as a host, IN transactions are handled in the same way as OUT transactions are handled when the USB controller is operating as a peripheral. First, the USB controller sets `USB_EP[n]_RXCSR_H.REQPKT` bit to initiate the transaction. This bit indicates to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target function.

When the packet is received and placed in the Rx FIFO, the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is set. The appropriate Rx endpoint interrupt is generated (if enabled) to signal that the processor can now unload a packet can now from the FIFO. When the processor unloads the packet, `USB_EP[n]_RXCSR_H.RXPKTRDY` is cleared. The USB controller uses the `USB_EP[n]_RXCSR_H.AUTOCLR` bit to clear the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit automatically when the processor unloads a maximum sized packet from the FIFO. There is also an `USB_EP[n]_RXCSR_H.AUTOREQ` bit that automatically sets the `USB_EP[n]_RXCSR_H.REQPKT` bit when the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is cleared. The USB controller can use the `USB_EP[n]_RXCSR_H.AUTOCLR` and `USB_EP[n]_RXCSR_H.AUTOREQ` bits with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to a bulk or interrupt IN token with a NAK, the USB controller retries the transaction until the NAK limit set in the `USB_EP0_NAKLIMIT[n]` register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the `USB_EP[n]_RXCSR_H.RXSTALL` bit to interrupt the processor core. If the target function does not respond to the IN token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller clears the `USB_EP[n]_RXCSR_H.REQPKT` bit and interrupts the processor core with the `DATAERROR_R` bit in `USB_RXCSR` set.

Host OUT Transactions

When the USB controller operates as a host, OUT transactions are handled like IN transactions are handled when the USB controller operates as a peripheral.

The `USB_EP[n]_TXCSR_H.TXPKTRDY` bit must be set as the processor loads each packet into the Tx FIFO. The USB controller uses the `USB_EP[n]_TXCSR_H.AUTOSET` bit to cause the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit to be automatically set when the processor loads a maximum sized packet into the FIFO. The USB controller can use the `USB_EP[n]_TXCSR_H.AUTOSET` bit with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to the OUT token with a NAK, the USB controller retries the transaction until the NAK limit set in the `USB_EP0_NAKLIMIT[n]` register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the `USB_EP[n]_TXCSR_H.RXSTALL` bit to interrupt the processor core. If the target function does not respond to the OUT token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller flushes the FIFO and sets the `USB_EP[n]_TXCSR_H.TXTOERR` bit to interrupt the processor core.

Multi-Point Support

The following sections describe the multi-point support of the USB controller.

- Allocating Devices to Endpoints
- Multi-Point Operation
- Multi-Point Bandwidth Considerations

Allocating Devices to Endpoints

The separate functions of the connected devices are allocated to the endpoints within the USB controller through a group of three registers. The registers are associated with each implemented Rx or Tx endpoint (including endpoint 0).

The registers are:

- `USB_MP[n]_TXFUNCADDR`/ `USB_MP[n]_RXFUNCADDR`
- `USB_MP[n]_TXHUBADDR`/ `USB_MP[n]_RXHUBADDR`
- `USB_MP[n]_TXHUBPORT`/ `USB_MP[n]_RXHUBPORT`

The location of these registers depends on which of the endpoints is being addressed.

Record the address of the target function that is accessed through the selected endpoint in the transmit and receive function address registers. Record this information separately for each Tx and Rx endpoint used. In particular, set both `USB_MP[n]_TXFUNCADDR` and `USB_MP[n]_RXFUNCADDR` for endpoint 0.

The USB controller uses the transmit and receive hub address and hub port registers when a full-speed or low-speed device is connected to it through a high-speed USB 2.0 hub. The hub carries out the required transaction translation between high-speed transmission and low-speed or full-speed transmission. In this situation, the `USB_MP[n]_TXHUBADDR`/ `USB_MP[n]_RXHUBADDR` and `USB_MP[n]_TXHUBPORT`/ `USB_MP[n]_RXHUBPORT` registers must record the address of the hub that carries out the transaction translation. It must also record the address of the port of that hub through which the associated Tx or Rx endpoint must access the device.

If endpoint 0 is connected to a hub, then set both the Tx and the Rx versions of these registers for this endpoint. The USB controller also uses hub address registers to record whether the hub offers multiple transaction translators or just a single transaction translator. This configuration has a significant effect on the overall bandwidth that can be achieved.

In addition to recording the address of the target function, record the endpoint number and operating speed of the target device and the type of transaction that is executed. For a Tx endpoint, set this information in the `USB_EP[n]_TXTYPE` register when the index register is set to select the required endpoint. For an Rx endpoint, set this information in the `USB_EP[n]_RXTYPE` register when the index register is set to select the required endpoint. In both cases, record the endpoint number in bits [3:0], select the transaction type through bits [5:4], and select the operating speed through bits [7:6].

Set only the speed for endpoint 0 because endpoint 0 only has the facilities to handle control transactions and therefore is always associated with a device endpoint 0. Use bits [7:6] of the Type 0 register to set the speed. The register is located at address 0x1A when the index register is set to 0.

Multi-Point Operation

After allocating functions to endpoints and recording the operating speed of the target device, multi-point operations can be configured. Most operations in a multi-point set-up are the same as for the equivalent actions where the core is attached to a single other device.

However, more steps are required when:

- The option of dynamically switching the allocation of functions to endpoints is taken (for example, to allow the support of a wider range of devices).
- The control packets normally associated with endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, the program must monitor the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. This knowledge allows the program to select the correct data toggle state when switching occurs between one device and the other. (This action is the programs responsibility. The core cannot determine what data toggle state is expected when a function switches in and out of use.)

The data toggle state can be switched from its current state by writing to the appropriate `USB_EP[n]_TXCSR_H` or `USB_EP[n]_RXCSR_H` register. This activity sets the data toggle write enable and data toggle bits that are included in the registers when the core is in host mode.

Data toggle write enable and data toggle bits are also included in the `USB_EP0_CSR[n]_H` register. However, control operations carried out through endpoint 0 of the core normally leave the data toggle in the expected state.

Where control packets are handled through an endpoint other than endpoint 0, programs must prompt for each setup token to be sent. Programs must set the `USB_EP[n]_TXCSR_H.SETUPPKT` bit when the core operates in host mode, along with the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. If the `USB_EP[n]_TXCSR_H.SETUPPKT` bit is not set, an OUT token is sent.

Use endpoint 0 of the USB controller to handle control packets for all of the devices attached to the controller, and to switch the allocation of this endpoint, as appropriate. Sending the correct token is ensured, as is ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described, but note the following:

- The control function must be allocated to an Rx/Tx endpoint pair (with the same endpoint number).
- The chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed. The size is a minimum of 8 bytes for low-speed or full-speed transactions but 64 bytes for high-speed transactions.

Multi-Point Bandwidth Considerations

The available bandwidth determines the ability of a multi-point system to cope with isochronous transactions.

Once an endpoint is set up, hardware handles all scheduling. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), software must determine that there is sufficient bandwidth available.

The bandwidth required for different transactions can be determined using algorithms similar to the ones used with PC-based hosts (detailed in Section 5.11.3 of the USB 2.0 Specification).

The available bandwidth is greater where the hub used supports multiple transaction translators.

Babble Interrupt

If the bus is still active at the end of a frame, the USB controller assumes that the function it is connected to has malfunctioned. It suspends all transactions, and generates a babble interrupt (`USB_IRQ.RSTBABBLE`). The USB controller does not start a transaction until the bus is inactive for at least the minimum inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame.

To recover from a babble error condition, the processor must take the following actions inside the interrupt service routine.

1. Turn off VBUS. Wait until the VBUS level indicator reads b#01.
2. Turn on VBUS. Wait until the VBUS level indicator reads b#11.
3. Set the `USB_IRQ.SESSREQ` bit.

The VBUS level indicator is the `USB_DEV_CTL.VBUS` bit field.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `USB_VBUS` signal to the external source. This connection enables software to turn VBUS on and off.

VBUS Events

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications must respond.

- VBUS valid (between 4.4 V and 4.75 V)
- Session valid for A device (between 0.8 V and 2.1 V)
- Session end (between 0.2 V and 0.8 V)

The critical thresholds and the processor response depend on whether the device is an A device or a B device and the circumstances of the event. The following sections describe these actions.

Actions as an A Device

VBUS >VBUS Valid with session initiated by USB controller. VBUS level indicator = b#11 and the session bit is set. When VBUS is greater than VBUS valid, the USB controller selects host mode and waits for a device to connect. It then generates a connect interrupt. The processor resets and enumerates the connected B device.

VBUS > Session valid with session initiated by B device. VBUS level indicator = b#10 and the session bit is clear. When VBUS is greater than session valid, the USB controller generates a session request interrupt. The processor sets the session bit. The USB controller either stays in host mode or changes to peripheral mode, depending upon the state of the pull-up resistor on the B device. For more information, refer to the host negotiation protocol of the OTG specification. The state of the host mode bit indicates the selected mode.

VBUS below VBUS Valid while the Session bit remains set. VBUS level indicator b#11 and the session bit is set. This event indicates a problem with the VBUS power level. For example, the battery power could have dropped too low to sustain VBUS valid. Or, the B device could be drawing more current than the A device can provide. In either case, the USB controller automatically terminates the session and generates a VBUS error interrupt.

To recover from this VBUS error condition, the processor must take the following actions inside the VBUS error interrupt handler.

- Turn off VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#01.
- Turn on VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#11.
- Set the `USB_DEV_CTL.SESSION` bit

The `USB_DEV_CTL.VBUS` bit field indicates the VBUS level.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `DrvVBUS` signal to the external source. Then, the software can be used to turn VBUS on and off.

Actions as a B Device

VBUS > Session Valid. VBUS level indicator = b#10 and session bit is clear. This event indicates activity from the A device. The USB controller sets the session bit and disconnects the pull down resistor on the D+ line.

VBUS < Session Valid. While the session bit remains set, VBUS level indicator = b#01 and session bit is set. This event indicates that the A device has lost power (or become disconnected). The USB controller clears the session bit and generates a disconnect interrupt. The processor ends the session.

VBUS < Session End. VBUS level indicator = b#00. This event is the condition under which a B device can initiate a session request. If the session bit is set, then after 2 ms of SE0 on the bus, the USB controller starts SRP by first pulsing the data line, then pulsing the `USB_VBUS` signal.

Host Mode Reset

If the `USB_POWER.RESET` is set while the USB controller is in host mode, the USB controller generates reset signaling on the bus. The processor core must keep this bit set for 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

Host Mode Suspend

The controller has a suspend mode that allows power savings for the processor. The mode operates as follows.

Entry into Suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the `USB_POWER.SUSPEND` bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start. No SOF packets are generated. If the `USB_POWER.SUSPEND` bit is set, the UTMI+ PHY goes into low-power mode when the USB controller goes into suspend mode and stops the clock.

Sending Resume Signaling. When the application requires the USB controller to leave suspend mode, it must clear and then set the `USB_POWER.RESUME` bit, and leave it set for 20 ms. While the `USB_POWER.RESUME` bit is high, the USB controller generates resume signaling on the bus. After 20 ms, the processor core must clear the `USB_POWER.RESUME` bit, at which point the frame counter and transaction scheduler start.

Responding to Remote Wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and the clock restarts. The USB controller then exits suspend mode and automatically sets the `USB_POWER.RESUME` bit to take over generating the resume signaling from the target. If the `USB_IRQ.RESUME` bit=1, software generates an interrupt.

Suspending and Resuming the Controller

With the introduction of link power management, there are two basic methods to suspend and resume the USB controller. The ***Basic LPM transaction*** diagram demonstrates these two methods.

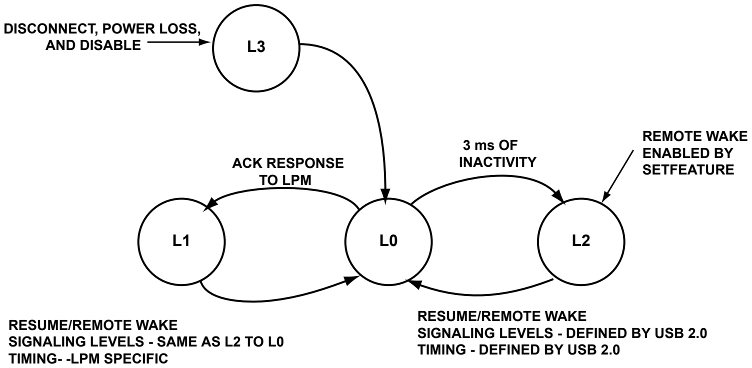

DISCONNECT, POWER LOSS, AND DISABLE
L3
ACK RESPONSE TO LPM
3 ms OF INACTIVITY
REMOTE WAKE ENABLED BY SETFEATURE
L1
L0
L2
RESUME/REMOTE WAKE SIGNALING LEVELS - SAME AS L2 TO L0 TIMING- -LPM SPECIFIC
RESUME/REMOTE WAKE SIGNALING LEVELS - DEFINED BY USB 2.0 TIMING - DEFINED BY USB 2.0

Figure 27-9: Basic LPM Transaction

The procedure that suspends and resumes the USB controller depends on whether the core operates as a device or a host, and the method of suspend desired. The following sections describe these options.

Suspend or Resume by Inactivity on the USB Bus (L0 to L2 State) in Peripheral Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a peripheral, the USB controller monitors activity on the USB and when no activity has occurred for 3 ms, the controller goes into suspend mode. If the `USB_IRQ.SUSPEND` interrupt has been enabled, the USB controller now generates an interrupt. The `USB_IRQ.SUSPEND` output also goes low (if enabled).

The *POWERDWN* signal is also asserted to indicate that the application can stop `USB_CLKIN` to save power. *POWERDWN* then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or resume signaling or reset signaling is detected on the bus.

2. When resume signaling occurs on the bus, the `USB_CLKIN` must be restarted, if necessary. The USB controller then automatically exits suspend mode. If the `USB_IRQ.RESUME` interrupt is enabled, the USB controller generates an interrupt.
3. Initiating a remote wake-up. To initiate a remote wake-up while the controller is in suspend mode, set the `USB_POWER.RESUME` bit=1. (If `USB_CLKIN` has been stopped, it must be restarted before this write can occur.) The software must leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub is driving resume signaling on the USB.

NOTE: The `USB_IRQ.RESUME` interrupt is not generated when the software initiates a remote wake-up.

Suspend or Resume by Inactivity on the USB Bus (L0 To L2 State) in Host Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the `USB_POWER.SUSPEND` bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start and no SOF packets are generated. If the `USB_POWER.SUSEN` bit is set, the UTMI+ PHY goes into low-power mode when the controller goes into suspend mode and stops `USB_CLKIN`.
2. Sending resume signaling. When the application requires the controller to leave suspend mode, it clears the `USB_POWER.SUSPEND` bit, sets the `USB_POWER.RESUME` bit, and leaves it set for 20 ms. While the `USB_POWER.RESUME` bit is high, the controller generates resume signaling on the bus. After 20 ms, the processor core must clear the `USB_POWER.RESUME` bit, at which point the frame counter and transaction scheduler start.
3. Responding to remote wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and restarts `USB_CLKIN`. The controller then exits suspend mode and automatically sets the `USB_POWER.RESUME` bit to 1 to take over generating the resume signaling from the target. If the `USB_IRQ.RESUME` interrupt is enabled, the USB controller generates an interrupt.

Suspend or Resume by an LPM Transaction (L0 To L1 State) in Peripheral Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a peripheral, the controller never initiates an LPM suspend (transition from the L0 state to the L1 state). Rather, the controller only suspends at the request of the host. Configure the `USB_LPM_CTL` register appropriately to enable the LPM feature. The USB controller uses the register field `USB_LPM_CTL.EN` bit to enable and support extended and LPM transactions. The USB controller uses the `USB_LPM_CTL.TX` field to instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the controller responds to the next LPM transaction with an ACK

if all other conditions are met. The ***Response to LPM Transaction*** table summarizes the response of the USB controller to an LPM transaction.

Table 27-6: Response to LPM Transaction

LPMXMT	LPMCNTRL	Data Pending (Resides in Tx FIFOs)	Response to Next LPM Transaction
1’b0, 1’b0 1’b1 1’b1	2’b00, 2’b10 2’b00 2’b10	Do-not-care	Timeout
1’b0, 1’b1	2’b01	Do-not-care	STALL
1’b0	2’b11	Do-not-care	NYET
1’b1	2’b11	Yes	NYET
1’b1	2’b11	No	ACK

For all cases in the table in which the controller responds (no timeout occurs), an LPM interrupt is generated in the `USB_LPM_IRQ` register. The controller responds with an ACK only if there is no data pending in any of the Tx endpoint FIFOs. If there is data pending, the USB controller responds with a NYET.

Once an LPM transaction is successfully received, three events occur:

a. The `USB_LPM_ATTR` register is updated with values received in the LPM transaction. See the “Register Descriptions” section of this chapter for complete information on this register.

b. The controller suspends 9 µs after transmitting the ACK. The host or the controller can drive resume signaling 50 µs after this event. During the 9 µs interval, the host can continue to transmit the LPM transaction. The controller responds with an ACK in this case regardless of the `USB_LPM_CTL.TX` bit value.

c. An interrupt is generated informing software of the response (an ACK in this case). An ACK response is the indication to software that the controller has suspended.

   Since the primary purpose of LPM is to save power, software reads the `USB_LPM_ATTR` register to determine the attributes of the suspend. Software must make a determination based on these attributes whether there are more potential power savings in the system. In making this determination, note that if the host initiates the resume signaling, the controller must respond to packet transmissions within the time specified by `USB_LPM_ATTR.HIRD` + 10 µs.

2. When resume signaling occurs on the bus. When the host resumes the bus, it drives resume signaling for a minimum time specified by the host initiated resume duration bit field (`USB_LPM_ATTR.HIRD`). The controller must be able to respond to traffic within the time HIRD + 10 µs. The controller transitions to a normal operating state automatically and a resume interrupt is generated in the `USB_LPM_IRQ` register.

However for this event to occur, the inputs *CLK* and *XCLK* must be available. To facilitate the resume timing requirement, a negative ACK (NAK) is provided using the `USB_LPM_CTL.NAK` bit. If this bit is set to 1'b1, all endpoints respond to any transaction (other then an LPM) with a NAK. This bit only takes effect after the controller has suspended LPM. Typically, this bit is asserted when the `USB_LPM_CTL.TX` field is also asserted. Using this feature can simplify the resume timing requirement because the controller only needs *XCLK* to respond (with a NAK) to traffic. Software can continue to restore the system to normal operation while the controller responds to all transactions with a NAK. After software completely restores the system, it can then clear the `USB_LPM_CTL.NAK` bit.

3. Initiating remote wake-up. To initiate a remote wake-up while in suspend mode, the controller writes a 1'b1 to the`USB_LPM_CTL.RESUME` bit. This bit is self clearing. Writing a 1'b1 drives resume signaling on the bus for 50 µs. The host responds by driving resume for 60 µs to 990 µs. 10 µs after the host stops driving resume, the controller transitions to its normal operational state and is ready for packet transmission. A resume interrupt is generated in the `USB_LPM_IRQ` register.

Suspend or Resume by an LPM Transaction (L0 to L1 State) in Host Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a host, the controller initiates an LPM suspend (transition from the L0 state to the L1 state) by initiating an LPM transaction as follows.
   a. Software sets up the desired attributes of the suspend in the `USB_LPM_ATTR` register. Enabling remote wake-up and a large HIRD gives the peripheral more opportunity to conserve power.
   b. Enable all LPM interrupts in the `USB_LPM_IEN` register.
   c. Software writes 0x01 to the `USB_LPM_CTL` register to initiate the transaction.
   d. An interrupt is generated to inform software of the response to the LPM transaction. If an ACK was received, then the controller suspends automatically within 8 µs. This event indicates that the controller has suspended.

   If the response from the device has a bit stuff error or a PID error, then an `USB_LPM_IRQ.LPMERR` interrupt is generated. The hardware immediately attempts the LPM transaction two more times. The device does not suspend for 8 µs after the initial LPM so it can respond to either of these subsequent LPM transactions. If an LPM timeout has occurred three times, the `USB_LPM_IRQ.LPMNC` and the `USB_LPM_IRQ.LPMERR` interrupts are set. Now, software is unaware of the device state and must deduce it by other means.

2. Send resume signaling. Software generates resume signaling as follows.
   a. Enable all LPM interrupts in the `USB_LPM_IEN` register.
   b. Software writes to the `USB_LPM_CTL.RESUME` bit which is self-clearing. This operation causes resume signaling on the bus for the time specified in the `USB_LPM_ATTR.HIRD` bit field. Hardware assumes that the last LPM transaction that caused the suspend used this value.
   c. After HIRD + 10 µs, the controller transitions to its normal operational state and is ready for packet transmission and a `USB_LPM_IRQ.LPMRES` interrupt is generated.

   NOTE: Prior to resuming, software must ensure that the system is restored from a low-power state and that the inputs *CLK* and *XCLK* are available.

3. Responding to remote wake-up. If the remote wake-up feature is enabled in the LPM transaction that caused the suspend, then the device can drive resume signaling on the bus. When this event occurs, the device drives resume signaling BUS for 50 µs. The controller immediately begins driving resume signaling on the BUS and continues for 60 µs. 10 µs after completion of the resume signaling, the controller transitions to its normal operating state and is ready for packet transmission. Then, the `USB_LPM_IRQ.LPMRES` interrupt is generated.

USB Event Control

The following sections provide information on the use of interrupts, reset, and the reporting of errors and interface status.

Interrupt Signals

The "Interrupt Table" section at the beginning of this chapter shows the two interrupts generated from the USB controller.

The software generates interrupts from control endpoint zero under the following conditions :

- When a control transaction ends before the end of the data is transferred
- When a data packet is sent or received from the endpoint 0 FIFOs

The USB controller generates interrupts from transmit endpoints (`USB_INTRTX`) under the following conditions:

- A packet is sent from the TX FIFO (host and peripheral mode)
- After three attempts at transmitting a packet, no valid handshake packet is received (host mode)

The software generates interrupts from receive endpoints (`USB_INTRRX`) under the following conditions:

- A packet is received into the RX FIFO (host and peripheral mode)
- A stall handshake is received (host mode)
- After three attempts at receiving a packet, no data packet is received (host mode)

The software generates interrupts from the USB status (`USB_IRQ`) under the following conditions:

- When VBUS drops below the VBUS valid threshold during a session (A device only)
- When SRP signaling is detected (A device only)
- When device disconnect is detected (host mode)
- When a session ends (peripheral mode)
- When a device connection is detected (host mode)
- At start of frame (SOF)
- When reset signaling is detected on USB (peripheral mode)

- When babble is detected (host mode)
- In suspend mode, when resume signaling is detected on USB
- When suspend signaling is detected (peripheral mode)

The software generates interrupts for the following VBUS control requests:

- Drive VBUS greater than 4.4 V (default A device)
- Stop driving VBUS
- Start charging VBUS (peripheral mode)
- Stop charging VBUS
- Start discharging VBUS (peripheral mode)
- Stop discharging VBUS

Interrupt Handling

When interrupted with a USB interrupt, the processor core must read the interrupt status register to determine which endpoints have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 must be serviced first, followed by the other endpoints. The ***USB Interrupt Service Routine*** figure shows a flowchart for the USB interrupt service routine.

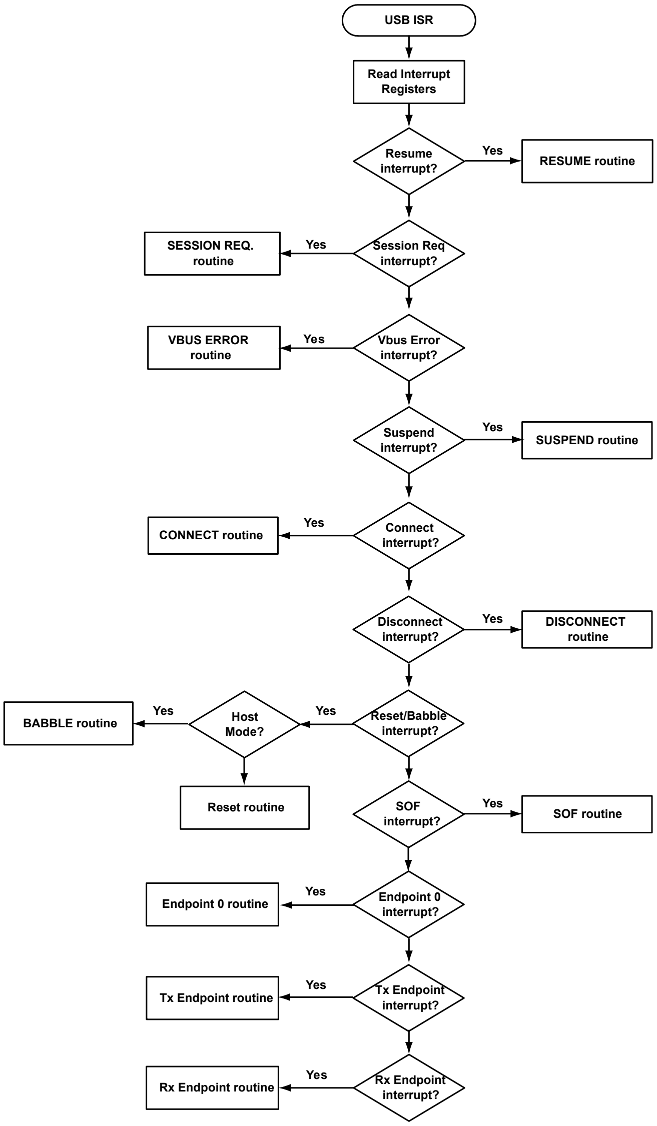

USB ISR
Read Interrupt Registers
Resume interrupt?
Yes
RESUME routine
Session Req interrupt?
Yes
SESSION REQ. routine
Vbus Error interrupt?
Yes
VBUS ERROR routine
Suspend interrupt?
Yes
SUSPEND routine
Connect interrupt?
Yes
CONNECT routine
Disconnect interrupt?
Yes
DISCONNECT routine
Reset/Babble interrupt?
Yes
Host Mode?
Yes
BABBLE routine
Reset routine
SOF interrupt?
Yes
SOF routine
Endpoint 0 interrupt?
Yes
Endpoint 0 routine
Tx Endpoint interrupt?
Yes
Tx Endpoint routine
Rx Endpoint interrupt?
Yes
Rx Endpoint routine

Figure 27-10: USB Interrupt Service Routine

Reset Signals

The USB controller includes an active-high synchronous hardware reset sourced from the processor core. Another source of peripheral reset is through the USB, when USB reset signaling is detected on the I/O lines. Per the USB 2.0 Specification, this state is entered when both the D+ and D- inputs are driven low for 2.5 ms or more. (The USB host typically holds the reset for greater than 10 ms.)

Reset in Peripheral Mode

When the USB controller detects a reset, it performs the following actions:

- Sets the `USB_FADDR` register to zero
- Sets the `USB_INDEX` register to zero

- Flushes all endpoint FIFOs
- Clears all control and status registers
- Enables all interrupts
- Generates a reset interrupt

The USB controller reset does not affect the `USB_IRQ` and `USB_VBUS_CTL` registers. These registers are only reset (along with the ones listed) during a system reset.

If the `USB_POWER.HSEN` bit was set, the USB controller also tries to negotiate for high-speed operation. The `USB_POWER.HSMODE` bit indicates whether high-speed operation is selected.

When the application software receives a reset interrupt, it closes any open pipes and waits for bus enumeration to begin.

USB Reset in Host Mode

If the `USB_POWER.RESET` bit =1 while the USB controller is in host mode, the controller generates reset signaling on the bus.

If the `USB_POWER.HSEN` bit =1, the controller also tries to negotiate for high-speed operation.

The processor core must keep the `USB_POWER.RESET` bit set for at least 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

The USB controller uses the `USB_POWER.HSMODE` bit to select high-speed operation.

USB Programming Model

The following sections describe the USB OTG programming model.

Peripheral Mode Flow Charts

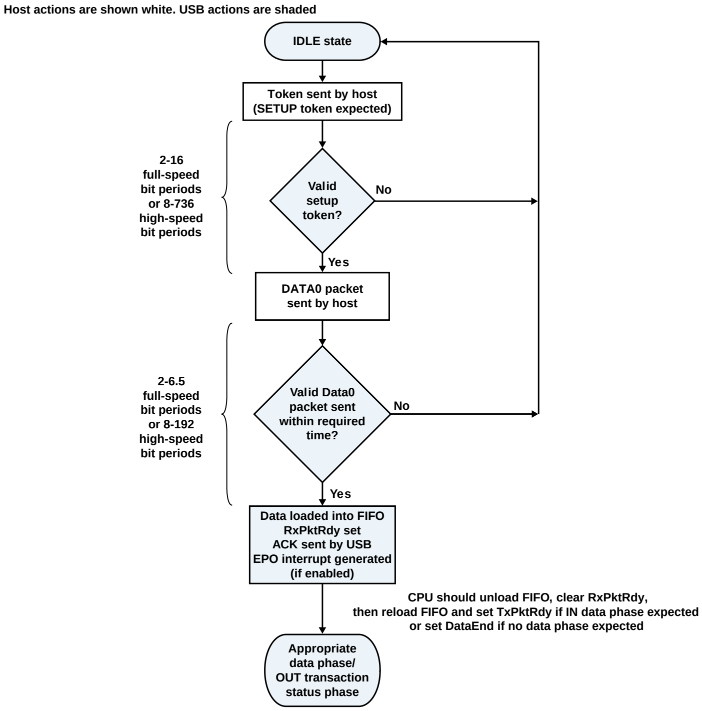

Host actions are shown white. USB actions are shaded
IDLE state
Token sent by host (SETUP token expected)
2-16 full-speed bit periods or 8-736 high-speed bit periods
Valid setup token?
No
Yes
DATA0 packet sent by host
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Valid Data0 packet sent within required time?
No
Yes
Data loaded into FIFO
RxPktRdy set
ACK sent by USB
EPO interrupt generated (if enabled)
CPU should unload FIFO, clear RxPktRdy, then reload FIFO and set TxPktRdy if IN data phase expected or set DataEnd if no data phase expected
Appropriate data phase/ OUT transaction status phase

Figure 27-11: USB Control SETUP Phase

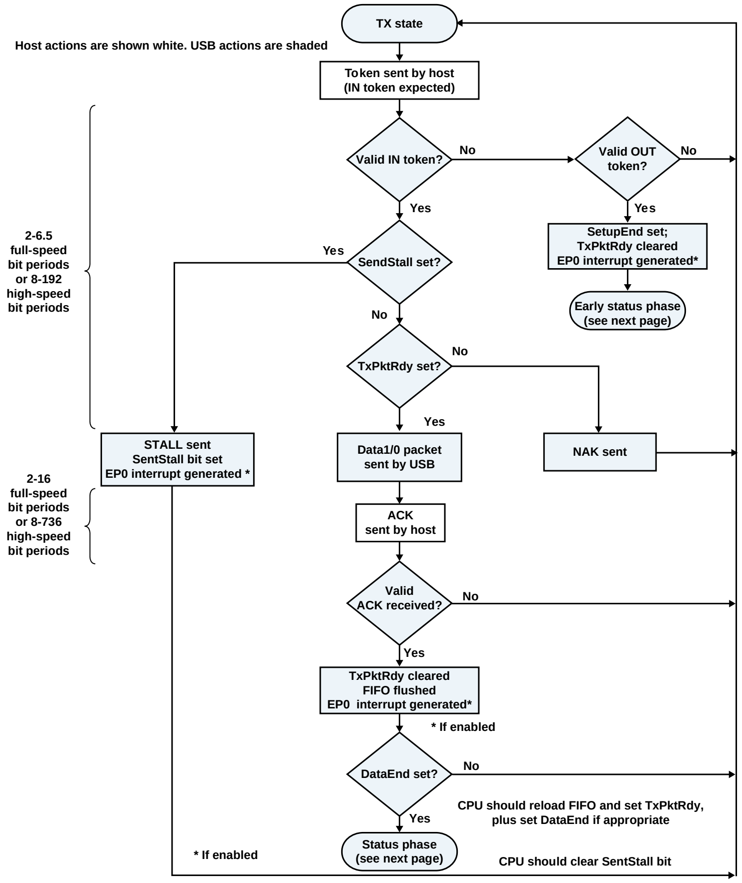

TX state
Host actions are shown white. USB actions are shaded
Token sent by host (IN token expected)
Valid IN token?
No
Valid OUT token?
No
Yes
Yes
SetupEnd set; TxPktRdy cleared EP0 interrupt generated*
Early status phase (see next page)
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Yes
SendStall set?
No
TxPktRdy set?
No
Yes
STALL sent SentStall bit set EP0 interrupt generated *
Data1/0 packet sent by USB
NAK sent
2-16 full-speed bit periods or 8-736 high-speed bit periods
ACK sent by host
Valid ACK received?
No
Yes
TxPktRdy cleared FIFO flushed EP0 interrupt generated*
* If enabled
DataEnd set?
No
CPU should reload FIFO and set TxPktRdy, plus set DataEnd if appropriate
Yes
Status phase (see next page)
* If enabled
CPU should clear SentStall bit

Figure 27-12: Control In Data Phase

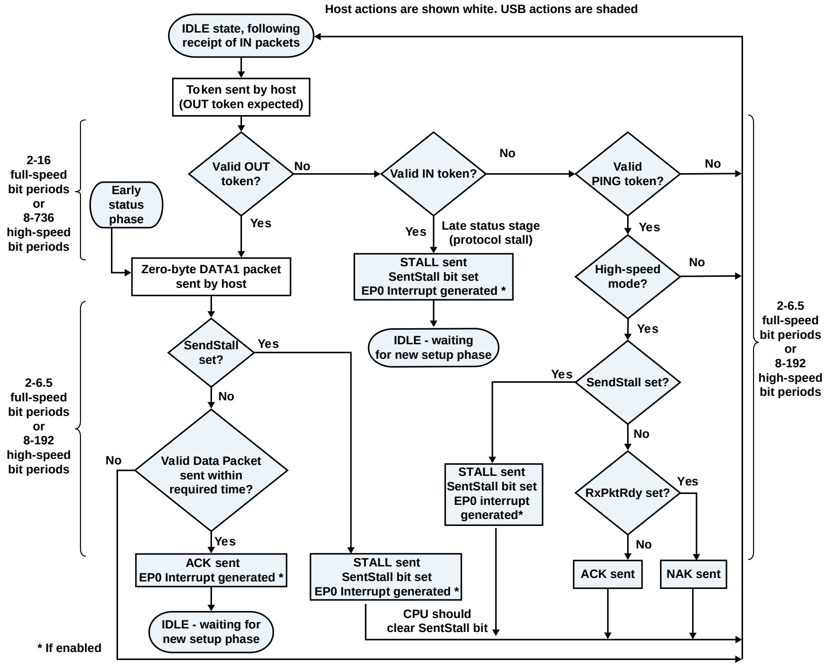

Host actions are shown white. USB actions are shaded
IDLE state, following receipt of IN packets
Token sent by host (OUT token expected)
2-16 full-speed bit periods or 8-736 high-speed bit periods
Early status phase
Valid OUT token?
No
Valid IN token?
No
Valid PING token?
No
Yes
Yes
Late status stage (protocol stall)
Yes
Zero-byte DATA1 packet sent by host
STALL sent SentStall bit set EP0 Interrupt generated *
High-speed mode?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
SendStall set?
Yes
IDLE - waiting for new setup phase
Yes
SendStall set?
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
No
No
No
Valid Data Packet sent within required time?
STALL sent SentStall bit set EP0 interrupt generated*
RxPktRdy set?
Yes
Yes
No
ACK sent EP0 Interrupt generated *
STALL sent SentStall bit set EP0 Interrupt generated *
ACK sent
NAK sent
IDLE - waiting for new setup phase
CPU should clear SentStall bit
* If enabled

Figure 27-13: Control In Status Phase

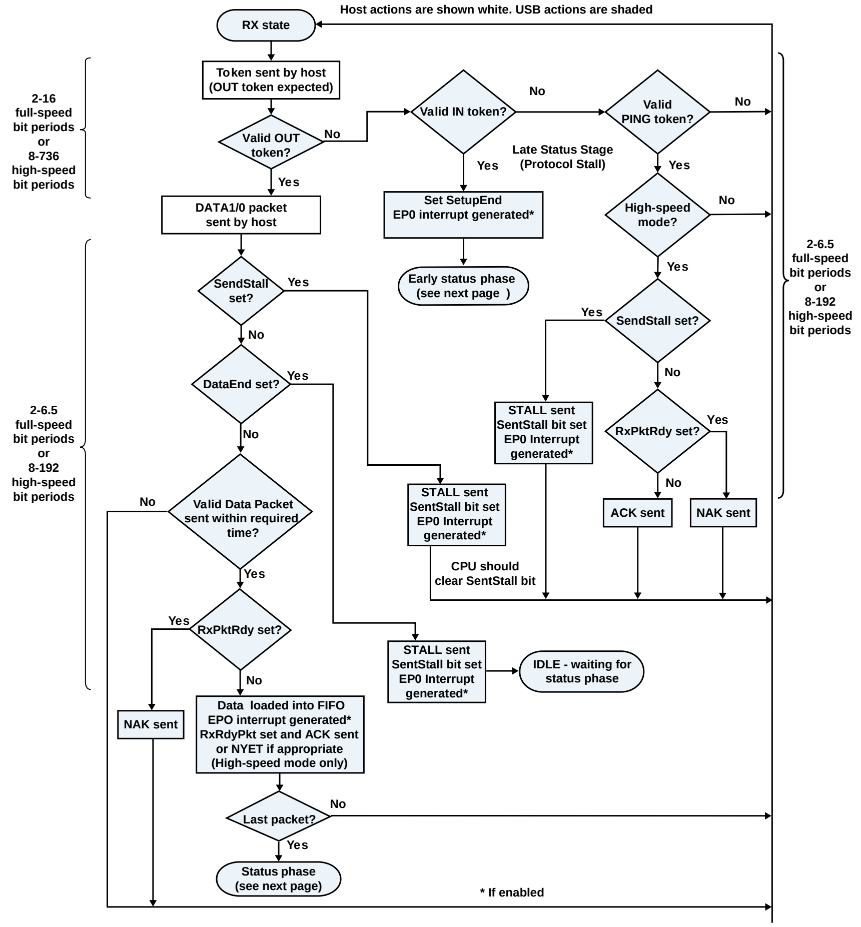

Host actions are shown white. USB actions are shaded
RX state
Token sent by host (OUT token expected)
Valid OUT token?
No
Yes
Valid IN token?
No
Yes
Valid PING token?
No
Yes
Late Status Stage (Protocol Stall)
2-16 full-speed bit periods or 8-736 high-speed bit periods
DATA1/0 packet sent by host
Set SetupEnd EP0 interrupt generated*
Early status phase (see next page)
High-speed mode?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
SendStall set?
Yes
No
SendStall set?
Yes
No
DataEnd set?
Yes
No
STALL sent SentStall bit set EP0 Interrupt generated*
RxPktRdy set?
Yes
No
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Valid Data Packet sent within required time?
No
Yes
STALL sent SentStall bit set EP0 Interrupt generated*
ACK sent
NAK sent
CPU should clear SentStall bit
RxPktRdy set?
Yes
No
STALL sent SentStall bit set EP0 Interrupt generated*
IDLE - waiting for status phase
NAK sent
Data loaded into FIFO EPO interrupt generated* RxRdyPkt set and ACK sent or NYET if appropriate (High-speed mode only)
Last packet?
No
Yes
Status phase (see next page)
* If enabled

Figure 27-14: Control Out Data Phase

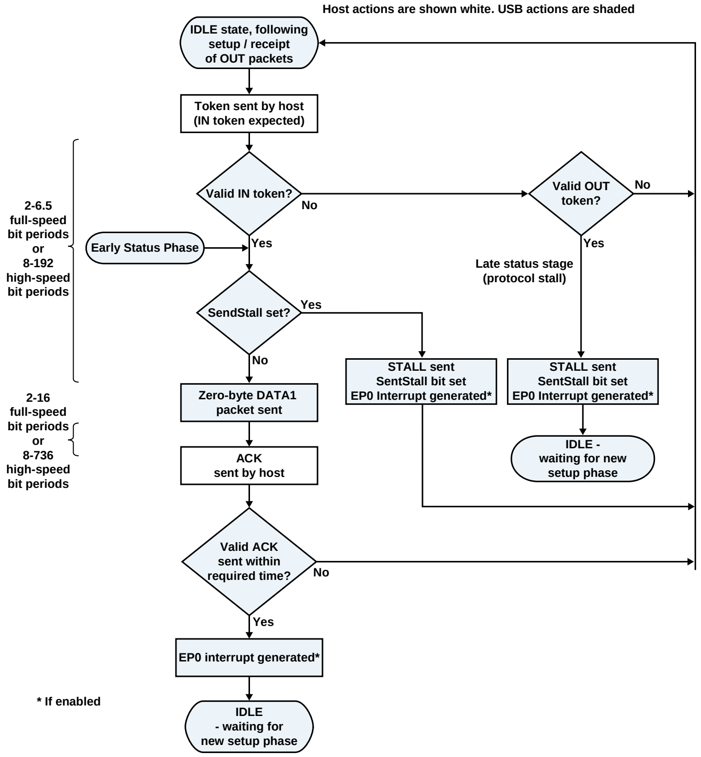

Host actions are shown white. USB actions are shaded
IDLE state, following setup / receipt of OUT packets
Token sent by host (IN token expected)
Valid IN token?
No
Valid OUT token?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Early Status Phase
Yes
Late status stage (protocol stall)
SendStall set?
Yes
No
STALL sent SentStall bit set EP0 Interrupt generated*
STALL sent SentStall bit set EP0 Interrupt generated*
Zero-byte DATA1 packet sent
2-16 full-speed bit periods or 8-736 high-speed bit periods
IDLE - waiting for new setup phase
ACK sent by host
Valid ACK sent within required time?
No
Yes
EP0 interrupt generated*
* If enabled
IDLE - waiting for new setup phase

Figure 27-15: Control Out Status Phase

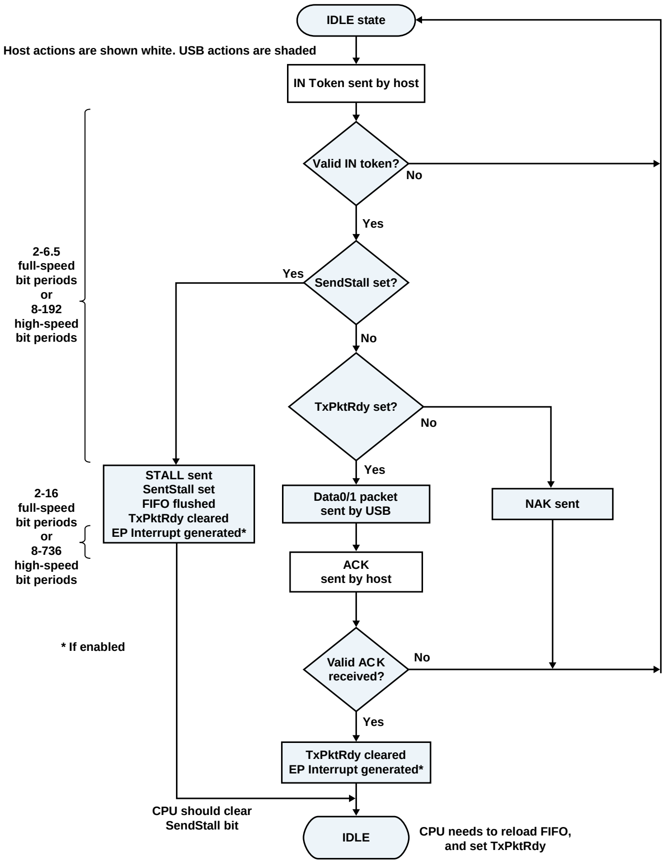

IDLE state
Host actions are shown white. USB actions are shaded
IN Token sent by host
Valid IN token?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Yes
SendStall set?
No
TxPktRdy set?
No
Yes
STALL sent
SentStall set
FIFO flushed
TxPktRdy cleared
EP Interrupt generated*
2-16 full-speed bit periods or 8-736 high-speed bit periods
Data0/1 packet sent by USB
NAK sent
ACK sent by host
* If enabled
Valid ACK received?
No
Yes
TxPktRdy cleared
EP Interrupt generated*
CPU should clear SendStall bit
IDLE
CPU needs to reload FIFO, and set TxPktRdy

Figure 27-16: Bulk/Low Bandwidth Interrupt In Transaction

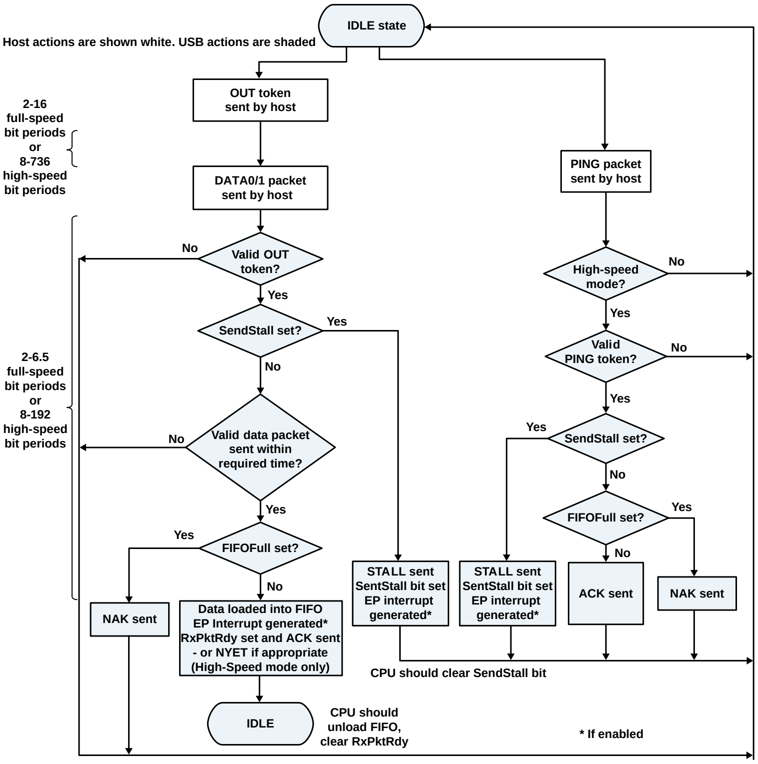

IDLE state
Host actions are shown white. USB actions are shaded
OUT token sent by host
2-16 full-speed bit periods or 8-736 high-speed bit periods
DATA0/1 packet sent by host
PING packet sent by host
No
Valid OUT token?
Yes
High-speed mode?
No
Yes
SendStall set?
Yes
No
Valid PING token?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
No
Valid data packet sent within required time?
Yes
SendStall set?
No
Yes
FIFOFull set?
Yes
Yes
FIFOFull set?
No
No
STALL sent SentStall bit set EP interrupt generated*
STALL sent SentStall bit set EP interrupt generated*
ACK sent
NAK sent
NAK sent
Data loaded into FIFO EP Interrupt generated* RxPktRdy set and ACK sent - or NYET if appropriate (High-Speed mode only)
CPU should clear SendStall bit
IDLE
CPU should unload FIFO, clear RxPktRdy
* If enabled

Figure 27-17: Bulk/Low Bandwidth Interrupt Out Transaction

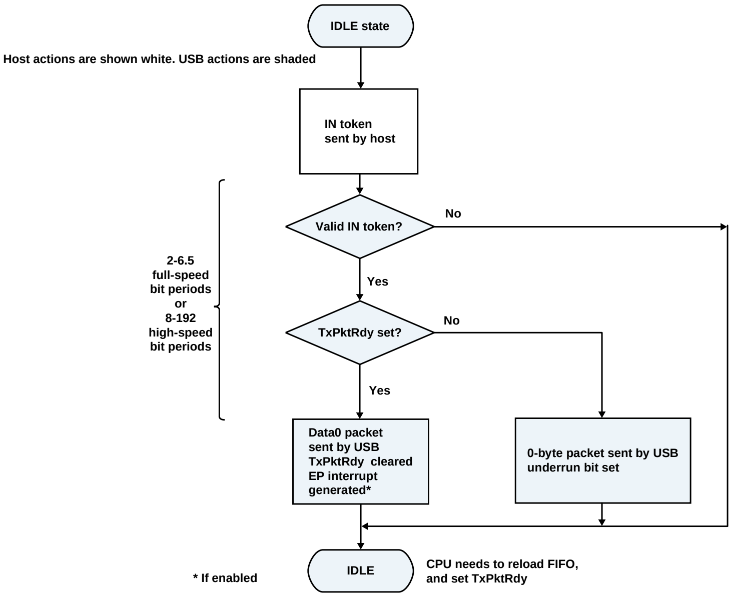

IDLE state
Host actions are shown white. USB actions are shaded
IN token sent by host
Valid IN token?
No
Yes
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
TxPktRdy set?
No
Yes
Data0 packet sent by USB TxPktRdy cleared EP interrupt generated*
0-byte packet sent by USB underrun bit set
* If enabled
IDLE
CPU needs to reload FIFO, and set TxPktRdy

Figure 27-18: Full-speed/Low Bandwidth Isochronous In Transaction

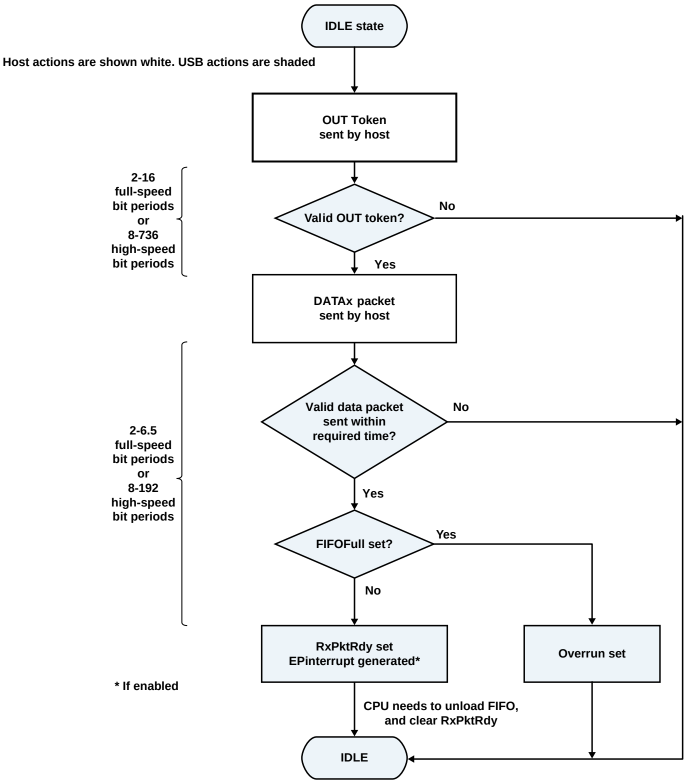

IDLE state
Host actions are shown white. USB actions are shaded
OUT Token sent by host
2-16 full-speed bit periods or 8-736 high-speed bit periods
Valid OUT token?
No
Yes
DATAx packet sent by host
2-6.5 full-speed bit periods or 8-192 high-speed bit periods
Valid data packet sent within required time?
No
Yes
FIFOFull set?
Yes
No
RxPktRdy set EPinterrupt generated*
Overrun set
* If enabled
CPU needs to unload FIFO, and clear RxPktRdy
IDLE

Figure 27-19: Full-speed/Low Bandwidth Isochronous Out Transaction

Host Mode Flow Charts

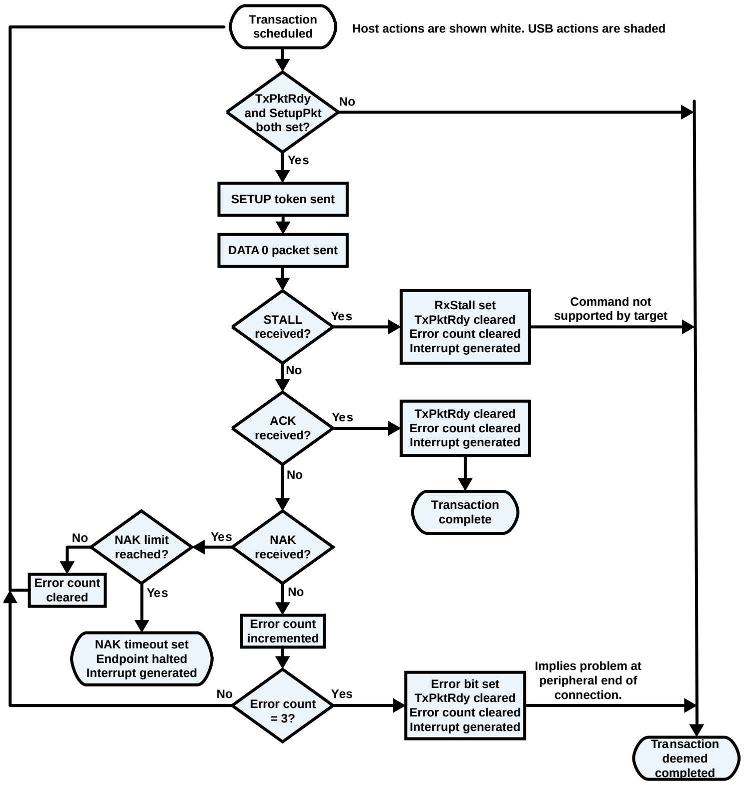

Transaction scheduled
Host actions are shown white. USB actions are shaded
TxPktRdy and SetupPkt both set?
No
Yes
SETUP token sent
DATA 0 packet sent
STALL received?
Yes
RxStall set
TxPktRdy cleared
Error count cleared
Interrupt generated
Command not supported by target
No
ACK received?
Yes
TxPktRdy cleared
Error count cleared
Interrupt generated
Transaction complete
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
TxPktRdy cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-20: USB Control SETUP Phase

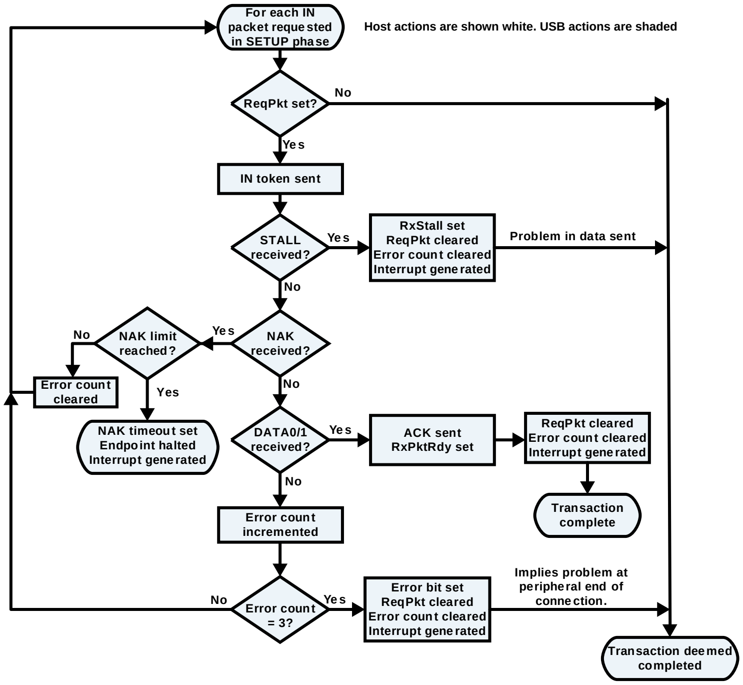

For each IN packet requested in SETUP phase
Host actions are shown white. USB actions are shaded
ReqPkt set?
No
Yes
IN token sent
STALL received?
Yes
RxStall set
ReqPkt cleared
Error count cleared
Interrupt generated
Problem in data sent
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
DATA0/1 received?
Yes
ACK sent
RxPktRdy set
ReqPkt cleared
Error count cleared
Interrupt generated
Transaction complete
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
ReqPkt cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-21: Control In Data Phase

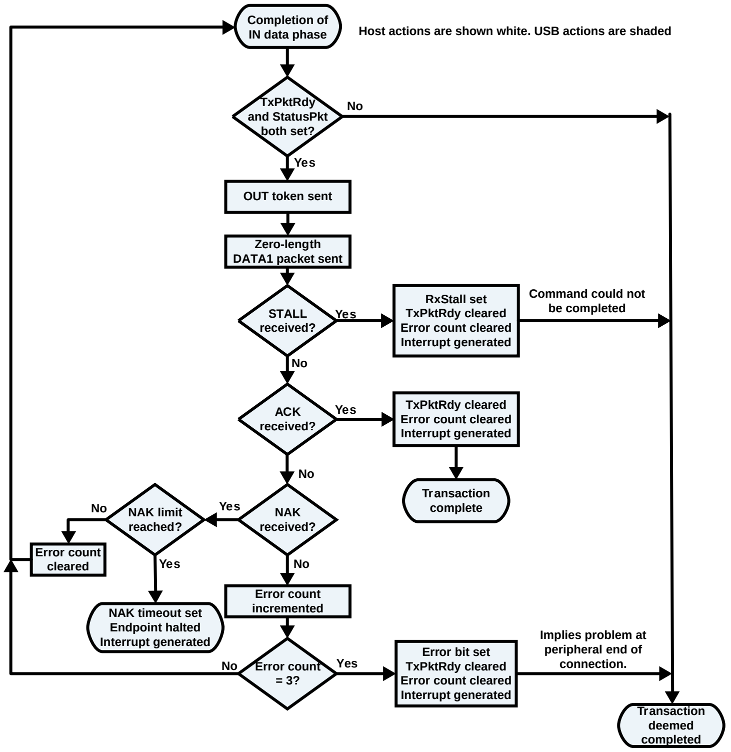

Completion of IN data phase
Host actions are shown white. USB actions are shaded
TxPktRdy and StatusPkt both set?
No
Yes
OUT token sent
Zero-length DATA1 packet sent
STALL received?
Yes
RxStall set
TxPktRdy cleared
Error count cleared
Interrupt generated
Command could not be completed
No
ACK received?
Yes
TxPktRdy cleared
Error count cleared
Interrupt generated
Transaction complete
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
TxPktRdy cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-22: Control In Data Status Phase

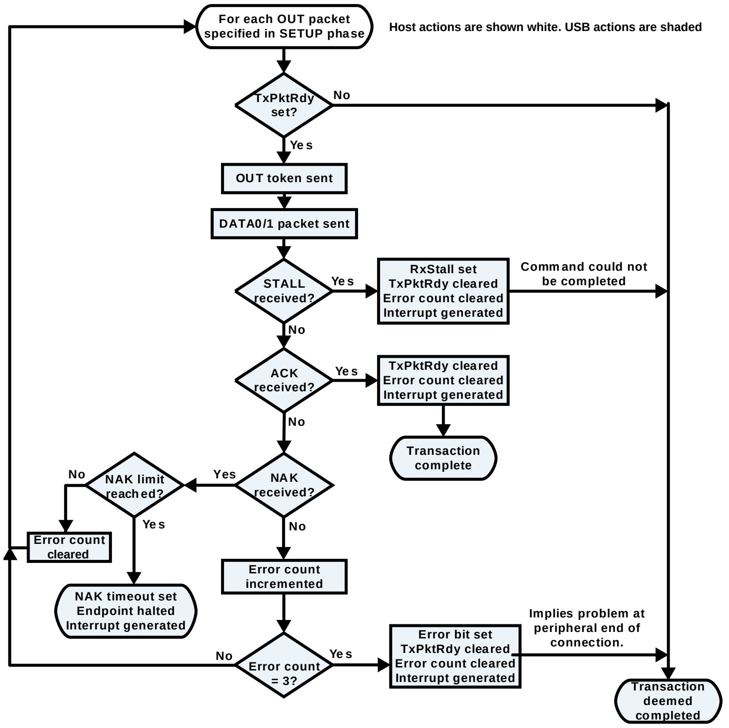

For each OUT packet specified in SETUP phase
Host actions are shown white. USB actions are shaded
TxPktRdy set?
No
Yes
OUT token sent
DATA0/1 packet sent
STALL received?
Yes
RxStall set
TxPktRdy cleared
Error count cleared
Interrupt generated
Command could not be completed
No
ACK received?
Yes
TxPktRdy cleared
Error count cleared
Interrupt generated
Transaction complete
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
TxPktRdy cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-23: Control Out Data Phase

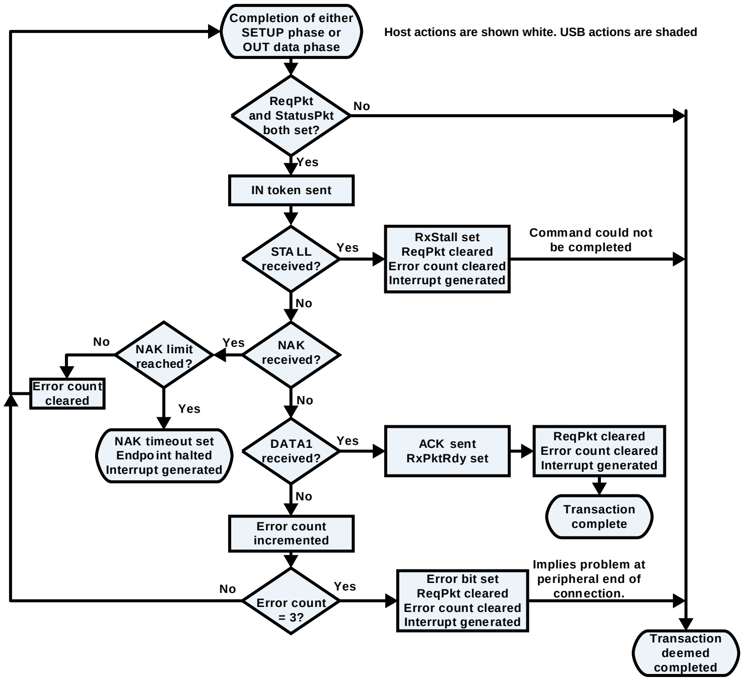

Completion of either SETUP phase or OUT data phase
Host actions are shown white. USB actions are shaded
ReqPkt and StatusPkt both set?
No
Yes
IN token sent
STALL received?
Yes
RxStall set
ReqPkt cleared
Error count cleared
Interrupt generated
Command could not be completed
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
DATA1 received?
Yes
ACK sent
RxPktRdy set
ReqPkt cleared
Error count cleared
Interrupt generated
Transaction complete
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
ReqPkt cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-24: Control Out Data Status Phase

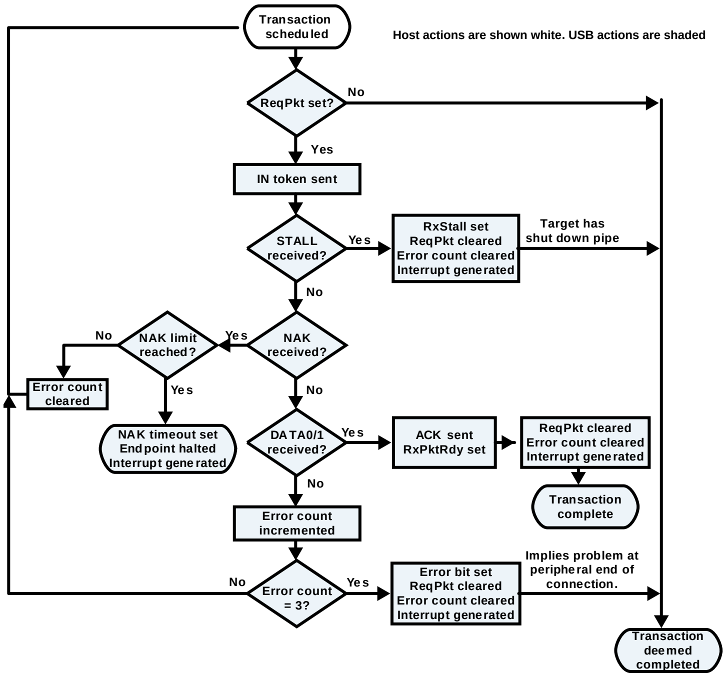

Transaction scheduled
Host actions are shown white. USB actions are shaded
ReqPkt set?
No
Yes
IN token sent
STALL received?
Yes
RxStall set
ReqPkt cleared
Error count cleared
Interrupt generated
Target has shut down pipe
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
DATA0/1 received?
Yes
ACK sent
RxPktRdy set
ReqPkt cleared
Error count cleared
Interrupt generated
Transaction complete
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
ReqPkt cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-25: Bulk/Low Bandwidth Interrupt In Transaction

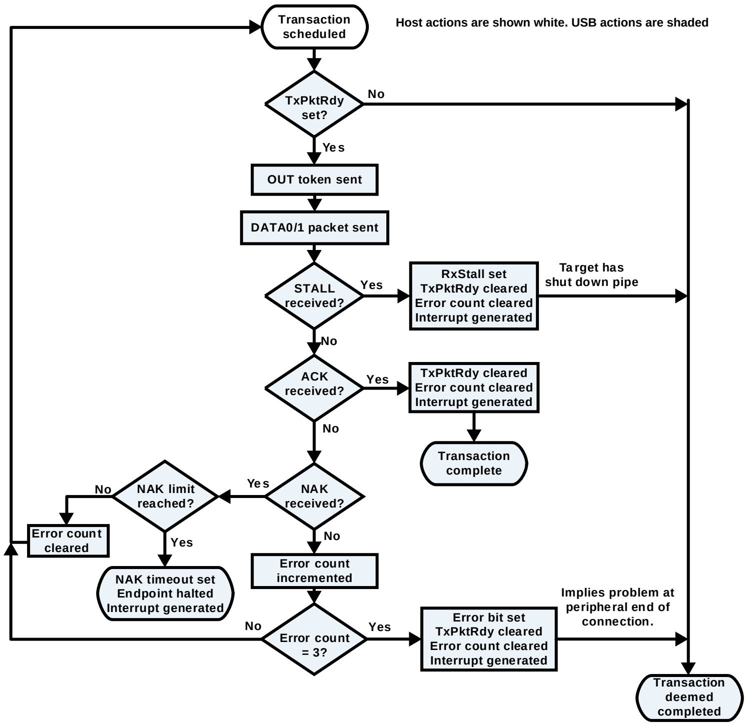

Transaction scheduled
Host actions are shown white. USB actions are shaded
TxPktRdy set?
No
Yes
OUT token sent
DATA0/1 packet sent
STALL received?
Yes
RxStall set
TxPktRdy cleared
Error count cleared
Interrupt generated
Target has shut down pipe
No
ACK received?
Yes
TxPktRdy cleared
Error count cleared
Interrupt generated
Transaction complete
No
NAK received?
Yes
NAK limit reached?
No
Error count cleared
Yes
NAK timeout set
Endpoint halted
Interrupt generated
No
Error count incremented
Error count = 3?
No
Yes
Error bit set
TxPktRdy cleared
Error count cleared
Interrupt generated
Implies problem at peripheral end of connection.
Transaction deemed completed

Figure 27-26: Bulk/Low Bandwidth Interrupt Out Transaction

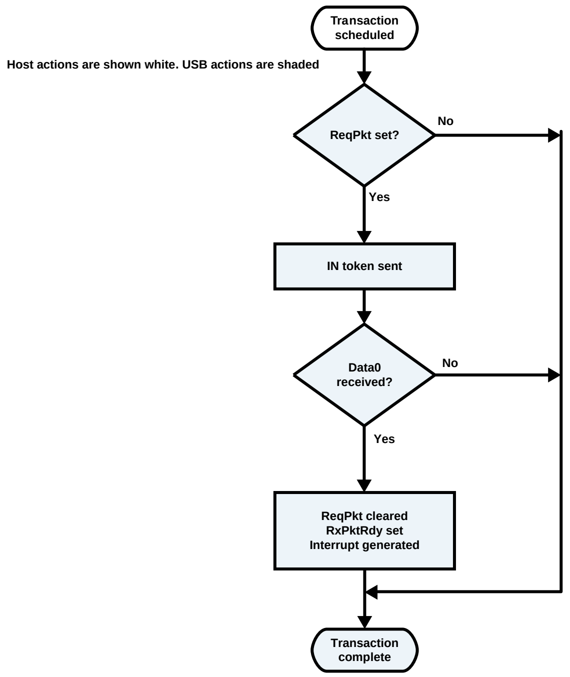

Transaction scheduled
Host actions are shown white. USB actions are shaded
ReqPkt set?
No
Yes
IN token sent
Data0 received?
No
Yes
ReqPkt cleared
RxPktRdy set
Interrupt generated
Transaction complete

Figure 27-27: Full-Speed/Low Bandwidth Isochronous In Transaction

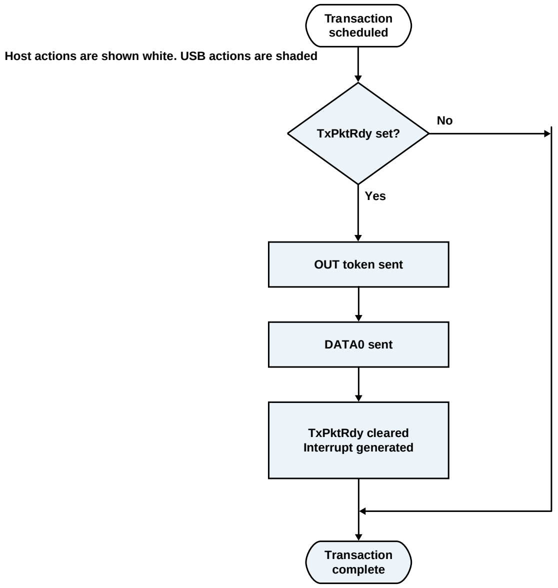

Transaction scheduled
Host actions are shown white. USB actions are shaded
TxPktRdy set?
No
Yes
OUT token sent
DATA0 sent
TxPktRdy cleared
Interrupt generated
Transaction complete

Figure 27-28: Full-Speed/Low Bandwidth Isochronous Out Transaction

DMA Mode Flow Charts

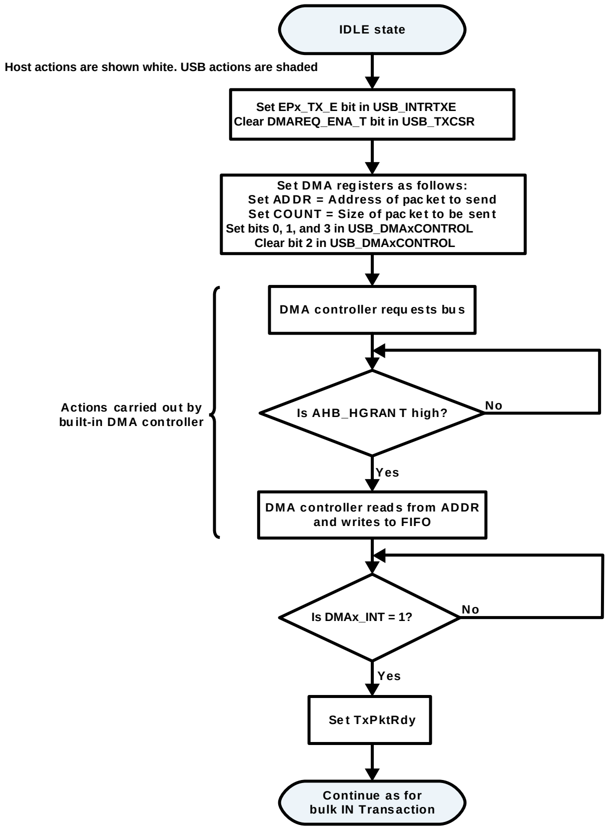

IDLE state
Host actions are shown white. USB actions are shaded
Set EPx_TX_E bit in USB_INTRTXE
Clear DMAREQ_ENA_T bit in USB_TXCSR
Set DMA registers as follows:
Set ADDR = Address of packet to send
Set COUNT = Size of packet to be sent
Set bits 0, 1, and 3 in USB_DMAxCONTROL
Clear bit 2 in USB_DMAxCONTROL
DMA controller requests bus
Actions carried out by built-in DMA controller
Is AHB_HGRANT high?
No
Yes
DMA controller reads from ADDR and writes to FIFO
Is DMAx_INT = 1?
No
Yes
Set TxPktRdy
Continue as for bulk IN Transaction

Figure 27-29: Single Packet Transmit During DMA Operation

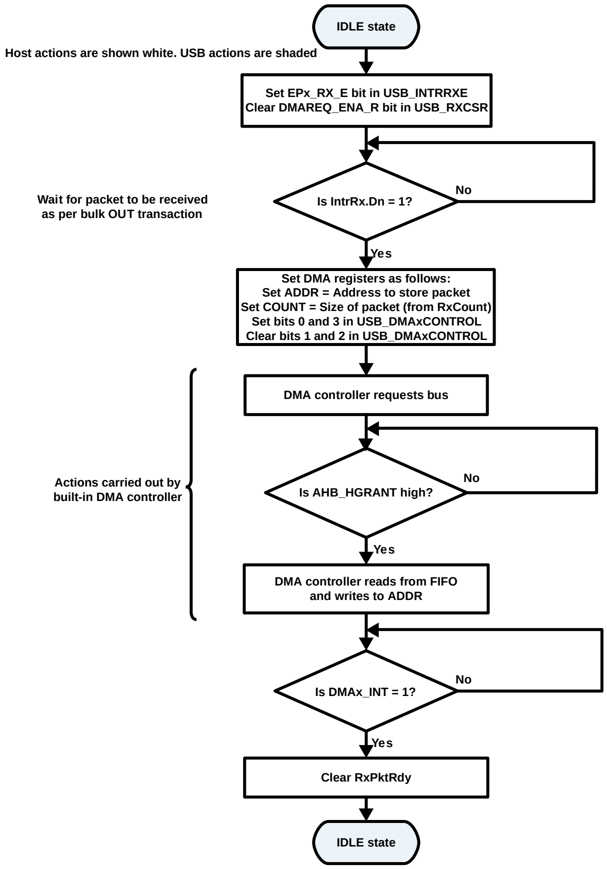

IDLE state
Host actions are shown white. USB actions are shaded
Set EPx_RX_E bit in USB_INTRRXE
Clear DMAREQ_ENA_R bit in USB_RXCSR
Wait for packet to be received
as per bulk OUT transaction
Is IntrRx.Dn = 1?
No
Yes
Set DMA registers as follows:
Set ADDR = Address to store packet
Set COUNT = Size of packet (from RxCount)
Set bits 0 and 3 in USB_DMAxCONTROL
Clear bits 1 and 2 in USB_DMAxCONTROL
DMA controller requests bus
Actions carried out by
built-in DMA controller
Is AHB_HGRANT high?
No
Yes
DMA controller reads from FIFO
and writes to ADDR
Is DMAx_INT = 1?
No
Yes
Clear RxPktRdy
IDLE state

Figure 27-30: Single Packet Receive During DMA Operation

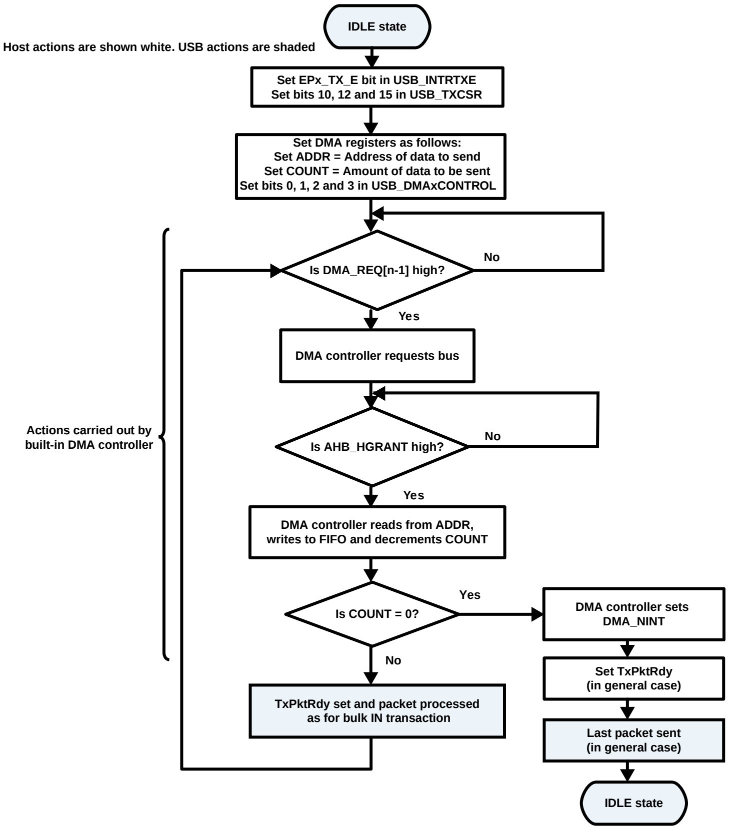

IDLE state
Host actions are shown white. USB actions are shaded
Set EPx_TX_E bit in USB_INTRTXE
Set bits 10, 12 and 15 in USB_TXCSR
Set DMA registers as follows:
Set ADDR = Address of data to send
Set COUNT = Amount of data to be sent
Set bits 0, 1, 2 and 3 in USB_DMAxCONTROL
Is DMA_REQ[n-1] high?
No
Yes
DMA controller requests bus
Actions carried out by built-in DMA controller
Is AHB_HGRANT high?
No
Yes
DMA controller reads from ADDR, writes to FIFO and decrements COUNT
Is COUNT = 0?
Yes
No
DMA controller sets DMA_NINT
Set TxPktRdy (in general case)
Last packet sent (in general case)
IDLE state
TxPktRdy set and packet processed as for bulk IN transaction

Figure 27-31: Multiple Packet Transmit During DMA Operation

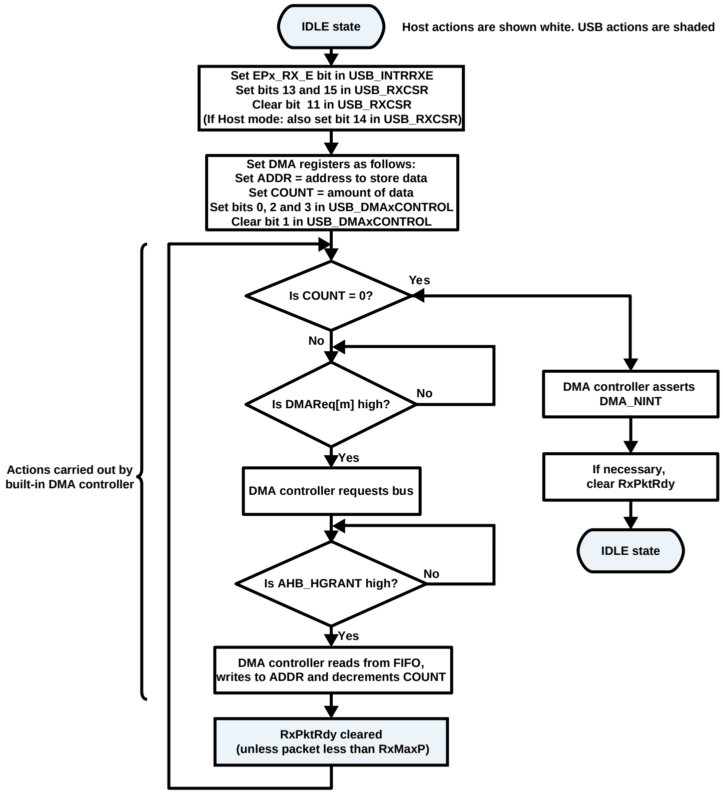

IDLE state
Host actions are shown white. USB actions are shaded
Set EPx_RX_E bit in USB_INTRRXE
Set bits 13 and 15 in USB_RXCSR
Clear bit 11 in USB_RXCSR
(If Host mode: also set bit 14 in USB_RXCSR)
Set DMA registers as follows:
Set ADDR = address to store data
Set COUNT = amount of data
Set bits 0, 2 and 3 in USB_DMAxCONTROL
Clear bit 1 in USB_DMAxCONTROL
Is COUNT = 0?
Yes
No
Is DMAReq[m] high?
No
Yes
DMA controller asserts DMA_NINT
If necessary, clear RxPktRdy
IDLE state
Actions carried out by built-in DMA controller
DMA controller requests bus
Is AHB_HGRANT high?
No
Yes
DMA controller reads from FIFO, writes to ADDR and decrements COUNT
RxPktRdy cleared (unless packet less than RxMaxP)

Figure 27-32: Multiple Packet Receive During DMA Operation (Data Size Known)

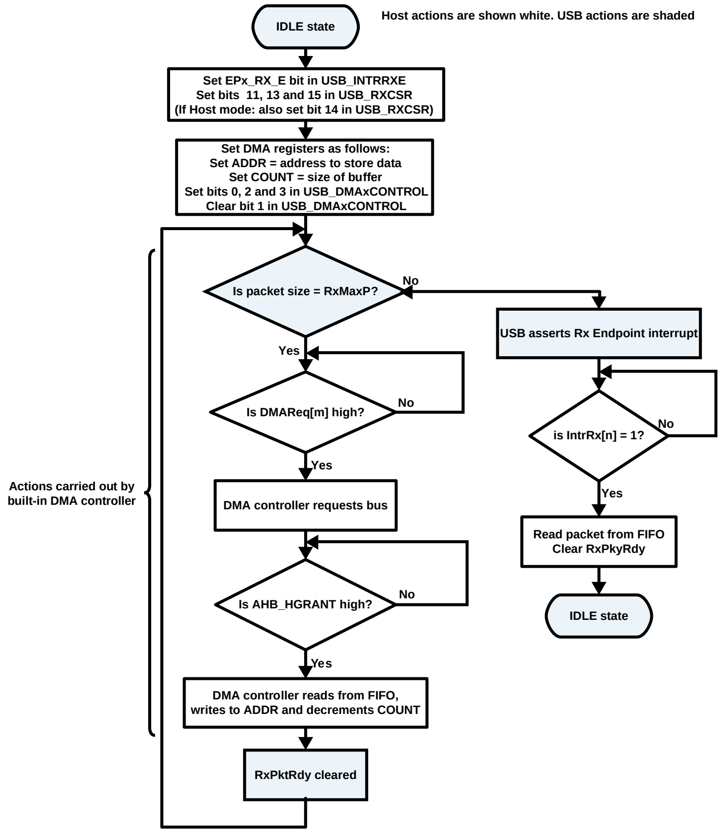

IDLE state
Host actions are shown white. USB actions are shaded
Set EPx_RX_E bit in USB_INTRRXE
Set bits 11, 13 and 15 in USB_RXCSR
(If Host mode: also set bit 14 in USB_RXCSR)
Set DMA registers as follows:
Set ADDR = address to store data
Set COUNT = size of buffer
Set bits 0, 2 and 3 in USB_DMAxCONTROL
Clear bit 1 in USB_DMAxCONTROL
Is packet size = RxMaxP?
No
Yes
USB asserts Rx Endpoint interrupt
Is DMAReq[m] high?
No
is IntrRx[n] = 1?
No
Yes
Yes
Actions carried out by built-in DMA controller
DMA controller requests bus
Read packet from FIFO
Clear RxPkyRdy
Is AHB_HGRANT high?
No
IDLE state
Yes
DMA controller reads from FIFO, writes to ADDR and decrements COUNT
RxPktRdy cleared

Figure 27-33: Multiple Packet Receive During DMA Operation (Data Size Not-known)

OTG Session Request

To conserve power, the USB on-the-go supplement allows VBUS to only power-up when required and to turn off when the bus is not in use.

The A device on the bus always supplies VBUS. The USB controller samples the `USB_ID` input from the PHY to determine whether it is the A device or the B device. The signal is pulled low when an A-type plug is sensed (signifying that the USB controller is the A device). The input is taken high when a B-type plug is sensed (signifying that the USB controller is the B device).

Starting a Session

When the device containing the USB controller wants to start a session, the processor core must set the `USB_DEV_CTL.SESSION` bit. The USB controller then enables ID pin sensing. This activity results in the `USB_ID` input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The `USB_DEV_CTL.BDEVICE` bit is also set to indicate whether the USB controller has adopted the role of the A device or the B device.

The USB controller is the A device. The USB controller then enters host mode (the A device is always the default host). It waits for VBUS to go above the VBUS valid threshold, as indicated when the `USB_DEV_CTL.VBUS` bits go to 11.

The USB controller then waits for a peripheral to be connected. When the USB controller detects a peripheral, it generates a connect interrupt (`USB_IRQ.CON` bit) (if enabled). It sets either the `USB_DEV_CTL.FSDEV` or `USB_DEV_CTL.LSDEV` bits, depending on whether a full-speed peripheral or a low-speed peripheral was detected. The processor core then resets this peripheral. To end the session, the processor core must clear the `USB_DEV_CTL.SESSION` bit.

The USB controller is the B device. The USB controller requests a session using the session request protocol defined in the USB on-the-go supplement. This functionality is accomplished by setting the `USB_DEV_CTL.SESSION` bit.

At the end of the session, typically the USB controller clears the `USB_DEV_CTL.SESSION` bit. But, the processor core can also clear it when the application software must perform a software disconnect. For more information, see the description of the `USB_DEV_CTL` register. The USB controller switches on the pull-up resistor on D+. This activity signals to the A device to end the session.

Detecting Activity

When the other device of the OTG set-up wants to start a session, USB controller either:

- Raises VBUS above the session valid threshold (if A device) or
- First pulses the data line, then pulses VBUS (if B device)

Depending on which of these actions happens, the USB controller can determine whether it is the A device or the B device in the current set-up and act accordingly. (The `USB_DEV_CTL.VBUS` bits=10 indicates if it is the A device.)

If VBUS is raised above the session valid threshold, the USB controller is the B device. The USB controller sets the `USB_DEV_CTL.SESSION` bit. When reset signaling is detected on the bus, a reset interrupt (`USB_IRQ.RSTBABBLE=1`) is generated (if enabled) that the processor core interprets as the start of a session. The USB controller is in peripheral mode as the B device is the default peripheral.

At the end of the session, the A device turns off the power to VBUS. When VBUS drops below the session valid threshold, the USB controller detects this state and clears the `USB_DEV_CTL.SESSION` bit to indicate that the session has ended. (The `USB_DEV_CTL.VBUS` bits=01 indicates that VBUS has dropped below the session valid threshold). A disconnect interrupt (`USB_IRQ.DISCON` bit) is also generated (if enabled).

If data line or VBUS pulsing is detected, the USB controller is the A device. The controller generates a `USB_IRQ.SESSREQ` interrupt to indicate that the B device is requesting a session. The processor core must then start a session by setting the `USB_DEV_CTL.SESSION` bit.

Host Negotiation Protocol

When the USB controller is the A device (`USB_ID` low, `USB_DEV_CTL.BDEVICE=0`), the controller automatically enters host mode when a session starts.

When the USB controller is the B device (`USB_ID` high, `USB_DEV_CTL.BDEVICE=1`), the controller automatically enters peripheral mode when a session starts. The processor core can request that the USB controller become the host by setting the `USB_DEV_CTL.HOSTREQ` bit. This bit can be set either when requesting a session start by setting the `USB_DEV_CTL.SESSION` bit or at any time after a session has started.

When the USB controller enters suspend mode (no activity on the bus for 3 ms), and assuming the `USB_DEV_CTL.HOSTREQ` bit remains set, it then enters host mode. It begins host negotiation (as specified in the USB OTG supplement), causing the PHY to disconnect the pull-up resistor on the D+ line. This event causes the A device to switch to peripheral mode and to connect its own pull-up resistor. When the USB controller detects this activity, it generates a connect interrupt (`USB_IRQ.CON` bit). The controller also sets the `USB_POWER.RESET` bit to begin resetting the A device. (The USB controller begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the A device connecting its pull-up resistor.) The processor core must wait at least 20 ms, then clear the `USB_POWER.RESET` bit and enumerate the A device.

When the USB controller-based B device has finished using the bus, the processor core must put it into suspend mode by setting the `USB_POWER.SUSPEND`. The A device detects this state and either terminates the session or reverts to host mode. If the A device is USB controller-based, it generates a disconnect interrupt (`USB_IRQ.DISCON` bit) if enabled.

Data Transfer

Whether the USB controller is operating in host or peripheral mode, data channels through the endpoint FIFOs to construct packets that are sent or received over the USB. The USB controller uses the Rx FIFOs to receive OUT packets when in peripheral mode and IN packets when operating in host mode. Similarly, The USB controller uses the Tx FIFOs to transmit IN packets when in peripheral mode and OUT packets as a host.

Data can be moved between the FIFOs and memory using either DMA or core accesses. Each endpoint FIFO has its own individually programmable options so that each can be set up separately. The system must treat each transfer type differently. Data transfers of significant size almost certainly require DMA to move the data around; but the processor can handle smaller packet sizes completely.

Each data endpoint supports both double and single-buffering modes. In single-buffered operation, the processor loads and unloads FIFOs on a packet-by-packet basis. Double-buffering imposes less burden on the system by allowing two packets to be buffered in a FIFO before it is necessary to use DMA or interrupts to service the FIFO. Double-buffering mode is automatically enabled when a *MaxPktSize* is set for an endpoint that is equal to or less than half the size in bytes of that FIFO.

Loading or Unloading Packets from Endpoints

Transfers to and from the FIFOs can be 32-bit, 16-bit, or 8-bit. When using core accesses, use the same width for transfers associated with one data packet, so that data is consistently byte, half-word, or word aligned. The last transfer can, however, contain fewer bytes than the previous transfers in order to complete an 8-bit or 16-bit transfer.

When using the DMA to access the FIFOs, the starting DMA address must be word aligned, or aligned on a 32-bit boundary. The packet transfer starts with a word transfer, but half-word or byte transfers can be added at the end to handle any leftovers.

DMA Master Channels

The USB controller provides eight DMA master channels.

These channels provide a more efficient transfer of larger amounts of data between the FIFOs and the processor core, and the channels free up the processor core for other tasks. The processor uses the DMA control registers to configure and control each of these channels.

Each DMA controller can operate in one of two DMA modes: 0 or 1. When operating in mode 0, the DMA controller can only be programmed to load or unload one packet, so processor intervention is required for each packet transferred over the USB. The DMA controller can use this mode with any endpoint, whether it uses control, bulk, isochronous, or interrupt transactions.

When operating in DMA mode 1, the DMA controller can only be programmed to load or unload a complete bulk transfer, which can be many packets. After set up, the DMA controller loads or unloads the packets, interrupting the processor only when the transfer has completed. DMA mode 1 can only be used with endpoints that use bulk transactions. It is most valuable where large blocks of data are transferred to a bulk endpoint. The USB protocol requires splitting such packets into a series of packets of *MaxPktSize* for the endpoint.

The DMA controller can use mode 1 to avoid the overhead of having to interrupt the processor after each individual packet. It interrupts the processor only after the transfer completes. In some cases, the block of data transferred consists of a predefined number of these packets that the controlling software counts through the transfer process. In other cases, the last packet in the series can be less than the maximum packet size. The receiver can use this short packet to signal the end of the transfer. If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software must send a null packet for the receiver to detect.

NOTE: Each channel can be independently programmed for the selected operating mode.

For bulk OUT transfers using DMA mode 1, the DMA request line is asserted only when:

- There is an edge transition of the state of the `USB_EP[n]_RXCSR_H.RXPKTRDY`, and
- A payload of *MaxPktSize* has been received

If a data packet is in the FIFO prior to setting the DMA request mode bits, the DMA request line is not asserted when the DMA is enabled. DMA is enabled using the `USB_DMA[n]_CTL.EN` bit. (DMA request mode bits are `USB_EP[n]_RXCSR_H.DMAREQMODE` or `USB_EP[n]_RXCSR_P.DMAREQMODE`). The data is not read from the Rx FIFO in this situation, resulting in a DMA hang. However, since the packet arrived before DMA request mode and DMA request enable bits (`USB_EP[n]_RXCSR_H.DMAREQEN` or `USB_EP[n]_RXCSR_P.DMAREQEN`) were enabled, an Rx interrupt is generated for the corresponding endpoint. Therefore, the software must set the DMA request mode to request mode 0 to unload the pre-received packet. The Rx interrupt service routine can be similar to the following:

If `USB_EP[n]_RXCNT` = MaxPktSize

Switch to DMA mode 0 and unload the packet (in mode 0, the DMA request enable is always asserted, whenever there is data in the FIFO)

Set the `USB_EP[n]_RXCNT` to MaxPktSize so as to unload only one packet

If `USB_EP[n]_RXCSR_H.AUTOCLR` is set, it is not necessary to clear `USB_EP[n]_RXCSR_H.RXPKTRDY` manually

Switch back to DMA Mode 1 and set the count to

(Total_Count – MaxPktSize)

Else

Handle as normal for case of short packet

DMA transfers can be 8-bit, 16-bit, or 32-bit. All transfers associated with one packet (except for the last) must be of the same width, so that the data is consistently byte-aligned or word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

DMA Bus Cycles

The DMA controller uses incrementing bursts of an unspecified length on the peripheral DMA bus. The controller starts a new burst when it is first granted bus mastership and when the peripheral address starts a new 1 KB block. (The controller is granted bus mastership at the start of a USB packet or when regaining the bus after losing it following a partial packet).

When unloading packets from the FIFOs, the DMA controller requests ahead to the USB controller. The DMA controller starts the transfer with two BUSY cycles while it is gets the first word from the FIFO. All subsequent words of the packet are immediately available and no further BUSY cycles are required. The DMA controller is associated with a two-word buffer, so it does not lose data when it loses bus mastership in the middle of unloading a packet. When the controller regains bus mastership, it can continue unloading the packet without adding any BUSY cycles.

The DMA start address (written to the `USB_DMA[n]_ADDR` register) must be word aligned. The DMA controller supports split transactions and retries.

The DMA request lines are individually enabled using the appropriate DMA request enable bit (there are four options: Tx peripheral and host and Rx peripheral and host) and operate in two modes, referred to as DMA request mode 0 and DMA request mode 1. The operating mode is configured using the appropriate DMA request mode bit (there are four options: Tx peripheral and host and Rx peripheral and host).

NOTE: When operating in host mode, if the `USB_EP[n]_TXCSR_H.RXSTALL` bit or the `USB_EP[n]_TXCSR_H.TXTOERR` is set following three failed transmit attempts, the DMA request line is disabled until the bits are cleared.

The mode selected also affects the generation of endpoint interrupts (if enabled). In DMA request mode 0, no interrupt is generated when packets are received but the appropriate endpoint interrupt is generated to prompt the loading of all packets. In DMA request mode 1, the endpoint interrupt is suppressed except following the receipt of a short packet (one less than `USB_EP[n]_RXMAXP` bytes).

Table 27-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	NO
1	1	Only is short packet

Table 27-8: Endpoint Interrupt Associated with the Receive Packet Ready Bit=0

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	YES
1	1	NO

NOTE: Set the `USB_EP[n]_TXMAXP`/`USB_EP[n]_RXMAXP` registers to an even number of bytes for proper interrupt generation in DMA mode 1.

DMA transfers can be 8-bit, 16-bit, or 32-bit as required. However, all transfers associated with one packet (except for the last) must be of the same width so that the data is consistently byte-aligned, word-aligned, or double-word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

NOTE: Disable DMA requests before changing the DMA request mode bit. In particular, do not set the `USB_EP[n]_TXCSR_H.DMAREQMODE` bit to zero either before or in the same cycle as the corresponding `USB_EP[n]_TXCSR_H.DMAREQEN` bit is cleared to zero.

Transferring Packets Using DMA

Both the channel and the endpoint must be programmed appropriately to use the the DMA master channels to access the USB controller FIFOs. Many variations are possible. The following sections detail the standard set ups used for the basic actions of transferring individual and multiple packets.

Individual Rx Endpoint Packet

The transfer of individual packets normally uses DMA mode 0. Program the USB controller Rx endpoint as follows.

1. Set to 1 the relevant bit in the `USB_INTRRXE` register.
2. Set to 0 the DMA enable bit of the appropriate `USB_EP[n]_RXCSR_H.DMAREQEN`/`USB_EP[n]_RXCSR_P.DMAREQEN` register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the USB controller receives a packet, it generates the appropriate endpoint interrupt (using the `USB_INTRRXE` register). The processor must then program the appropriate DMA master channel as follows.
    - Configure the `USB_DMA[n]_ADDR` register with the memory address to store the packet

- Configure the USB_DMA[n]_CNT register with the size of packet (determined by reading the USB controller USB_RQPKTCNT[n] register)
- Configure the USB_DMA[n]_CTL register using the following bit settings:USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=0

The DMA controller then requests bus mastership and transfers the packet to memory. It interrupts the processor when it has completed the transfer. The processor then clears the USB_EP[n]_RXCSR_H.RXPKTRDY bit.

Individual Tx Endpoint Packet

Using DMA mode 0, program a USB controller Tx endpoint as follows.

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 0 the DMA enable bit of the appropriate USB_EP[n]_TXCSR_H.DMAREQEN/ USB_EP[n]_TXCSR_P.DMAREQEN register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the FIFO can accommodate data, the USB controller interrupts the processor with the appropriate Tx endpoint interrupt. The processor must then program the DMA channel as follows:
   - Configure the USB_DMA[n]_ADDR register with the memory address to store the packet.
   - Configure the USB_DMA[n]_CNT register with the size of packet.
   - Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=0.

The DMA controller then requests bus mastership and transfers the packet to the USB controller FIFO. When it has completed the transfer, it generates a DMA interrupt. The processor then sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit.

Multiple Rx Endpoint Packets

The transfer of multiple packets normally uses DMA mode 1. Program the DMA controller using the DMA registers:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the maximum size of data buffer.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Rx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRRX register.
2. Set to 1 the USB_EP[n]_RXCSR_H.AUTOCLR, USB_EP[n]_RXCSR_H.DMAREQEN, and USB_EP[n]_RXCSR_H.DMAREQMODE bits of the appropriate receive control and status register (host or

peripheral). In host mode, set to 1 the USB_EP[n]_RXCSR_H.AUTOREQ and USB_EP[n]_RXCSR_H.DMAREQMODE bits.

As the USB controller receives each packet, the DMA master channel requests bus mastership and transfers the packet to memory. With USB_EP[n]_RXCSR_H.AUTOCLR set, the USB controller automatically clears its USB_EP[n]_RXCSR_H.RXPKTRDY bit. This process continues automatically until the USB controller receives a short packet (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. The DMA controller does not transfer this short packet: instead the USB controller interrupts the processor by generating the appropriate endpoint interrupt. The processor can then read the USB_EP[n]_RXCNT register to see the size of the short packet. It either unloads the packet manually or reprograms the DMA controller in mode 0 to unload the packet.

The USB_DMA[n]_ADDR register is incremented as the DMA controller unloads the packets. The processor determines the size of the transfer by comparing the current value of USB_DMA[n]_ADDR with the start address of the memory buffer.

If the size of the transfer exceeds the data buffer size, the DMA controller stops unloading the FIFO and interrupts the processor.

Multiple Tx Endpoint Packets

Using DMA mode 1 for a Tx endpoint, program the DMA controller as follows:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the size of the data block.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Tx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 1 the USB_EP[n]_TXCSR_H.AUTOSET and USB_EP[n]_TXCSR_H.DMAREQEN bits of the appropriate transmit control and status register (host or peripheral).

When the FIFO in the USB controller becomes available, the DMA controller requests bus mastership and transfers a packet to the FIFO. With USB_EP[n]_TXCSR_H.AUTOSET set, the USB controller automatically sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit. This process continues until the entire data block is transferred to the USB controller.

The DMA controller then interrupts the processor by taking the appropriate USB_DMA_IRQ register bit low.

- If the last packet loaded was less than the maximum packet size for the endpoint, the USB_EP[n]_TXCSR_H.TXPKTRDY bit is not set for this packet. The processor must respond to the DMA interrupt by setting the USB_EP[n]_TXCSR_H.TXPKTRDY bit to allow the last short packet to be sent.

- If the last packet loaded is the maximum packet size, then the appropriate action depends on whether the transfer is under the control of an application. One example is the mass storage software on Windows system that keeps count of the individual packets sent.
- If the transfer is not under such control, the processor must respond to the DMA interrupt by setting the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. This operation sends a null packet for the receiving software to interpret as indicating the end of the transfer.

ADSP-SC58x USB Register Descriptions

Universal Serial Bus Controller (USB) contains the following registers.

Table 27-9: ADSP-SC58x USB Register List

Name	Description
`USB_BAT_CHG`	Battery Charging Control Register
`USB_CT_HHSRTN`	Host High-Speed Return to Normal Register
`USB_CT_HSBT`	High-Speed Timeout Register
`USB_CT_UCH`	Chirp Timeout Register
`USB_DEV_CTL`	Device Control Register
`USB_DMA[n]_ADDR`	DMA Channel n Address Register
`USB_DMA[n]_CNT`	DMA Channel n Count Register
`USB_DMA[n]_CTL`	DMA Channel n Control Register
`USB_DMA_IRQ`	DMA Interrupt Register
`USB_EP0I_CFGDATA[N]`	EP0 Configuration Information Register
`USB_EP0I_CNT[N]`	EP0 Number of Received Bytes Register
`USB_EP0I_CSR[N]_H`	EP0 Configuration and Status (Host) Register
`USB_EP0I_CSR[N]_P`	EP0 Configuration and Status (Peripheral) Register
`USB_EP0I_NAKLIMIT[N]`	EP0 NAK Limit Register
`USB_EP0I_TYPE[N]`	EP0 Connection Type Register
`USB_EP0_CFGDATA[n]`	EP0 Configuration Information Register
`USB_EP0_CNT[n]`	EP0 Number of Received Bytes Register
`USB_EP0_CSR[n]_H`	EP0 Configuration and Status (Host) Register
`USB_EP0_CSR[n]_P`	EP0 Configuration and Status (Peripheral) Register
`USB_EP0_NAKLIMIT[n]`	EP0 NAK Limit Register
`USB_EP0_TYPE[n]`	EP0 Connection Type Register
`USB_EPINFO`	Endpoint Information Register

Table 27-9: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_IDCTL	ID Control
USB_IEN	Common Interrupts Enable Register

Table 27-9: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_VBUS_CTL	VBUS Control Register

Table 27-9: ADSP-SC58x USB Register List (Continued)

Name	Description
USB_VPLEN	VBUS Pulse Length Register

Battery Charging Control Register

The USB_BAT_CHG register controls USB controller battery change-related features.

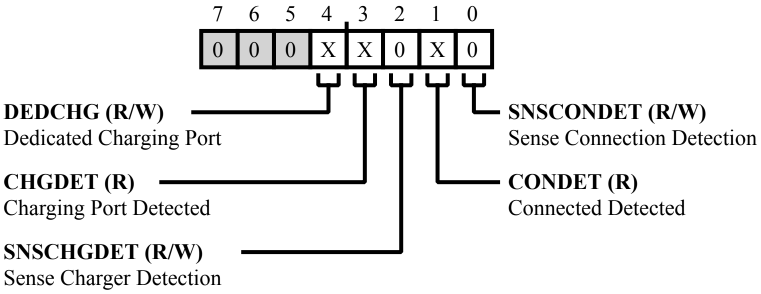

7 6 5 4 3 2 1 0
0 0 0 X X 0 X 0
DEDCHG (R/W)
Dedicated Charging Port
CHGDET (R)
Charging Port Detected
SNSCHGDET (R/W)
Sense Charger Detection
SNSCONDET (R/W)
Sense Connection Detection
CONDET (R)
Connected Detected

Figure 27-34: USB_BAT_CHG Register Diagram

Table 27-10: USB_BAT_CHG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	DEDCHG	Dedicated Charging Port. The `USB_BAT_CHG.DEDCHG` bit is asserted if both D+ and D- are high. This can be used to determine if the attached device is a dedicated charging port. This bit is the decode of LineState[1] and LineState[0]. This bit is only valid when a session is initiated, which enables a pullup on D+ when acting as a B-device.
3 (R/NW)	CHGDET	Charging Port Detected. The `USB_BAT_CHG.CHGDET` bit indicates when a charging port is detected. This bit indicates that D+/- is above V_{DAT_REF} and below V_{LGC}.
2 (R/W)	SNSCHGDET	Sense Charger Detection. The `USB_BAT_CHG.SNSCHGDET` bit enables charger detection. Setting this bit enables VD_SRC and ID_SINK.
1 (R/NW)	CONDET	Connected Detected. The `USB_BAT_CHG.CONDET` bit is valid when `USB_BAT_CHG.SNSCONDET` is enabled. This bit reflects the inverse of D+ (!LineState[0]). If nothing is connected, D+ is pulled high. If a charger or USB port is connected, D+ is pulled low.
0 (R/W)	SNSCONDET	Sense Connection Detection. The `USB_BAT_CHG.SNSCONDET` bit enables connection detection. Enabling this bit enables I_{DP_SRC} and R_{DM_DWN}.

Host High-Speed Return to Normal Register

The `USB_CT_HHSRTN` register selects the delay from the end of the high-speed resume signaling (acting as a host) to the return to normal mode operation. This value is multiplied by 4 times the XCLK period (or 16.7 ns). The default setting corresponds to a delay of 100us.

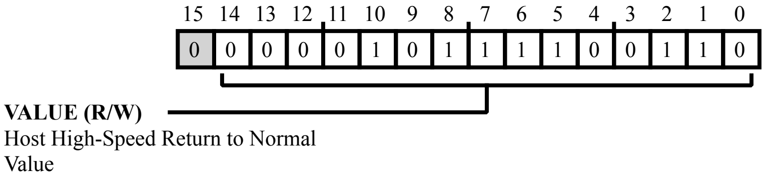

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 1 1 1 1 0 0 1 1 0
VALUE (R/W)
Host High-Speed Return to Normal Value

Figure 27-35: USB_CT_HHSRTN Register Diagram

Table 27-11: USB_CT_HHSRTN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Host High-Speed Return to Normal Value.

High-Speed Timeout Register

The USB_CT_HSBT register selects an amount of time to add to the minimum high-speed timeout in units of 64 bit times. The USB 2.0 specification section 7.1.19.2 states that the controller must not timeout less than 736 bit times and must timeout after 816 bit times. The value in USB_CT_HSBT is multiplied by 64-bit times and added to the minimum 736 bit times. Settings less than 1 violate the USB 2.0 specification, making the controller non-compliant.

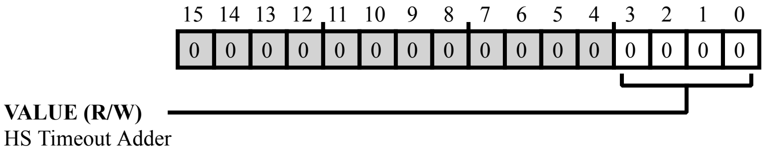


Figure 27-36: USB_CT_HSBT Register Diagram

Table 27-12: USB_CT_HSBT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	VALUE	HS Timeout Adder. The USB_CT_HSBT.VALUE bits selects an amount of time to add to the minimum high-speed timeout in units of 64 bit times.	
		0	HS Timeout = 736 (bit time)
		1	HS Timeout = 800 (bit time)
		2	HS Timeout = 864 (bit time)

Chirp Timeout Register

The `USB_CT_UCH` register selects chirp timeout value. The value is multiplied by 4 times the XCLK period (or 67ns). The default setting is 1.1ms.

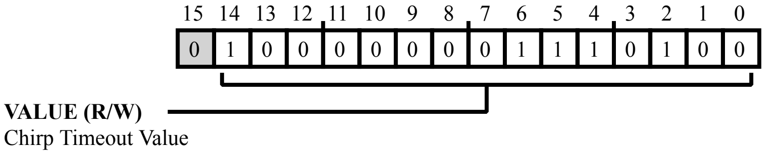

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 0 0 0 0 1 1 1 0 1 0 0
VALUE (R/W)
Chirp Timeout Value

Figure 27-37: USB_CT_UCH Register Diagram

Table 27-13: USB_CT_UCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Chirp Timeout Value. The `USB_CT_UCH.VALUE` bits select the chirp timeout value.

Device Control Register

The `USB_DEV_CTL` register selects whether the USB controller is operating in peripheral mode or in host mode. It is used for controlling and monitoring the VBUS line.

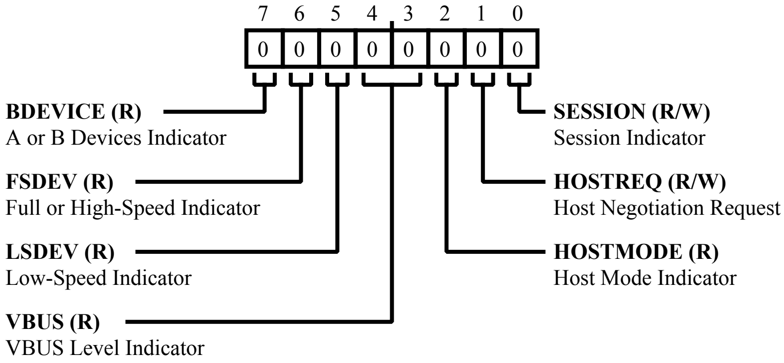

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
BDEVICE (R)
A or B Devices Indicator
FSDEV (R)
Full or High-Speed Indicator
LSDEV (R)
Low-Speed Indicator
VBUS (R)
VBUS Level Indicator
SESSION (R/W)
Session Indicator
HOSTREQ (R/W)
Host Negotiation Request
HOSTMODE (R)
Host Mode Indicator

Figure 27-38: USB_DEV_CTL Register Diagram

Table 27-14: USB_DEV_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	BDEVICE	A or B Devices Indicator. The `USB_DEV_CTL.BDEVICE` bit indicates whether the USB controller is operating as the A device or the B device. This bit is only valid while a session is in progress.	
		0	A Device Detected
		1	B Device Detected
6 (R/NW)	FSDEV	Full or High-Speed Indicator. The `USB_DEV_CTL.FSDEV` bit is set when a full-speed or high-speed device is detected being connected to the port. High-speed devices are distinguished from full-speed by checking for high-speed chirps when the device detects a USB controller reset. This bit is only valid in host mode.	
		0	Not Detected
		1	Full or High-Speed Detected
5 (R/NW)	LSDEV	Low-Speed Indicator. The `USB_DEV_CTL.LSDEV` bit is set when a low-speed device is detected being connected to the port. This bit is only valid in host mode.	
		0	Not Detected
		1	Low-Speed Detected

Table 27-14: USB_DEV_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/NW)	VBUS	VBUS Level Indicator. The `USB_DEV_CTL.VBUS` bits indicated the current VBUS level.	
		0	Below SessionEnd
		1	Above SessionEnd, below AValid
		2	Above AValid, below VBUSValid
		3	Above VBUSValid
2 (R/NW)	HOSTMODE	Host Mode Indicator. The `USB_DEV_CTL.HOSTMODE` bit is set when the USB controller is acting as a host.	
		0	Peripheral Mode
		1	Host Mode
1 (R/W)	HOSTREQ	Host Negotiation Request. When the `USB_DEV_CTL.HOSTREQ` bit is set, the USB controller initiates the host negotiation when suspend mode is entered. This bit is cleared when host negotiation is completed. The `USB_DEV_CTL.HOSTREQ` bit applies when the USB controller is operating as a B device only.	
		0	No Request
		1	Place Request
0 (R/W)	SESSION	Session Indicator. When operating as an A device, the `USB_DEV_CTL.SESSION` bit is set or cleared by the processor core to start or end a session. When operating as a B device, the `USB_DEV_CTL.SESSION` bit is set or cleared by the USB controller when a session starts or ends. This bit is also set by the processor core to initiate the session request protocol. When the USB controller is in suspend mode, the bit may be cleared by the processor core to perform a software disconnect.	
		0	Not Detected
		1	Detected Session

DMA Channel n Address Register

The `USB_DMA[n]_ADDR` register indicates the location in on-chip memory where DMA data is written or read. The address must be aligned to 32-bit words (The lower two address bits are always zero.) This register increments as the DMA transfer progresses.

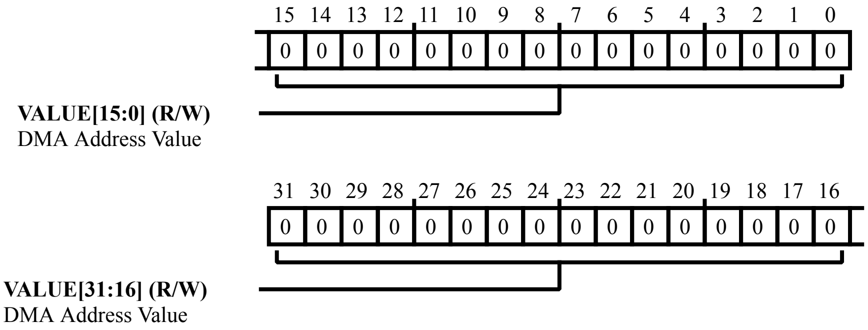

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Address Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Address Value

Figure 27-39: USB_DMA[n]_ADDR Register Diagram

Table 27-15: USB_DMA[n]_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Address Value. The `USB_DMA[n]_ADDR.VALUE` bits hold the address value for the location in on-chip memory where DMA data is written or read.

DMA Channel n Count Register

The `USB_DMA[n]_CNT` register holds the DMA count, indicating the number of bytes to be transferred for a given DMA work block. If this field is set to zero, no data is transferred, and an interrupt is generated.

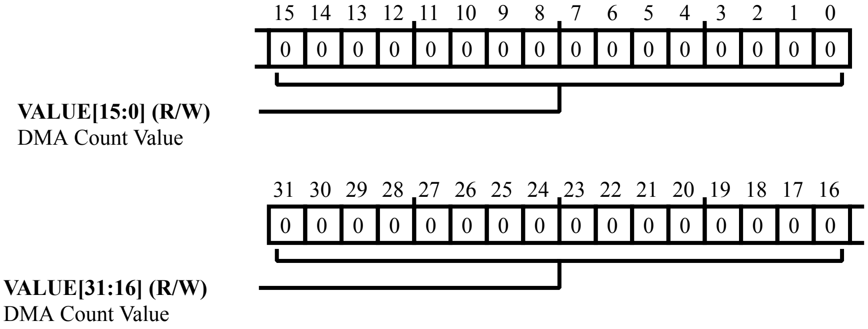

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Count Value

Figure 27-40: USB_DMA[n]_CNT Register Diagram

Table 27-16: USB_DMA[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Count Value. The `USB_DMA[n]_CNT.VALUE` bits indicate the number of bytes to be transferred for a given DMA work block.

DMA Channel n Control Register

There is a `USB_DMA[n]_CTL` register for each DMA master channel. This register assigns, configures, and controls each endpoint with a corresponding DMA master channel.

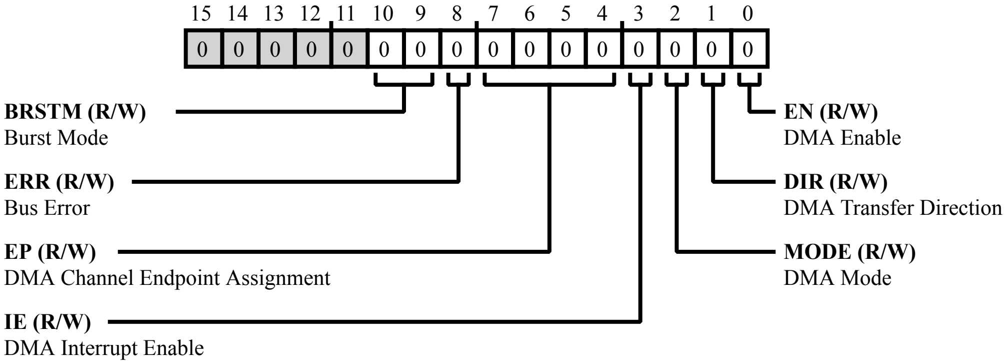

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BRSTM (R/W)
Burst Mode
ERR (R/W)
Bus Error
EP (R/W)
DMA Channel Endpoint Assignment
IE (R/W)
DMA Interrupt Enable
EN (R/W)
DMA Enable
DIR (R/W)
DMA Transfer Direction
MODE (R/W)
DMA Mode

Figure 27-41: USB_DMA[n]_CTL Register Diagram

Table 27-17: USB_DMA[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/W)	BRSTM	Burst Mode. The `USB_DMA[n]_CTL.BRSTM` bits select the type or length of burst transfer used by the corresponding DMA channel to transfer data.	
		0	Unspecified Length
		1	INCR4 or Unspecified Length
		2	INCR8, INCR4, or Unspecified Length
		3	INCR16, INCR8, INCR4, or Unspecified Length
8 (R/W)	ERR	Bus Error. The `USB_DMA[n]_CTL.ERR` bit indicates when a peripheral bus error has been encountered by the master channel. This bit is cleared by software.	
		0	No Status
		1	Bus Error
7:4 (R/W)	EP	DMA Channel Endpoint Assignment. The `USB_DMA[n]_CTL.EP` bits select the endpoint assignments for the DMA channel. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3

Table 27-17: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15
3 (R/W)	IE	DMA Interrupt Enable. The `USB_DMA[n]_CTL.IE` bit enables DMA interrupts for the DMA channel, enabling operation of the channels corresponding bit in the `USB_DMA_IRQ` register.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	MODE	DMA Mode. The `USB_DMA[n]_CTL.MODE` bit selects whether the DMA channel operates in DMA mode 0 or operates in DMA mode 1. Note that DMA mode 1 may only be used with bulk endpoints.	
		0	DMA Mode 0
		1	DMA Mode 1
1 (R/W)	DIR	DMA Transfer Direction. The `USB_DMA[n]_CTL.DIR` bit selects the DMA channel transfer direction, which must be selected for use with receive endpoints (DMA write=0) or transmit endpoints (DMA read=1).	
		0	DMA Write (for Rx Endpoint)
		1	DMA Read (for Tx Endpoint)

Table 27-17: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Enable. The `USB_DMA[n]_CTL.EN` bit enables the DMA channel to start the DMA transfer.	
		0	Disable DMA
		1	Enable DMA (Start Transfer)

DMA Interrupt Register

The USB_DMA_IRQ register indicates which of the DMA master channels have a pending interrupt. The USB controller generates the interrupt when the corresponding DMA count register (USB_DMA[n]_CNT) reaches zero. The USB controller clears this register when it is read.

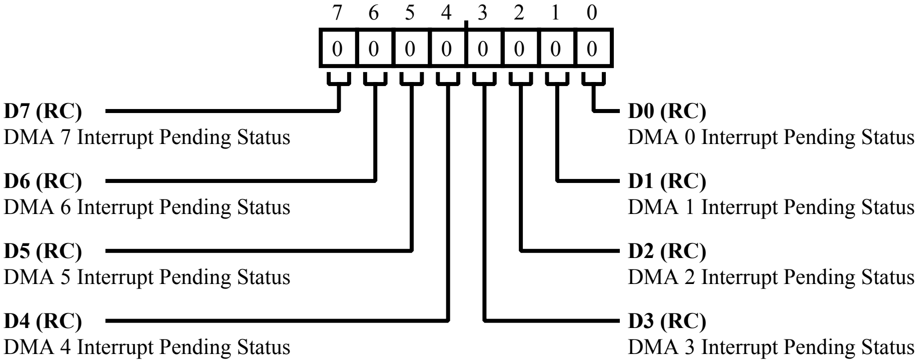

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
D7 (RC)
DMA 7 Interrupt Pending Status
D6 (RC)
DMA 6 Interrupt Pending Status
D5 (RC)
DMA 5 Interrupt Pending Status
D4 (RC)
DMA 4 Interrupt Pending Status
D0 (RC)
DMA 0 Interrupt Pending Status
D1 (RC)
DMA 1 Interrupt Pending Status
D2 (RC)
DMA 2 Interrupt Pending Status
D3 (RC)
DMA 3 Interrupt Pending Status

Figure 27-42: USB_DMA_IRQ Register Diagram

Table 27-18: USB_DMA_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	D7	DMA 7 Interrupt Pending Status. The USB_DMA_IRQ.D7 indicates whether there is a DMA 7 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
6 (RC/NW)	D6	DMA 6 Interrupt Pending Status. The USB_DMA_IRQ.D6 indicates whether there is a DMA 6 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
5 (RC/NW)	D5	DMA 5 Interrupt Pending Status. The USB_DMA_IRQ.D5 indicates whether there is a DMA 5 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
4 (RC/NW)	D4	DMA 4 Interrupt Pending Status. The USB_DMA_IRQ.D4 indicates whether there is a DMA 4 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

Table 27-18: USB_DMA_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	D3	DMA 3 Interrupt Pending Status. The `USB_DMA_IRQ.D3` indicates whether there is a DMA 3 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
2 (RC/NW)	D2	DMA 2 Interrupt Pending Status. The `USB_DMA_IRQ.D2` indicates whether there is a DMA 2 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
1 (RC/NW)	D1	DMA 1 Interrupt Pending Status. The `USB_DMA_IRQ.D1` indicates whether there is a DMA 1 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
0 (RC/NW)	D0	DMA 0 Interrupt Pending Status. The `USB_DMA_IRQ.D0` indicates whether there is a DMA 0 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

EP0 Configuration Information Register

The `USB_EP0I_CFGDATA[N]` register describes the USB controller hardware configuration. This register only exists for endpoint 0.

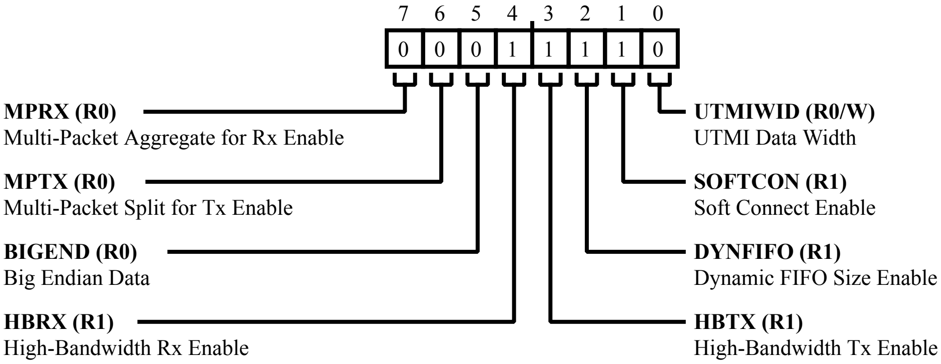

7 6 5 4 3 2 1 0
0 0 0 1 1 1 1 0
MPRX (R0)
Multi-Packet Aggregate for Rx Enable
MPTX (R0)
Multi-Packet Split for Tx Enable
BIGEND (R0)
Big Endian Data
HBRX (R1)
High-Bandwidth Rx Enable
UTMIWID (R0/W)
UTMI Data Width
SOFTCON (R1)
Soft Connect Enable
DYNFIFO (R1)
Dynamic FIFO Size Enable
HBTX (R1)
High-Bandwidth Tx Enable

Figure 27-43: USB_EP0I_CFGDATA[N] Register Diagram

Table 27-19: USB_EP0I_CFGDATA[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The `USB_EP0I_CFGDATA[N].MPRX` bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The `USB_EP0I_CFGDATA[N].MPTX` bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The `USB_EP0I_CFGDATA[N].BIGEND` bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 27-19: USB_EP0I_CFGDATA[N] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R1/NW)	HBRX	High-Bandwidth Rx Enable. The `USB_EP0I_CFGDATA[N].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint.	
		0	No High-Bandwidth Rx
		1	High-Bandwidth Rx
3 (R1/NW)	HBTX	High-Bandwidth Tx Enable. The `USB_EP0I_CFGDATA[N].HBTX` bit indicates whether the USB controller supports high-bandwidth transmit ISO endpoint.	
		0	No High-Bandwidth Tx
		1	High-Bandwidth Tx
2 (R1/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0I_CFGDATA[N].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0I_CFGDATA[N].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0I_CFGDATA[N].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The `USB_EP0I_CNT[N]` register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change. It is only valid while the `USB_EP0_CSR[n]_H.RXPKTRDY` bit or `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set.

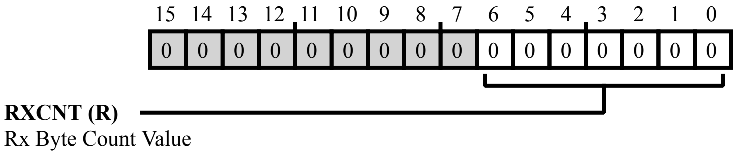


Figure 27-44: USB_EP0I_CNT[N] Register Diagram

Table 27-20: USB_EP0I_CNT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The `USB_EP0I_CNT[N].RXCNT` bits holds the number of data bytes currently in line ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded. It is only valid while the `USB_EP0_CSR[n]_H.RXPKTRDY` bit or `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set.

EP0 Configuration and Status (Host) Register

The `USB_EP0I_CSR[N]_H` register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

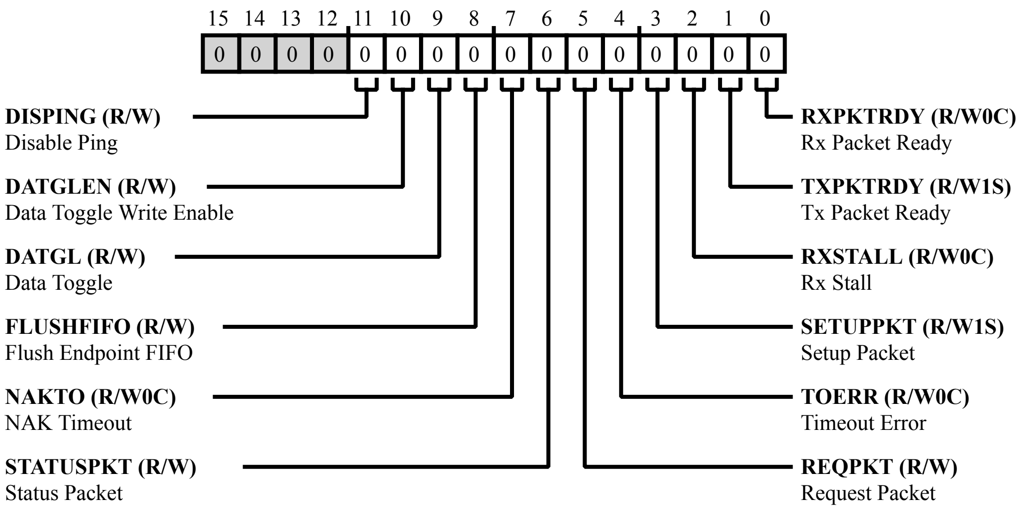

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DISPING (R/W)
Disable Ping
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
FLUSHFIFO (R/W)
Flush Endpoint FIFO
NAKTO (R/W0C)
NAK Timeout
STATUSPKT (R/W)
Status Packet
RXPKTRDY (R/W0C)
Rx Packet Ready
TXPKTRDY (R/W1S)
Tx Packet Ready
RXSTALL (R/W0C)
Rx Stall
SETUPPKT (R/W1S)
Setup Packet
TOERR (R/W0C)
Timeout Error
REQPKT (R/W)
Request Packet

Figure 27-45: USB_EP0I_CSR[N]_H Register Diagram

Table 27-21: USB_EP0I_CSR[N]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DISPING	Disable Ping. The `USB_EP0I_CSR[N]_H.DISPING` bit disables (in host mode) high-speed PING tokens for the data and status phases of a control transfer.	
		0	Issue PING tokens
		1	Do not issue PING
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP0I_CSR[N]_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint 0 `USB_EP0I_CSR[N]_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL

Table 27-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	DATGL	Data Toggle. The `USB_EP0I_CSR[N]_H.DATGL` bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit should only be set if the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The `USB_EP0I_CSR[N]_H.NAKTO` bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the `USB_EP0_NAKLIMIT[n]` register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The `USB_EP0I_CSR[N]_H.STATUSPKT` bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction

Table 27-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

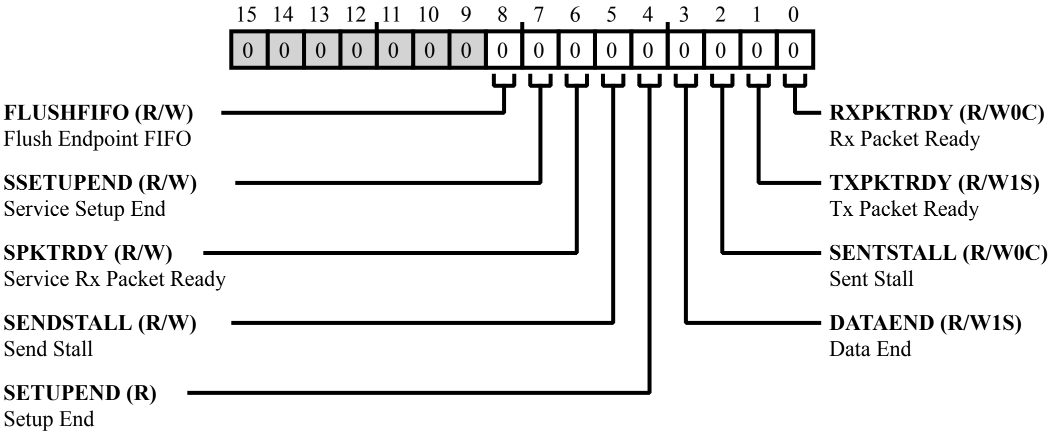
Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	REQPKT	Request Packet. The `USB_EP0I_CSR[N]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when the `USB_EP0I_CSR[N]_H.RXPKTRDY` bit is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0I_CSR[N]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0I_CSR[N]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0I_CSR[N]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

Table 27-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The `USB_EP0I_CSR[N]_P` register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SSETUPEND (R/W)
Service Setup End
SPKTRDY (R/W)
Service Rx Packet Ready
SENDSTALL (R/W)
Send Stall
SETUPEND (R)
Setup End
RXPKTRDY (R/W0C)
Rx Packet Ready
TXPKTRDY (R/W1S)
Tx Packet Ready
SENTSTALL (R/W0C)
Sent Stall
DATAEND (R/W1S)
Data End

Figure 27-46: USB_EP0I_CSR[N]_P Register Diagram

Table 27-22: USB_EP0I_CSR[N]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0I_CSR[N]_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0I_CSR[N]_P.TXPKTRDY` and `USB_EP0I_CSR[N]_P.RXPKTRDY` bits. The `USB_EP0I_CSR[N]_P.FLUSHFIFO` bit should only be set if the `USB_EP0I_CSR[N]_P.TXPKTRDY` and `USB_EP0I_CSR[N]_P.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The `USB_EP0I_CSR[N]_P.SSETUPEND` bit is set (in peripheral mode) by the processor core to clear the `USB_EP0I_CSR[N]_P.SETUPEND`. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 27-22: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The `USB_EP0I_CSR[N]_P.SPKTRDY` bit is set (in peripheral mode) by the processor core to clear the `USB_EP0I_CSR[N]_P.RXPKTRDY` bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The `USB_EP0I_CSR[N]_P.SENDSTALL` bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The `USB_EP0I_CSR[N]_P.SETUPEND` bit indicates (in peripheral mode) when a control transaction ends before the `USB_EP0I_CSR[N]_P.DATAEND` bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the `USB_EP0I_CSR[N]_P.SSETUPEND` bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The `USB_EP0I_CSR[N]_P.DATAEND` bit is set (in peripheral mode) by the processor core sets when the core: • Sets the `USB_EP0I_CSR[N]_P.TXPKTRDY` bit for the last data packet. • Clears the `USB_EP0I_CSR[N]_P.RXPKTRDY` bit after unloading the last data packet. • Sets the `USB_EP0I_CSR[N]_P.TXPKTRDY` bit for a zero-length data packet. The `USB_EP0I_CSR[N]_P.DATAEND` bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The `USB_EP0I_CSR[N]_P.SENTSTALL` bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 27-22: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0I_CSR[N]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The `USB_EP0I_NAKLIMIT[N]` register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

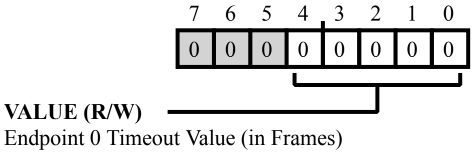


Figure 27-47: USB_EP0I_NAKLIMIT[N] Register Diagram

Table 27-23: USB_EP0I_NAKLIMIT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The `USB_EP0I_NAKLIMIT[N].VALUE` bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The `USB_EP0I_TYPE[N]` register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

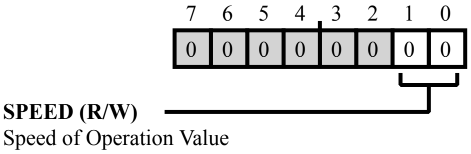


Figure 27-48: USB_EP0I_TYPE[N] Register Diagram

Table 27-24: USB_EP0I_TYPE[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The `USB_EP0I_TYPE[N].SPEED` bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed

EP0 Configuration Information Register

The `USB_EP0_CFGDATA[n]` register describes the USB controller hardware configuration. This register only exists for endpoint 0.

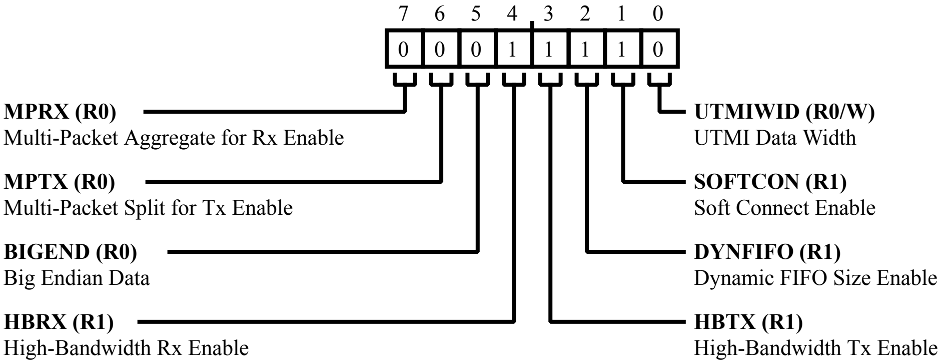

7 6 5 4 3 2 1 0
0 0 0 1 1 1 1 0
MPRX (R0)
Multi-Packet Aggregate for Rx Enable
MPTX (R0)
Multi-Packet Split for Tx Enable
BIGEND (R0)
Big Endian Data
HBRX (R1)
High-Bandwidth Rx Enable
UTMIWID (R0/W)
UTMI Data Width
SOFTCON (R1)
Soft Connect Enable
DYNFIFO (R1)
Dynamic FIFO Size Enable
HBTX (R1)
High-Bandwidth Tx Enable

Figure 27-49: USB_EP0_CFGDATA[n] Register Diagram

Table 27-25: USB_EP0_CFGDATA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The `USB_EP0_CFGDATA[n].MPRX` bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The `USB_EP0_CFGDATA[n].MPTX` bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The `USB_EP0_CFGDATA[n].BIGEND` bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 27-25: USB_EP0_CFGDATA[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R1/NW)	HBRX	High-Bandwidth Rx Enable. The `USB_EP0_CFGDATA[n].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint.	
		0	No High-Bandwidth Rx
		1	High-Bandwidth Rx
3 (R1/NW)	HBTX	High-Bandwidth Tx Enable. The `USB_EP0_CFGDATA[n].HBTX` bit indicates whether the USB controller supports high-bandwidth transmit ISO endpoint.	
		0	No High-Bandwidth Tx
		1	High-Bandwidth Tx
2 (R1/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0_CFGDATA[n].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0_CFGDATA[n].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0_CFGDATA[n].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The `USB_EP0_CNT[n]` register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change. It is only valid while the `USB_EP0_CSR[n]_H.RXPKTRDY` bit or `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set.

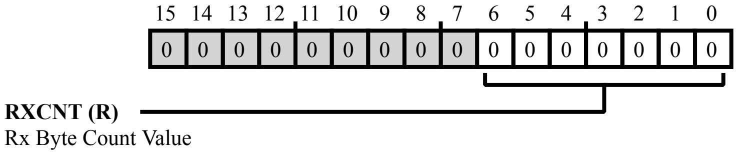


Figure 27-50: USB_EP0_CNT[n] Register Diagram

Table 27-26: USB_EP0_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The `USB_EP0_CNT[n].RXCNT` bits holds the number of data bytes currently in line ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded. It is only valid while the `USB_EP0_CSR[n]_H.RXPKTRDY` bit or `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set.

EP0 Configuration and Status (Host) Register

The `USB_EP0_CSR[n]_H` register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

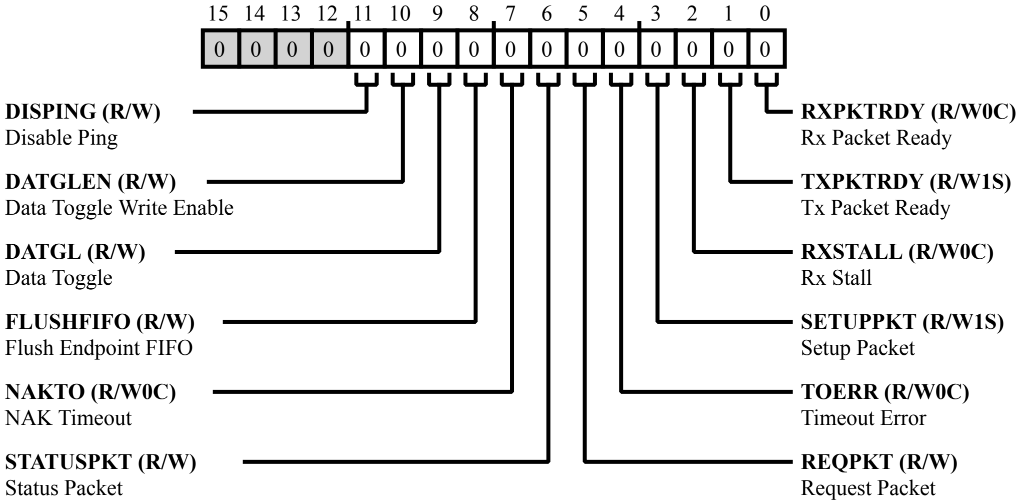

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DISPING (R/W)
Disable Ping
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
FLUSHFIFO (R/W)
Flush Endpoint FIFO
NAKTO (R/W0C)
NAK Timeout
STATUSPKT (R/W)
Status Packet
RXPKTRDY (R/W0C)
Rx Packet Ready
TXPKTRDY (R/W1S)
Tx Packet Ready
RXSTALL (R/W0C)
Rx Stall
SETUPPKT (R/W1S)
Setup Packet
TOERR (R/W0C)
Timeout Error
REQPKT (R/W)
Request Packet

Figure 27-51: USB_EP0_CSR[n]_H Register Diagram

Table 27-27: USB_EP0_CSR[n]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DISPING	Disable Ping. The `USB_EP0_CSR[n]_H.DISPING` bit disables (in host mode) high-speed PING tokens for the data and status phases of a control transfer.	
		0	Issue PING tokens
		1	Do not issue PING
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP0_CSR[n]_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint 0 `USB_EP0_CSR[n]_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL

Table 27-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	DATGL	Data Toggle. The `USB_EP0_CSR[n]_H.DATGL` bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0_CSR[n]_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0_CSR[n]_H.TXPKTRDY` and `USB_EP0_CSR[n]_H.RXPKTRDY` bits. The `USB_EP0_CSR[n]_H.FLUSHFIFO` bit should only be set if the `USB_EP0_CSR[n]_H.TXPKTRDY` and `USB_EP0_CSR[n]_H.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The `USB_EP0_CSR[n]_H.NAKTO` bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the `USB_EP0_NAKLIMIT[n]` register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The `USB_EP0_CSR[n]_H.STATUSPKT` bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the `USB_EP0_CSR[n]_H.TXPKTRDY` and `USB_EP0_CSR[n]_H.RXPKTRDY` bits. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction

Table 27-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	REQPKT	Request Packet. The `USB_EP0_CSR[n]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when the `USB_EP0_CSR[n]_H.RXPKTRDY` bit is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0_CSR[n]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0_CSR[n]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0_CSR[n]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0_CSR[n]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

Table 27-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The `USB_EP0_CSR[n]_P` register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

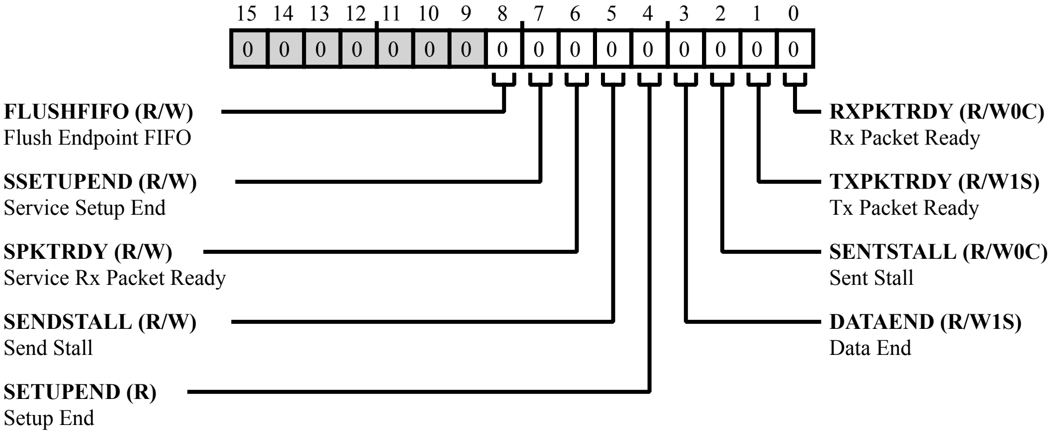

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SSETUPEND (R/W)
Service Setup End
SPKTRDY (R/W)
Service Rx Packet Ready
SENDSTALL (R/W)
Send Stall
SETUPEND (R)
Setup End
RXPKTRDY (R/W0C)
Rx Packet Ready
TXPKTRDY (R/W1S)
Tx Packet Ready
SENTSTALL (R/W0C)
Sent Stall
DATAEND (R/W1S)
Data End

Figure 27-52: USB_EP0_CSR[n]_P Register Diagram

Table 27-28: USB_EP0_CSR[n]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0_CSR[n]_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0_CSR[n]_P.TXPKTRDY` and `USB_EP0_CSR[n]_P.RXPKTRDY` bits. The `USB_EP0_CSR[n]_P.FLUSHFIFO` bit should only be set if the `USB_EP0_CSR[n]_P.TXPKTRDY` and `USB_EP0_CSR[n]_P.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The `USB_EP0_CSR[n]_P.SSETUPEND` bit is set (in peripheral mode) by the processor core to clear the `USB_EP0_CSR[n]_P.SETUPEND`. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 27-28: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The `USB_EP0_CSR[n]_P.SPKTRDY` bit is set (in peripheral mode) by the processor core to clear the `USB_EP0_CSR[n]_P.RXPKTRDY` bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The `USB_EP0_CSR[n]_P.SENDSTALL` bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The `USB_EP0_CSR[n]_P.SETUPEND` bit indicates (in peripheral mode) when a control transaction ends before the `USB_EP0_CSR[n]_P.DATAEND` bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the `USB_EP0_CSR[n]_P.SSETUPEND` bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The `USB_EP0_CSR[n]_P.DATAEND` bit is set (in peripheral mode) by the processor core sets when the core: • Sets the `USB_EP0_CSR[n]_P.TXPKTRDY` bit for the last data packet. • Clears the `USB_EP0_CSR[n]_P.RXPKTRDY` bit after unloading the last data packet. • Sets the `USB_EP0_CSR[n]_P.TXPKTRDY` bit for a zero-length data packet. The `USB_EP0_CSR[n]_P.DATAEND` bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The `USB_EP0_CSR[n]_P.SENTSTALL` bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 27-28: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The `USB_EP0_NAKLIMIT[n]` register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

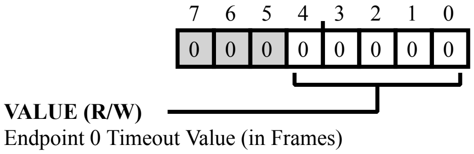


Figure 27-53: USB_EP0_NAKLIMIT[n] Register Diagram

Table 27-29: USB_EP0_NAKLIMIT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The `USB_EP0_NAKLIMIT[n].VALUE` bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The `USB_EP0_TYPE[n]` register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

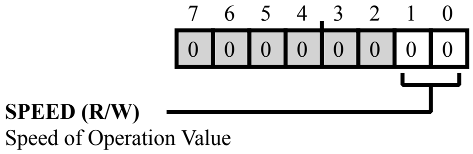


Figure 27-54: USB_EP0_TYPE[n] Register Diagram

Table 27-30: USB_EP0_TYPE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation. The `USB_EP0_TYPE[n].SPEED` bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed

Endpoint Information Register

The `USB_EPINFO` register allows read-back of the number of Tx and Rx endpoints available

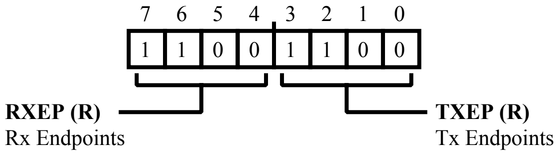


Figure 27-55: USB_EPINFO Register Diagram

Table 27-31: USB_EPINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXEP	Rx Endpoints. The `USB_EPINFO.RXEP` bits indicate the number of receive endpoints. excluding EP0.
3:0 (R/NW)	TXEP	Tx Endpoints. The `USB_EPINFO.TXEP` bits indicate the number of transmit endpoints, excluding EP0.

EPn Number of Bytes Received Register

The `USB_EPI[N]_RXCNT` register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit or `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is set.

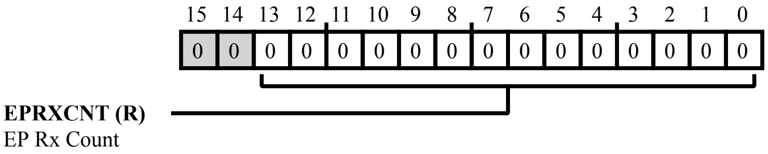


Figure 27-56: USB_EPI[N]_RXCNT Register Diagram

Table 27-32: USB_EPI[N]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The `USB_EPI[N]_RXCNT.EPRXCNT` bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The `USB_EPI[N]_RXCSR_H` register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

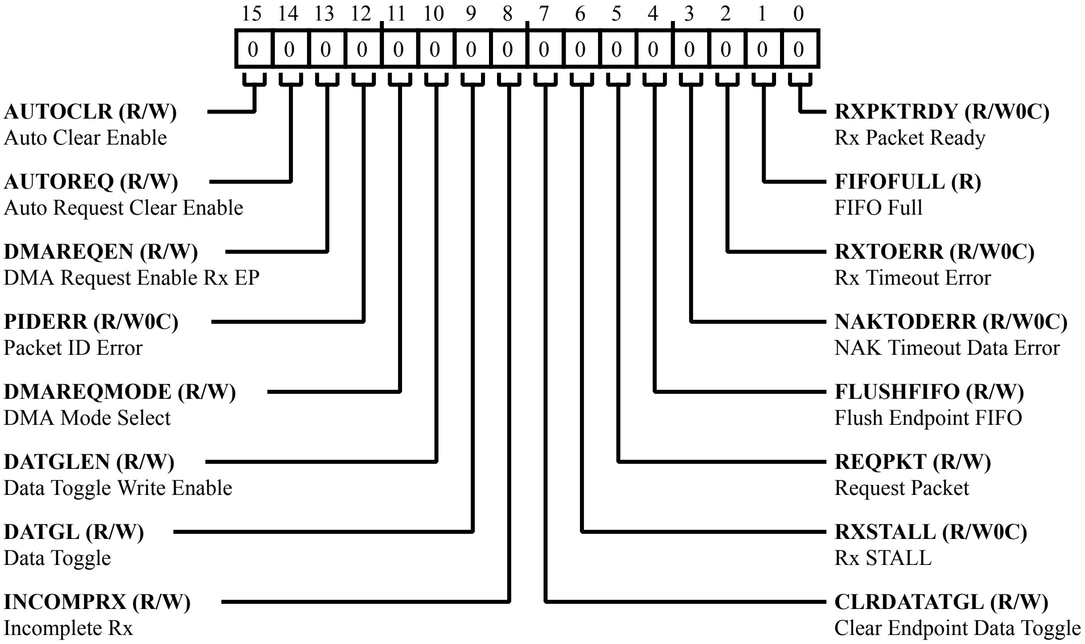

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOCLR (R/W)
Auto Clear Enable
AUTOREQ (R/W)
Auto Request Clear Enable
DMAREQEN (R/W)
DMA Request Enable Rx EP
PIDERR (R/W0C)
Packet ID Error
DMAREQMODE (R/W)
DMA Mode Select
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
INCOMPRX (R/W)
Incomplete Rx
RXPKTRDY (R/W0C)
Rx Packet Ready
FIFOFULL (R)
FIFO Full
RXTOERR (R/W0C)
Rx Timeout Error
NAKTODERR (R/W0C)
NAK Timeout Data Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
REQPKT (R/W)
Request Packet
RXSTALL (R/W0C)
Rx STALL
CLRDATATGL (R/W)
Clear Endpoint Data Toggle

Figure 27-57: USB_EPI[N]_RXCSR_H Register Diagram

Table 27-33: USB_EPI[N]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The `USB_EPI[N]_RXCSR_H.AUTOCLR` bit directs (in host mode) the USB controller to automatically clear the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit when a packet of size `USB_EP[n]_RXMAXP` bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear `USB_EPI[N]_RXCSR_H.RXPKTRDY` manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the `USB_EP[n]_RXMAXP` value. The USB controller auto clears the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the `USB_EPI[N]_RXCSR_H.AUTOCLR` bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The `USB_EPI[N]_RXCSR_H.AUTOREQ` bit directs (in host mode) the USB controller to automatically clear the `USB_EPI[N]_RXCSR_H.REQPKT` bit when `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The `USB_EPI[N]_RXCSR_H.DMAREQEN` bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The `USB_EPI[N]_RXCSR_H.PIDERR` bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_RXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_RXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EPI[N]_RXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_RXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The `USB_EPI[N]_RXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EPI[N]_RXCSR_H.INCOMPRX` bit indicates (in host mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EPI[N]_RXCSR_H.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx

Table 27-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	RXSTALL	Rx STALL. The `USB_EPI[N]_RXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The `USB_EPI[N]_RXCSR_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EPI[N]_RXCSR_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_RXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit. The `USB_EPI[N]_RXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 27-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The `USB_EPI[N]_RXCSR_H.NAKTODERR` bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit is cleared. The `USB_EPI[N]_RXCSR_H.NAKTODERR` bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the `USB_EP[n]_RXINTERVAL` register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the `USB_EPI[N]_RXCSR_H.REQPKT` bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error
2 (R/W0C)	RXTOERR	Rx Timeout Error. The `USB_EPI[N]_RXCSR_H.RXTOERR` bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that `USB_EPI[N]_RXCSR_H.RXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EPI[N]_RXCSR_H.FIFOFULL` bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EPI[N]_RXCSR_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The `USB_EPI[N]_RXCSR_P` register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

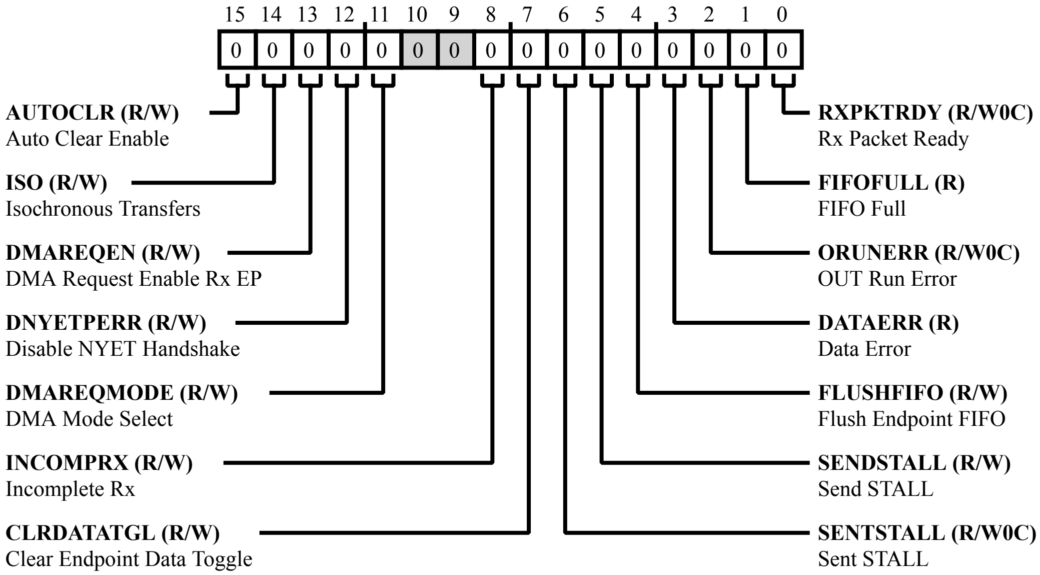

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOCLR (R/W)
Auto Clear Enable
ISO (R/W)
Isochronous Transfers
DMAREQEN (R/W)
DMA Request Enable Rx EP
DNYETPERR (R/W)
Disable NYET Handshake
DMAREQMODE (R/W)
DMA Mode Select
INCOMPRX (R/W)
Incomplete Rx
CLRDATATGL (R/W)
Clear Endpoint Data Toggle
RXPKTRDY (R/W0C)
Rx Packet Ready
FIFOFULL (R)
FIFO Full
ORUNERR (R/W0C)
OUT Run Error
DATAERR (R)
Data Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SENDSTALL (R/W)
Send STALL
SENTSTALL (R/W0C)
Sent STALL

Figure 27-58: USB_EPI[N]_RXCSR_P Register Diagram

Table 27-34: USB_EPI[N]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The `USB_EPI[N]_RXCSR_P.AUTOCLR` bit directs (in peripheral mode) the USB controller to automatically clear the `USB_EPI[N]_RXCSR_P.RXPKTRDY` bit when a packet of size `USB_EP[n]_RXMAXP` bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear `USB_EPI[N]_RXCSR_P.RXPKTRDY` manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the `USB_EP[n]_RXMAXP` value. The USB controller auto clears the `USB_EPI[N]_RXCSR_P.RXPKTRDY` bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the `USB_EPI[N]_RXCSR_P.AUTOCLR` bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The `USB_EPI[N]_RXCSR_P.ISO` bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The `USB_EPI[N]_RXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DNYETPERR	Disable NYET Handshake. The `USB_EPI[N]_RXCSR_P.DNYETPERR` bit disables (in peripheral mode for high speed bulk/interrupt transactions) NYET handshakes. When this bit is set, all successful receive packets are ACK'd, including the point at which the FIFO becomes full. The `USB_EPI[N]_RXCSR_P.DNYETPERR` bit must be set for all interrupt endpoints in high speed mode.	
		0	Enable NYET Handshake
		1	Disable NYET Handshake
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_RXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_RXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EPI[N]_RXCSR_P.INCOMPRX` bit indicates (in peripheral mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EPI[N]_RXCSR_P.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_RXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EPI[N]_RXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 27-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SENDSTALL	Send STALL. The `USB_EPI[N]_RXCSR_P.SENDSTALL` bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_RXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_RXCSR_P.RXPKTRDY` bit. The `USB_EPI[N]_RXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_RXCSR_P.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/NW)	DATAERR	Data Error. The `USB_EPI[N]_RXCSR_P.DATAERR` bit indicates (in peripheral mode for isochronous transfers) when the `USB_EPI[N]_RXCSR_P.RXPKTRDY` bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when `USB_EPI[N]_RXCSR_P.RXPKTRDY` is cleared. The `USB_EPI[N]_RXCSR_P.DATAERR` bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The `USB_EPI[N]_RXCSR_P.ORUNERR` bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The `USB_EPI[N]_RXCSR_P.ORUNERR` bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EPI[N]_RXCSR_P.FIFOFULL` bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full

Table 27-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EPI[N]_RXCSR_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The `USB_EPI[N]_RXINTERVAL` register defines the polling interval for the currently-selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a `USB_EPI[N]_RXINTERVAL` register for each configured receive endpoint, except endpoint 0. The transfer types related to speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full -speed, `USB_EPI[N]_RXINTERVAL` = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, `USB_EPI[N]_RXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, `USB_EPI[N]_RXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, `USB_EPI[N]_RXINTERVAL` = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a `USB_EPI[N]_RXINTERVAL` value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

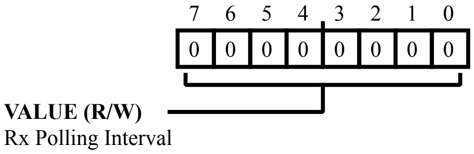


Figure 27-59: USB_EPI[N]_RXINTERVAL Register Diagram

Table 27-35: USB_EPI[N]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The `USB_EPI[N]_RXINTERVAL.VALUE` bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The USB_EPI[N]_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also, note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

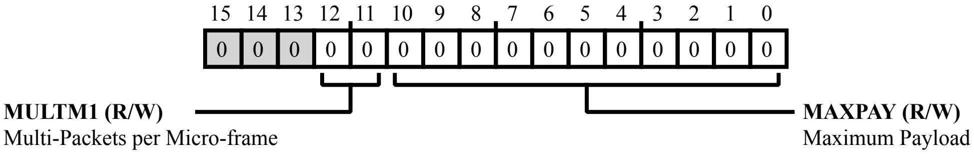


Figure 27-60: USB_EPI[N]_RXMAXP Register Diagram

Table 27-36: USB_EPI[N]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The USB_EPI[N]_RXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller combines multiple packets received within a micro-frame into a single packet in the FIFO.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024, but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The `USB_EPI[N]_RXTYPE` register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a `USB_EPI[N]_RXTYPE` register for each receive endpoint. Note that this register is only used in host mode.

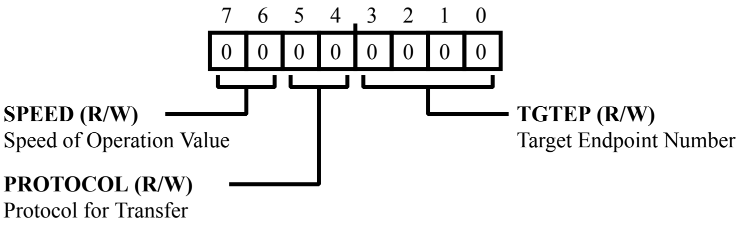


Figure 27-61: USB_EPI[N]_RXTYPE Register Diagram

Table 27-37: USB_EPI[N]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The `USB_EPI[N]_RXTYPE.SPEED` bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When it is not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The `USB_EPI[N]_RXTYPE.PROTOCOL` bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The `USB_EPI[N]_RXTYPE.TGTEP` bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 27-37: USB_EPI[N]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The USB_EPI[N]_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently-selected transmit endpoint.

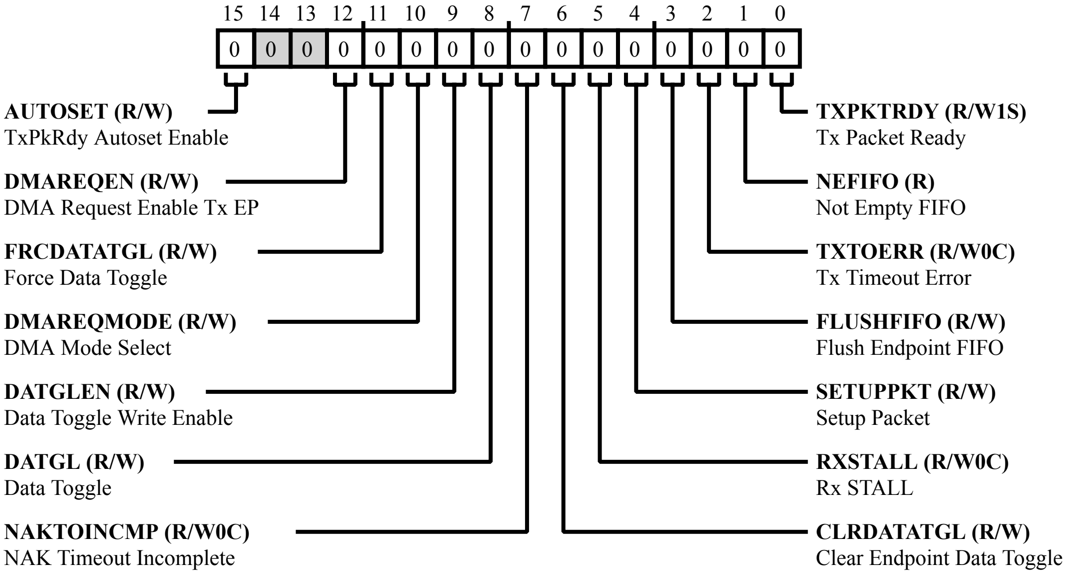

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOSET (R/W)
TxPkRdy Autoset Enable
DMAREQEN (R/W)
DMA Request Enable Tx EP
FRCDATATGL (R/W)
Force Data Toggle
DMAREQMODE (R/W)
DMA Mode Select
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
NAKTOINCMP (R/W0C)
NAK Timeout Incomplete
TXPKTRDY (R/W1S)
Tx Packet Ready
NEFIFO (R)
Not Empty FIFO
TXTOERR (R/W0C)
Tx Timeout Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SETUPPKT (R/W)
Setup Packet
RXSTALL (R/W0C)
Rx STALL
CLRDATATGL (R/W)
Clear Endpoint Data Toggle

Figure 27-62: USB_EPI[N]_TXCSR_H Register Diagram

Table 27-38: USB_EPI[N]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_H.AUTOSET bit enables (in host mode) the automatic setting of the USB_EPI[N]_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_H.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_H.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EPI[N]_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">11
(R/W)</td><td rowspan="3">FRCDATATGL</td><td colspan="2">Force Data Toggle.
The `USB_EPI[N]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints to communicate rate feedback for isochronous endpoints.</td></tr>
<tr><td>0</td><td>No Action</td></tr>
<tr><td>1</td><td>Toggle Endpoint Data</td></tr>
<tr><td rowspan="3">10
(R/W)</td><td rowspan="3">DMAREQMODE</td><td colspan="2">DMA Mode Select.
The `USB_EPI[N]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or the same cycle that the `USB_EPI[N]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.</td></tr>
<tr><td>0</td><td>DMA Request Mode 0</td></tr>
<tr><td>1</td><td>DMA Request Mode 1</td></tr>
<tr><td rowspan="3">9
(R/W)</td><td rowspan="3">DATGLEN</td><td colspan="2">Data Toggle Write Enable.
The `USB_EPI[N]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.</td></tr>
<tr><td>0</td><td>Disable Write to DATGL</td></tr>
<tr><td>1</td><td>Enable Write to DATGL</td></tr>
<tr><td rowspan="3">8
(R/W)</td><td rowspan="3">DATGL</td><td colspan="2">Data Toggle.
The `USB_EPI[N]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.</td></tr>
<tr><td>0</td><td>DATA0 is set</td></tr>
<tr><td>1</td><td>DATA1 is set</td></tr>
</table>

Table 27-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The `USB_EPI[N]_TXCSR_H.NAKTOINCMP` bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the `USB_EP[n]_TXINTERVAL` register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_TXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The `USB_EPI[N]_TXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The `USB_EPI[N]_TXCSR_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_TXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit. The `USB_EPI[N]_TXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 27-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EPI[N]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EPI[N]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EPI[N]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EPI[N]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

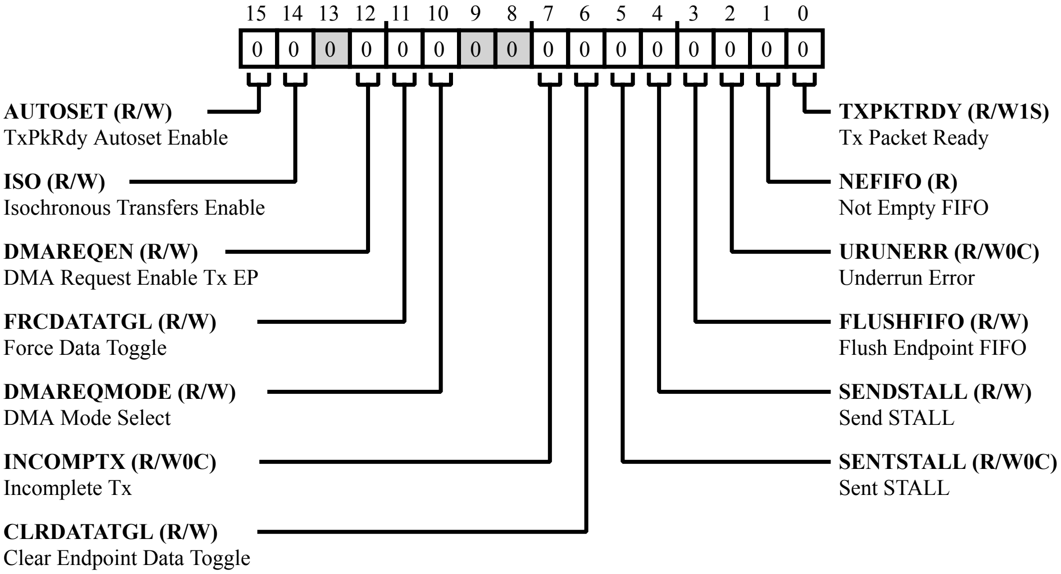

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOSET (R/W)
TxPkRdy Autoset Enable
ISO (R/W)
Isochronous Transfers Enable
DMAREQEN (R/W)
DMA Request Enable Tx EP
FRCDATATGL (R/W)
Force Data Toggle
DMAREQMODE (R/W)
DMA Mode Select
INCOMPTX (R/W0C)
Incomplete Tx
CLRDATATGL (R/W)
Clear Endpoint Data Toggle
TXPKTRDY (R/W1S)
Tx Packet Ready
NEFIFO (R)
Not Empty FIFO
URUNERR (R/W0C)
Underrun Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SENDSTALL (R/W)
Send STALL
SENTSTALL (R/W0C)
Sent STALL

Figure 27-63: USB_EPI[N]_TXCSR_P Register Diagram

Table 27-39: USB_EPI[N]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EPI[N]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_P.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset

Table 27-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ISO	Isochronous Transfers Enable. The `USB_EPI[N]_TXCSR_P.ISO` bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EPI[N]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EPI[N]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or in the same cycle that the `USB_EPI[N]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W0C)	INCOMPTX	Incomplete Tx. The `USB_EPI[N]_TXCSR_P.INCOMPTX` bit indicates (for high-bandwidth isochronous endpoints in peripheral mode) when a large packet has been split into two or three packets for transmission, but insufficient IN tokens have been received to send all parts.	
		0	No Status
		1	Incomplete Tx (Insufficient IN Tokens)

Table 27-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EPI[N]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
4 (R/W)	SENDSTALL	Send STALL. The `USB_EPI[N]_TXCSR_P.SENDSTALL` bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_TXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit. The `USB_EPI[N]_TXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The `USB_EPI[N]_TXCSR_P.URUNERR` bit indicates (in peripheral mode) when an IN token is received while the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error

Table 27-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EPI[N]_TXCSR_P.NEFIFO` bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The `USB_EPI[N]_TXINTERVAL` register defines the polling interval for the currently-selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a `USB_EPI[N]_TXINTERVAL` register for each configured transmit endpoint, except endpoint 0. The transfer types related to the speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full-speed, `USB_EPI[N]_TXINTERVAL` = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, `USB_EPI[N]_TXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, `USB_EPI[N]_TXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, `USB_EPI[N]_TXINTERVAL` = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a `USB_EPI[N]_TXINTERVAL` value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

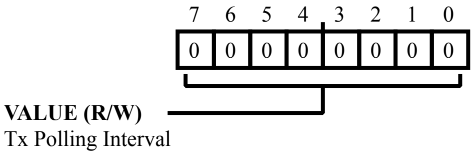


Figure 27-64: USB_EPI[N]_TXINTERVAL Register Diagram

Table 27-40: USB_EPI[N]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The `USB_EPI[N]_TXINTERVAL.VALUE` bits define the polling interval value for interrupt and isochronous transfers. The `USB_EPI[N]_TXINTERVAL.VALUE` bits select the number of frames (or micro-frames, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EPI[N]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EPI[N]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

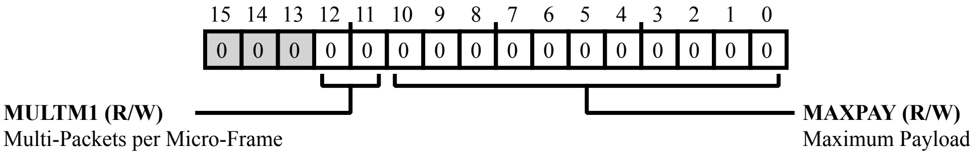


Figure 27-65: USB_EPI[N]_TXMAXP Register Diagram

Table 27-41: USB_EPI[N]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-Frame. The USB_EPI[N]_TXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller splits the FIFO data into multiple packets less than or equal to the maximum payload size.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The `USB_EPI[N]_TXTYPE` register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a `USB_EPI[N]_TXTYPE` register for each transmit endpoint. Note that this register is only used in host mode.

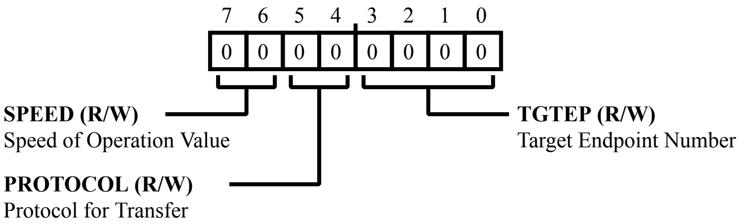


Figure 27-66: USB_EPI[N]_TXTYPE Register Diagram

Table 27-42: USB_EPI[N]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The `USB_EPI[N]_TXTYPE.SPEED` bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The `USB_EPI[N]_TXTYPE.PROTOCOL` bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The `USB_EPI[N]_TXTYPE.TGTEP` bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 27-42: USB_EPI[N]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Number of Bytes Received Register

The `USB_EP[n]_RXCNT` register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit or `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is set.

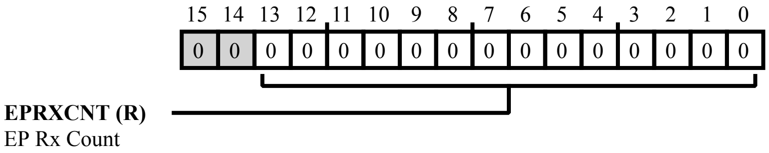


Figure 27-67: USB_EP[n]_RXCNT Register Diagram

Table 27-43: USB_EP[n]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The `USB_EP[n]_RXCNT.EPRXCNT` bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The `USB_EP[n]_RXCSR_H` register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

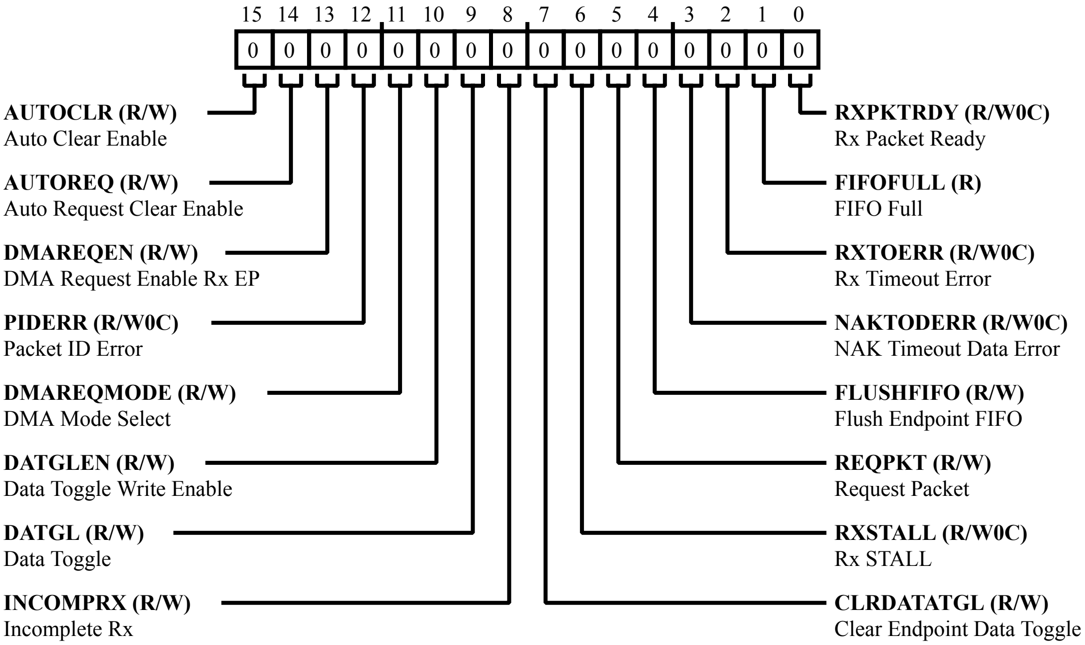

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOCLR (R/W)
Auto Clear Enable
AUTOREQ (R/W)
Auto Request Clear Enable
DMAREQEN (R/W)
DMA Request Enable Rx EP
PIDERR (R/W0C)
Packet ID Error
DMAREQMODE (R/W)
DMA Mode Select
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
INCOMPRX (R/W)
Incomplete Rx
RXPKTRDY (R/W0C)
Rx Packet Ready
FIFOFULL (R)
FIFO Full
RXTOERR (R/W0C)
Rx Timeout Error
NAKTODERR (R/W0C)
NAK Timeout Data Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
REQPKT (R/W)
Request Packet
RXSTALL (R/W0C)
Rx STALL
CLRDATATGL (R/W)
Clear Endpoint Data Toggle

Figure 27-68: USB_EP[n]_RXCSR_H Register Diagram

Table 27-44: USB_EP[n]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The `USB_EP[n]_RXCSR_H.AUTOCLR` bit directs (in host mode) the USB controller to automatically clear the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit when a packet of size `USB_EP[n]_RXMAXP` bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear `USB_EP[n]_RXCSR_H.RXPKTRDY` manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the `USB_EP[n]_RXMAXP` value. The USB controller auto clears the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the `USB_EP[n]_RXCSR_H.AUTOCLR` bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The `USB_EP[n]_RXCSR_H.AUTOREQ` bit directs (in host mode) the USB controller to automatically clear the `USB_EP[n]_RXCSR_H.REQPKT` bit when `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The `USB_EP[n]_RXCSR_H.DMAREQEN` bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The `USB_EP[n]_RXCSR_H.PIDERR` bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_RXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EP[n]_RXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP[n]_RXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EP[n]_RXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The `USB_EP[n]_RXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EP[n]_RXCSR_H.INCOMPRX` bit indicates (in host mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EP[n]_RXCSR_H.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx

Table 27-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	RXSTALL	Rx STALL. The `USB_EP[n]_RXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The `USB_EP[n]_RXCSR_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EP[n]_RXCSR_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_RXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit. The `USB_EP[n]_RXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 27-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The `USB_EP[n]_RXCSR_H.NAKTODERR` bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is cleared. The `USB_EP[n]_RXCSR_H.NAKTODERR` bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the `USB_EP[n]_RXINTERVAL` register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the `USB_EP[n]_RXCSR_H.REQPKT` bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error
2 (R/W0C)	RXTOERR	Rx Timeout Error. The `USB_EP[n]_RXCSR_H.RXTOERR` bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that `USB_EP[n]_RXCSR_H.RXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EP[n]_RXCSR_H.FIFOFULL` bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP[n]_RXCSR_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The `USB_EP[n]_RXCSR_P` register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

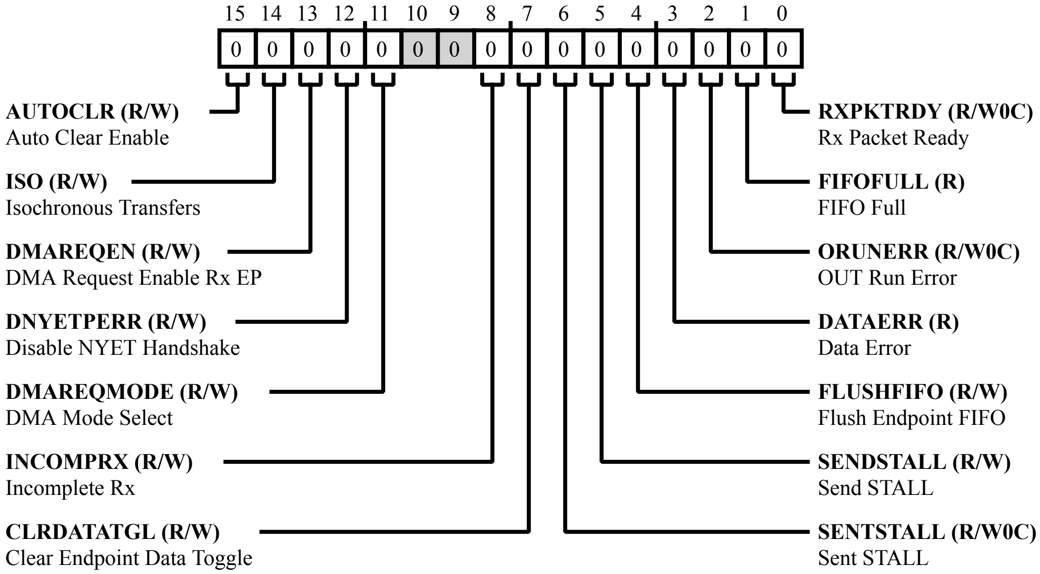

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOCLR (R/W)
Auto Clear Enable
ISO (R/W)
Isochronous Transfers
DMAREQEN (R/W)
DMA Request Enable Rx EP
DNYETPERR (R/W)
Disable NYET Handshake
DMAREQMODE (R/W)
DMA Mode Select
INCOMPRX (R/W)
Incomplete Rx
CLRDATATGL (R/W)
Clear Endpoint Data Toggle
RXPKTRDY (R/W0C)
Rx Packet Ready
FIFOFULL (R)
FIFO Full
ORUNERR (R/W0C)
OUT Run Error
DATAERR (R)
Data Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SENDSTALL (R/W)
Send STALL
SENTSTALL (R/W0C)
Sent STALL

Figure 27-69: USB_EP[n]_RXCSR_P Register Diagram

Table 27-45: USB_EP[n]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The `USB_EP[n]_RXCSR_P.AUTOCLR` bit directs (in peripheral mode) the USB controller to automatically clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit when a packet of size `USB_EP[n]_RXMAXP` bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear `USB_EP[n]_RXCSR_P.RXPKTRDY` manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the `USB_EP[n]_RXMAXP` value. The USB controller auto clears the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the `USB_EP[n]_RXCSR_P.AUTOCLR` bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The `USB_EP[n]_RXCSR_P.ISO` bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The `USB_EP[n]_RXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DNYETPERR	Disable NYET Handshake. The `USB_EP[n]_RXCSR_P.DNYETPERR` bit disables (in peripheral mode for high speed bulk/interrupt transactions) NYET handshakes. When this bit is set, all successful receive packets are ACK'd, including the point at which the FIFO becomes full. The `USB_EP[n]_RXCSR_P.DNYETPERR` bit must be set for all interrupt endpoints in high speed mode.	
		0	Enable NYET Handshake
		1	Disable NYET Handshake
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_RXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EP[n]_RXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EP[n]_RXCSR_P.INCOMPRX` bit indicates (in peripheral mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EP[n]_RXCSR_P.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_RXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EP[n]_RXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 27-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SENDSTALL	Send STALL. The `USB_EP[n]_RXCSR_P.SENDSTALL` bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_RXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit. The `USB_EP[n]_RXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/NW)	DATAERR	Data Error. The `USB_EP[n]_RXCSR_P.DATAERR` bit indicates (in peripheral mode for isochronous transfers) when the `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when `USB_EP[n]_RXCSR_P.RXPKTRDY` is cleared. The `USB_EP[n]_RXCSR_P.DATAERR` bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The `USB_EP[n]_RXCSR_P.ORUNERR` bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The `USB_EP[n]_RXCSR_P.ORUNERR` bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EP[n]_RXCSR_P.FIFOFULL` bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full

Table 27-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP[n]_RXCSR_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The `USB_EP[n]_RXINTERVAL` register defines the polling interval for the currently-selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a `USB_EP[n]_RXINTERVAL` register for each configured receive endpoint, except endpoint 0. The transfer types related to speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full -speed, `USB_EP[n]_RXINTERVAL` = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, `USB_EP[n]_RXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, `USB_EP[n]_RXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, `USB_EP[n]_RXINTERVAL` = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a `USB_EP[n]_RXINTERVAL` value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

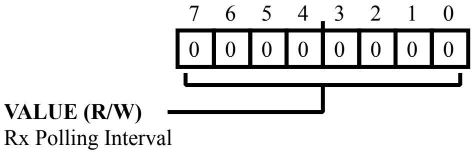


Figure 27-70: USB_EP[n]_RXINTERVAL Register Diagram

Table 27-46: USB_EP[n]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The `USB_EP[n]_RXINTERVAL.VALUE` bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The `USB_EP[n]_RXMAXP` register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also, note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

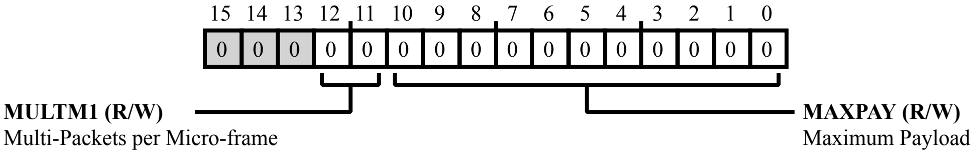


Figure 27-71: USB_EP[n]_RXMAXP Register Diagram

Table 27-47: USB_EP[n]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The `USB_EP[n]_RXMAXP.MULTM1` bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller combines multiple packets received within a micro-frame into a single packet in the FIFO.
10:0 (R/W)	MAXPAY	Maximum Payload. The `USB_EP[n]_RXMAXP.MAXPAY` bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024, but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The `USB_EP[n]_RXMAXP.MAXPAY` bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The `USB_EP[n]_RXTYPE` register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a `USB_EP[n]_RXTYPE` register for each receive endpoint. Note that this register is only used in host mode.

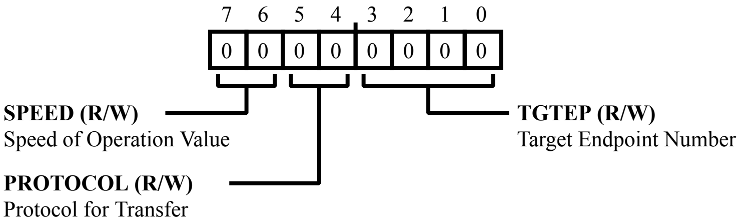


Figure 27-72: USB_EP[n]_RXTYPE Register Diagram

Table 27-48: USB_EP[n]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The `USB_EP[n]_RXTYPE.SPEED` bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When it is not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The `USB_EP[n]_RXTYPE.PROTOCOL` bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The `USB_EP[n]_RXTYPE.TGTEP` bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 27-48: USB_EP[n]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The `USB_EP[n]_TXCSR_H` register provides (in host mode) control and status bits for transfers through the currently-selected transmit endpoint.

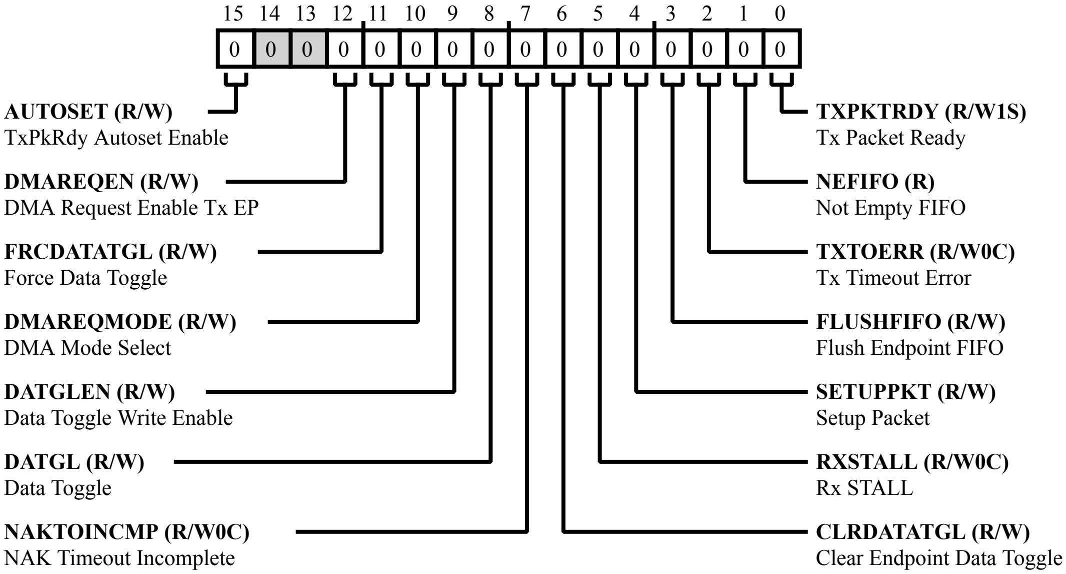

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOSET (R/W)
TxPkRdy Autoset Enable
DMAREQEN (R/W)
DMA Request Enable Tx EP
FRCDATATGL (R/W)
Force Data Toggle
DMAREQMODE (R/W)
DMA Mode Select
DATGLEN (R/W)
Data Toggle Write Enable
DATGL (R/W)
Data Toggle
NAKTOINCMP (R/W0C)
NAK Timeout Incomplete
TXPKTRDY (R/W1S)
Tx Packet Ready
NEFIFO (R)
Not Empty FIFO
TXTOERR (R/W0C)
Tx Timeout Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SETUPPKT (R/W)
Setup Packet
RXSTALL (R/W0C)
Rx STALL
CLRDATATGL (R/W)
Clear Endpoint Data Toggle

Figure 27-73: USB_EP[n]_TXCSR_H Register Diagram

Table 27-49: USB_EP[n]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The `USB_EP[n]_TXCSR_H.AUTOSET` bit enables (in host mode) the automatic setting of the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit when the maximum data packet size (`USB_EP[n]_TXMAXP`) is loaded into the transmit FIFO. The `USB_EP[n]_TXMAXP` value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit needs to be set manually. For products supporting high-speed operation, this `USB_EP[n]_TXCSR_H.AUTOSET` bit should not be set for high-bandwidth endpoints (endpoints with `USB_EP[n]_TXMAXP` value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EP[n]_TXCSR_H.DMAREQEN` bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 27-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or the same cycle that the `USB_EP[n]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP[n]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EP[n]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
8 (R/W)	DATGL	Data Toggle. The `USB_EP[n]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set

Table 27-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The `USB_EP[n]_TXCSR_H.NAKTOINCMP` bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the `USB_EP[n]_TXINTERVAL` register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The `USB_EP[n]_TXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The `USB_EP[n]_TXCSR_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 27-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EP[n]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EP[n]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EP[n]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP[n]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EP[n]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

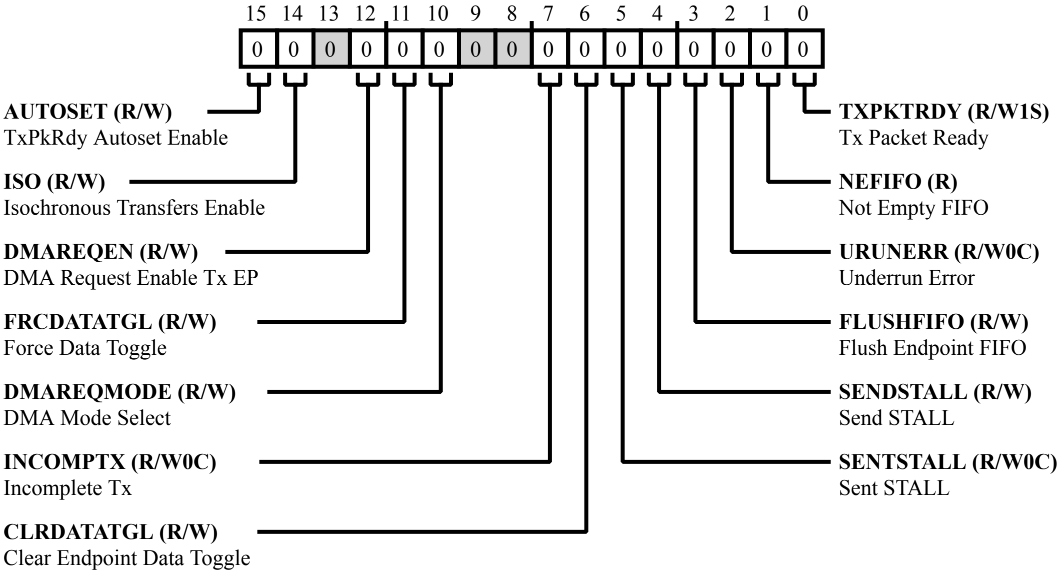

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AUTOSET (R/W)
TxPkRdy Autoset Enable
ISO (R/W)
Isochronous Transfers Enable
DMAREQEN (R/W)
DMA Request Enable Tx EP
FRCDATATGL (R/W)
Force Data Toggle
DMAREQMODE (R/W)
DMA Mode Select
INCOMPTX (R/W0C)
Incomplete Tx
CLRDATATGL (R/W)
Clear Endpoint Data Toggle
TXPKTRDY (R/W1S)
Tx Packet Ready
NEFIFO (R)
Not Empty FIFO
URUNERR (R/W0C)
Underrun Error
FLUSHFIFO (R/W)
Flush Endpoint FIFO
SENDSTALL (R/W)
Send STALL
SENTSTALL (R/W0C)
Sent STALL

Figure 27-74: USB_EP[n]_TXCSR_P Register Diagram

Table 27-50: USB_EP[n]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EP[n]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EP[n]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EP[n]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EP[n]_TXCSR_P.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset

Table 27-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ISO	Isochronous Transfers Enable. The `USB_EP[n]_TXCSR_P.ISO` bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EP[n]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or in the same cycle that the `USB_EP[n]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W0C)	INCOMPTX	Incomplete Tx. The `USB_EP[n]_TXCSR_P.INCOMPTX` bit indicates (for high-bandwidth isochronous endpoints in peripheral mode) when a large packet has been split into two or three packets for transmission, but insufficient IN tokens have been received to send all parts.	
		0	No Status
		1	Incomplete Tx (Insufficient IN Tokens)

Table 27-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EP[n]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
4 (R/W)	SENDSTALL	Send STALL. The `USB_EP[n]_TXCSR_P.SENDSTALL` bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_TXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit. The `USB_EP[n]_TXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The `USB_EP[n]_TXCSR_P.URUNERR` bit indicates (in peripheral mode) when an IN token is received while the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error

Table 27-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EP[n]_TXCSR_P.NEFIFO` bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The `USB_EP[n]_TXINTERVAL` register defines the polling interval for the currently-selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a `USB_EP[n]_TXINTERVAL` register for each configured transmit endpoint, except endpoint 0. The transfer types related to the speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full-speed, `USB_EP[n]_TXINTERVAL` = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, `USB_EP[n]_TXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, `USB_EP[n]_TXINTERVAL` = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, `USB_EP[n]_TXINTERVAL` = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a `USB_EP[n]_TXINTERVAL` value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

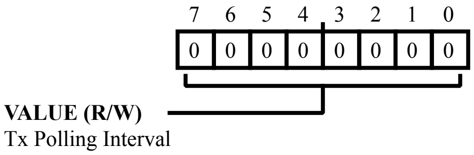


Figure 27-75: USB_EP[n]_TXINTERVAL Register Diagram

Table 27-51: USB_EP[n]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The `USB_EP[n]_TXINTERVAL.VALUE` bits define the polling interval value for interrupt and isochronous transfers. The `USB_EP[n]_TXINTERVAL.VALUE` bits select the number of frames (or micro-frames, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EP[n]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EP[n]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

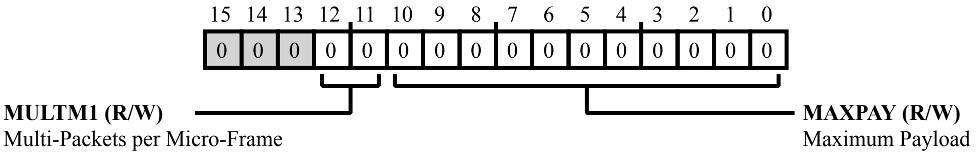


Figure 27-76: USB_EP[n]_TXMAXP Register Diagram

Table 27-52: USB_EP[n]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-Frame. The USB_EP[n]_TXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller splits the FIFO data into multiple packets less than or equal to the maximum payload size.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EP[n]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EP[n]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The `USB_EP[n]_TXTYPE` register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a `USB_EP[n]_TXTYPE` register for each transmit endpoint. Note that this register is only used in host mode.

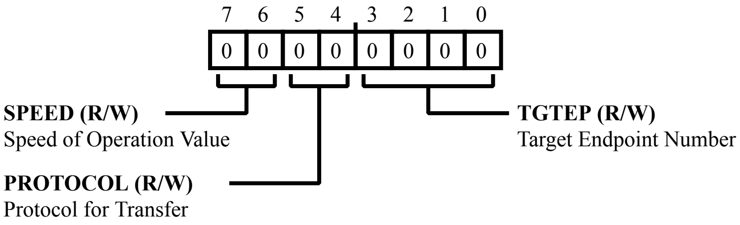


Figure 27-77: USB_EP[n]_TXTYPE Register Diagram

Table 27-53: USB_EP[n]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The `USB_EP[n]_TXTYPE.SPEED` bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The `USB_EP[n]_TXTYPE.PROTOCOL` bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The `USB_EP[n]_TXTYPE.TGTEP` bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 27-53: USB_EP[n]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

Function Address Register

The `USB_FADDR` register contains the device address used in peripheral mode. The processor writes this register with the address received through a SET_ADDRESS command from the host.

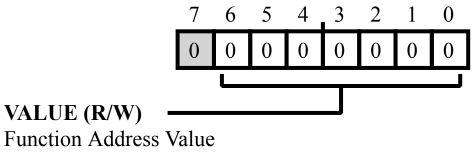


Figure 27-78: USB_FADDR Register Diagram

Table 27-54: USB_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The `USB_FADDR.VALUE` bits contain the address of the peripheral part of the transaction.

FIFO Byte (8-Bit) Register

Writes to the `USB_FIFOB[n]` register go to the endpoint Tx FIFO and reads from the `USB_FIFOB[n]` register come from the endpoint Rx FIFO. The `USB_FIFOB[n]`, `USB_FIFOH[n]`, and `USB_FIFO[n]` registers are the same register. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (`USB_FIFO[n]` register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, the program should use a half-word (`USB_FIFOH[n]` register) or byte (`USB_FIFOB[n]` register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

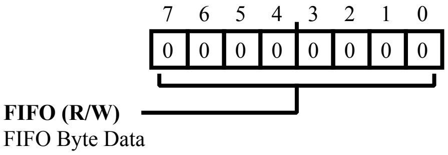


Figure 27-79: USB_FIFOB[n] Register Diagram

Table 27-55: USB_FIFOB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FIFO	FIFO Byte Data. The `USB_FIFOB[n].FIFO` bits provide byte access to the USB Tx and Rx endpoint FIFOs.

FIFO Half-Word (16-Bit) Register

Writes to the `USB_FIFOH[n]` register go to the endpoint Tx FIFO and reads from the `USB_FIFOH[n]` register come from the endpoint Rx FIFO. The `USB_FIFOB[n]`, `USB_FIFOH[n]`, and `USB_FIFO[n]` registers are the same register. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (`USB_FIFO[n]` register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, programs should use a half-word (`USB_FIFOH[n]` register) or byte (`USB_FIFOB[n]` register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

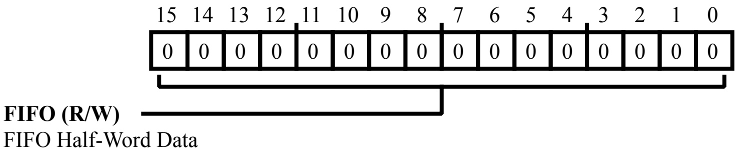


Figure 27-80: USB_FIFOH[n] Register Diagram

Table 27-56: USB_FIFOH[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	FIFO	FIFO Half-Word Data. The `USB_FIFOH[n].FIFO` bits provide half-word access to the USB Tx and Rx endpoint FIFOs.

FIFO Word (32-Bit) Register

Writes to the `USB_FIFO[n]` register go to the endpoint Tx FIFO and reads from the `USB_FIFO[n]` register come from the endpoint Rx FIFO. The `USB_FIFOB[n]`, `USB_FIFOH[n]`, and `USB_FIFO[n]` registers are the same registers. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (`USB_FIFO[n]` register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, programs should use a half-word (`USB_FIFOH[n]` register) or byte (`USB_FIFOB[n]` register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

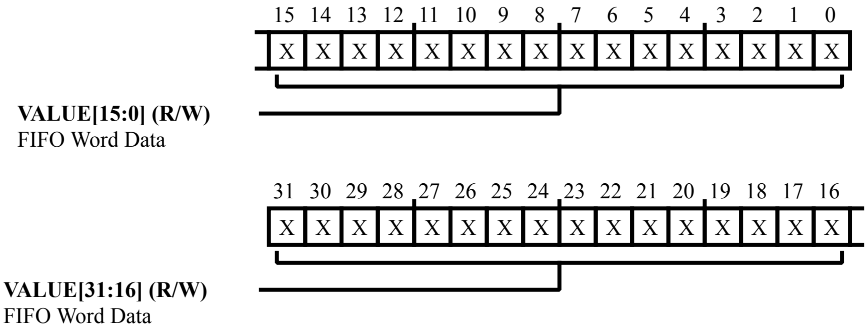

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
FIFO Word Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
FIFO Word Data

Figure 27-81: USB_FIFO[n] Register Diagram

Table 27-57: USB_FIFO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	FIFO Word Data. The `USB_FIFO[n].VALUE` bits provide word access to the USB Tx and Rx endpoint FIFOs.

Frame Number Register

The `USB_FRAME` register contains the frame number of the last received frame. The data in this register has bit 10 as the MSB and bit 0 as the LSB.

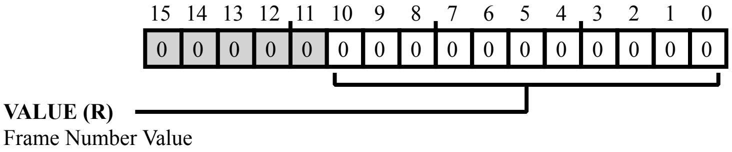

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Frame Number Value

Figure 27-82: USB_FRAME Register Diagram

Table 27-58: USB_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Frame Number Value. The `USB_FRAME.VALUE` bits contains the frame number of the last received frame. The data in this field has bit 10 as the MSB and bit 0 as the LSB.

Full-Speed EOF 1 Register

The `USB_FS_EOF1` register defines the minimum time gap allowed between the start of the last transaction and the end of frame for full-speed transactions.

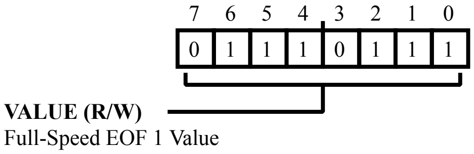


Figure 27-83: USB_FS_EOF1 Register Diagram

Table 27-59: USB_FS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Full-Speed EOF 1 Value. The `USB_FS_EOF1.VALUE` bits set the time before the end of the frame to stop beginning new transactions (in units of 533.3ns) for full-speed transactions. The default setting corresponds to 63.46us.

High-Speed EOF 1 Register

The `USB_HS_EOF1` register defines the minimum time gap allowed between the start of the last transaction and the end of frame for high-speed transactions.

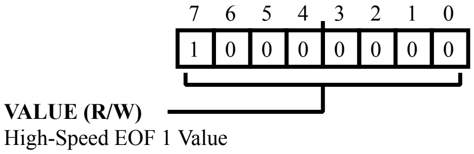


Figure 27-84: USB_HS_EOF1 Register Diagram

Table 27-60: USB_HS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	High-Speed EOF 1 Value. The `USB_HS_EOF1.VALUE` sets the time before the end of the frame to stop beginning new transactions (in units of 133.3ns) for high-speed transactions. The default setting corresponds to 17.07us.

ID Control

The `USB_IDCTL` register can be used to override the ID pin and force the controller to act as an A-device or B-device.

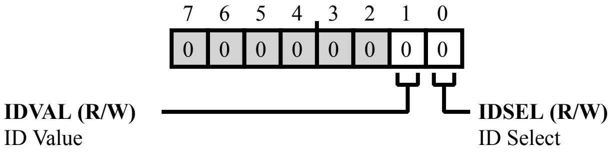


Figure 27-85: USB_IDCTL Register Diagram

Table 27-61: USB_IDCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	IDVAL	ID Value. When the `USB_IDCTL.IDSEL` bit =1, the `USB_IDCTL.IDVAL` bit sets the value of the ID input to the controller. This bit has no effect if `USB_IDCTL.IDSEL`=0.	
		0	A-Device
		1	B-Device
0 (R/W)	IDSEL	ID Select. The `USB_IDCTL.IDSEL` bit selects the source of the ID input to the controller. This can be used to bypass the ID input pin and force the controller to act as an A-device or B-device.	
		0	ID pin selected for controller input
		1	IDCTL[1] selected for controller input

Common Interrupts Enable Register

The `USB_IEN` register enables interrupts for USB controller system sources. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the `USB_IRQ` register is set.

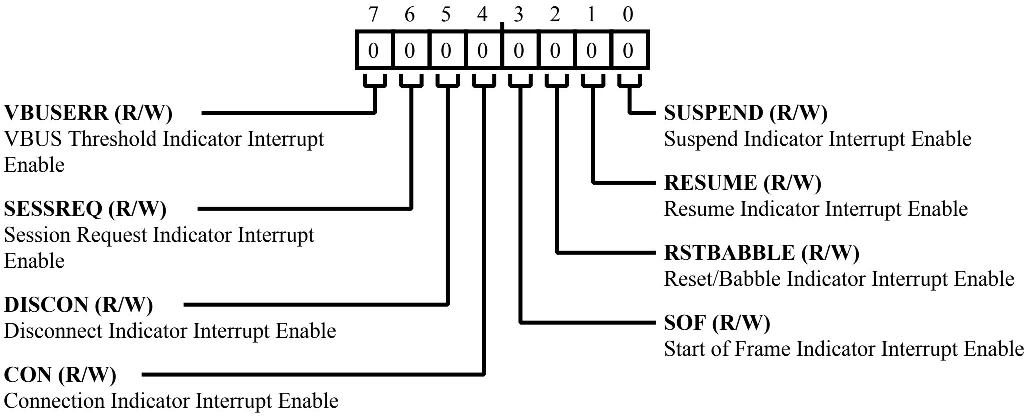

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
VBUSERR (R/W)
VBUS Threshold Indicator Interrupt Enable
SESSREQ (R/W)
Session Request Indicator Interrupt Enable
DISCON (R/W)
Disconnect Indicator Interrupt Enable
CON (R/W)
Connection Indicator Interrupt Enable
SUSPEND (R/W)
Suspend Indicator Interrupt Enable
RESUME (R/W)
Resume Indicator Interrupt Enable
RSTBABBLE (R/W)
Reset/Babble Indicator Interrupt Enable
SOF (R/W)
Start of Frame Indicator Interrupt Enable

Figure 27-86: USB_IEN Register Diagram

Table 27-62: USB_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	VBUSERR	VBUS Threshold Indicator Interrupt Enable. The `USB_IEN.VBUSERR` bit enables the `USB_IRQ.VBUSERR` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	SESSREQ	Session Request Indicator Interrupt Enable. The `USB_IEN.SESSREQ` bit enables the `USB_IRQ.SESSREQ` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	DISCON	Disconnect Indicator Interrupt Enable. The `USB_IEN.DISCON` bit enables the `USB_IRQ.DISCON` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	CON	Connection Indicator Interrupt Enable. The `USB_IEN.CON` bit enables the `USB_IRQ.CON` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-62: USB_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SOF	Start of Frame Indicator Interrupt Enable. The `USB_IEN.SOF` bit enables the `USB_IRQ.SOF` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	RSTBABBLE	Reset/Babble Indicator Interrupt Enable. The `USB_IEN.RSTBABBLE` bit enables the `USB_IRQ.RSTBABBLE` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	RESUME	Resume Indicator Interrupt Enable. The `USB_IEN.RESUME` bit enables the `USB_IRQ.RESUME` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	SUSPEND	Suspend Indicator Interrupt Enable. The `USB_IEN.SUSPEND` bit enables the `USB_IRQ.SUSPEND` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Index Register

The `USB_INDEX` register contains an index value for mirrored addressing of USB controller endpoint control and status registers.

There is one set of registers, but they are mirrored at two address locations if the endpoint is selected by the `USB_INDEX` register. An endpoint's register set only appears in the indexed location if the `USB_INDEX` register is written with that endpoint number. You can read/write an endpoint's register in either the directly mapped location which is always visible, or in the indexed location which is only visible if the `USB_INDEX` register is written with the endpoint number. The `USB_INDEX` register and indexed address locations only affect address decoding. For example, loading a 0 into the `USB_INDEX` register selects endpoint 0 access.

The `USB_INDEX` register can be used for indexed access of the directly mapped control/status registers from USB controller address offset 0x100-0x1FF. For products supporting the dynamic FIFO size feature, the endpoint Tx/Rx size and address registers always use the `USB_INDEX` register, there is no direct mapping for these endpoint-specific registers. The multi-point `USB_MP[n]_TXFUNCADDR`, `USB_MP[n]_TXHUBADDR`, `USB_MP[n]_TXHUBPORT`, `USB_MP[n]_RXFUNCADDR`, `USB_MP[n]_RXHUBADDR`, and `USB_MP[n]_RXHUBPORT` registers only have direct mapping, no indexed mapping.

Before accessing an endpoint's control/status registers using the indexed range, write the endpoint number to the `USB_INDEX` register to ensure that the correct control/status registers appear in the indexed range of the memory map.

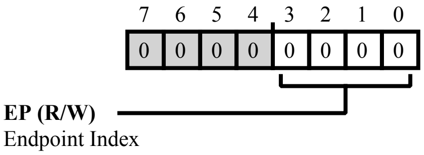


Figure 27-87: USB_INDEX Register Diagram

Table 27-63: USB_INDEX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EP	Endpoint Index. The `USB_INDEX.EP` bits select mirrored access for an endpoints indexed control and status registers. Valid values for this bit field are 0-11.

Receive Interrupt Register

The `USB_INTRRX` register indicates which interrupts are currently active for the receive (Rx) endpoints. Note that the USB controller automatically clears this register when it is read.

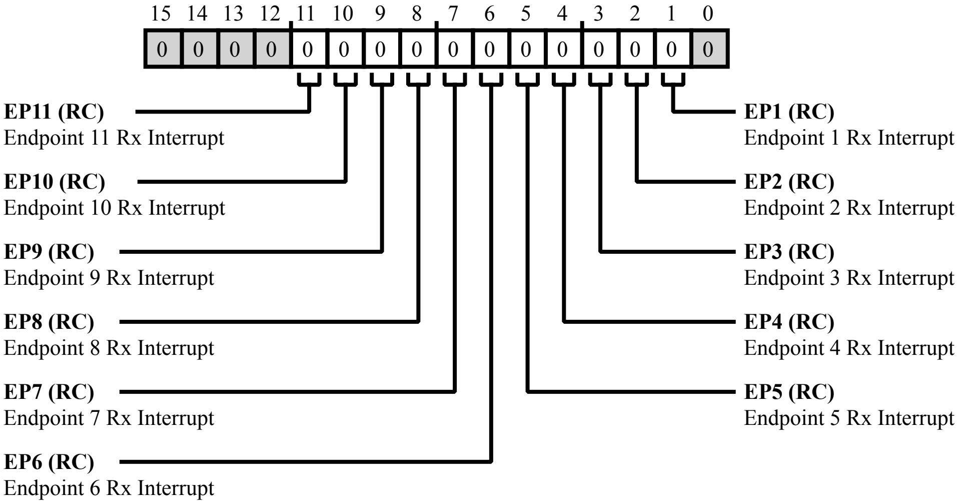

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EP11 (RC)
Endpoint 11 Rx Interrupt
EP10 (RC)
Endpoint 10 Rx Interrupt
EP9 (RC)
Endpoint 9 Rx Interrupt
EP8 (RC)
Endpoint 8 Rx Interrupt
EP7 (RC)
Endpoint 7 Rx Interrupt
EP6 (RC)
Endpoint 6 Rx Interrupt
EP1 (RC)
Endpoint 1 Rx Interrupt
EP2 (RC)
Endpoint 2 Rx Interrupt
EP3 (RC)
Endpoint 3 Rx Interrupt
EP4 (RC)
Endpoint 4 Rx Interrupt
EP5 (RC)
Endpoint 5 Rx Interrupt

Figure 27-88: USB_INTRRX Register Diagram

Table 27-64: USB_INTRRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	Endpoint 11 Rx Interrupt. The `USB_INTRRX.EP11` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
10 (RC/NW)	EP10	Endpoint 10 Rx Interrupt. The `USB_INTRRX.EP10` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	Endpoint 9 Rx Interrupt. The `USB_INTRRX.EP9` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 27-64: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (RC/NW)	EP8	Endpoint 8 Rx Interrupt. The `USB_INTRRX.EP8` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
7 (RC/NW)	EP7	Endpoint 7 Rx Interrupt. The `USB_INTRRX.EP7` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	Endpoint 6 Rx Interrupt. The `USB_INTRRX.EP6` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	Endpoint 5 Rx Interrupt. The `USB_INTRRX.EP5` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	Endpoint 4 Rx Interrupt. The `USB_INTRRX.EP4` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	EP3	Endpoint 3 Rx Interrupt. The `USB_INTRRX.EP3` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 27-64: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (RC/NW)	EP2	Endpoint 2 Rx Interrupt. The `USB_INTRRX.EP2` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	Endpoint 1 Rx Interrupt. The `USB_INTRRX.EP1` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Receive Interrupt Enable Register

The `USB_INTRRXE` register enables interrupts for the receive (Rx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the `USB_INTRRX` register is set.

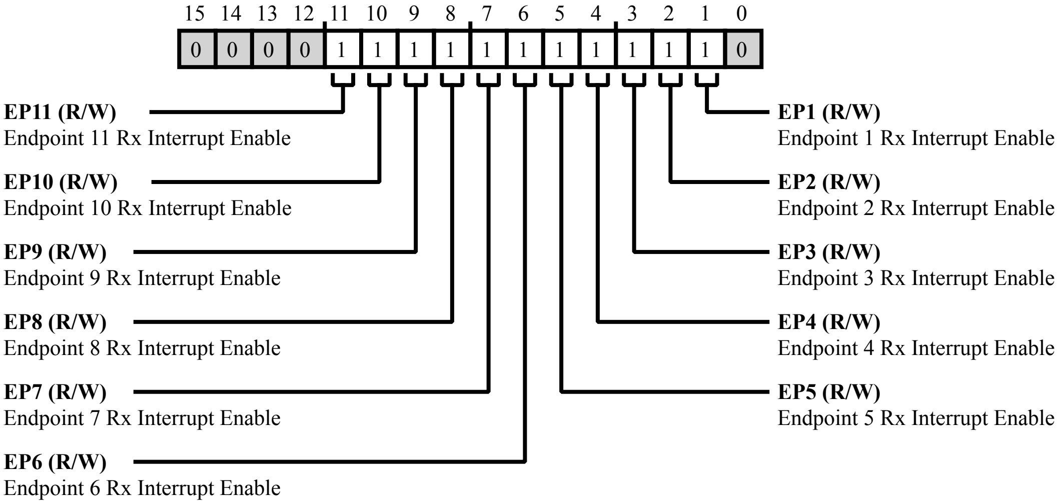

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0
EP11 (R/W)
Endpoint 11 Rx Interrupt Enable
EP10 (R/W)
Endpoint 10 Rx Interrupt Enable
EP9 (R/W)
Endpoint 9 Rx Interrupt Enable
EP8 (R/W)
Endpoint 8 Rx Interrupt Enable
EP7 (R/W)
Endpoint 7 Rx Interrupt Enable
EP6 (R/W)
Endpoint 6 Rx Interrupt Enable
EP1 (R/W)
Endpoint 1 Rx Interrupt Enable
EP2 (R/W)
Endpoint 2 Rx Interrupt Enable
EP3 (R/W)
Endpoint 3 Rx Interrupt Enable
EP4 (R/W)
Endpoint 4 Rx Interrupt Enable
EP5 (R/W)
Endpoint 5 Rx Interrupt Enable

Figure 27-89: USB_INTRRXE Register Diagram

Table 27-65: USB_INTRRXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	Endpoint 11 Rx Interrupt Enable. The `USB_INTRRXE.EP11` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	Endpoint 10 Rx Interrupt Enable. The `USB_INTRRXE.EP10` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	Endpoint 9 Rx Interrupt Enable. The `USB_INTRRXE.EP9` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-65: USB_INTRRXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	EP8	Endpoint 8 Rx Interrupt Enable. The `USB_INTRRXE.EP8` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	Endpoint 7 Rx Interrupt Enable. The `USB_INTRRXE.EP7` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	Endpoint 6 Rx Interrupt Enable. The `USB_INTRRXE.EP6` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	Endpoint 5 Rx Interrupt Enable. The `USB_INTRRXE.EP5` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	Endpoint 4 Rx Interrupt Enable. The `USB_INTRRXE.EP4` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	Endpoint 3 Rx Interrupt Enable. The `USB_INTRRXE.EP3` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	Endpoint 2 Rx Interrupt Enable. The `USB_INTRRXE.EP2` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-65: USB_INTRRXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	EP1	Endpoint 1 Rx Interrupt Enable. The `USB_INTRRXE.EP1` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Transmit Interrupt Register

The `USB_INTRTX` register indicates which interrupts are currently active for endpoint 0 and the transmit (Tx) endpoints. Note that the USB controller automatically clears this register when it is read.

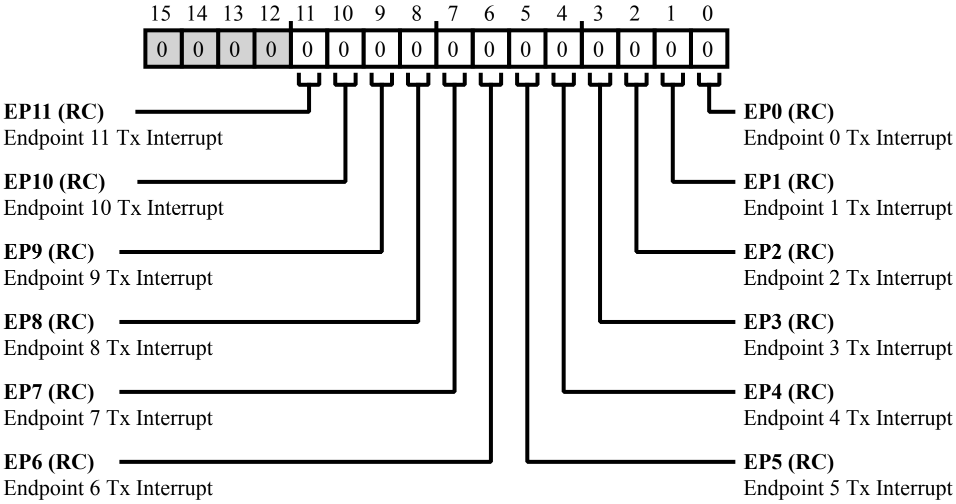

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EP11 (RC)
Endpoint 11 Tx Interrupt
EP10 (RC)
Endpoint 10 Tx Interrupt
EP9 (RC)
Endpoint 9 Tx Interrupt
EP8 (RC)
Endpoint 8 Tx Interrupt
EP7 (RC)
Endpoint 7 Tx Interrupt
EP6 (RC)
Endpoint 6 Tx Interrupt
EP0 (RC)
Endpoint 0 Tx Interrupt
EP1 (RC)
Endpoint 1 Tx Interrupt
EP2 (RC)
Endpoint 2 Tx Interrupt
EP3 (RC)
Endpoint 3 Tx Interrupt
EP4 (RC)
Endpoint 4 Tx Interrupt
EP5 (RC)
Endpoint 5 Tx Interrupt

Figure 27-90: USB_INTRTX Register Diagram

Table 27-66: USB_INTRTX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	Endpoint 11 Tx Interrupt. The `USB_INTRTX.EP11` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
10 (RC/NW)	EP10	Endpoint 10 Tx Interrupt. The `USB_INTRTX.EP10` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	Endpoint 9 Tx Interrupt. The `USB_INTRTX.EP9` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 27-66: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (RC/NW)	EP8	Endpoint 8 Tx Interrupt. The `USB_INTRTX.EP8` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
7 (RC/NW)	EP7	Endpoint 7 Tx Interrupt. The `USB_INTRTX.EP7` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	Endpoint 6 Tx Interrupt. The `USB_INTRTX.EP6` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	Endpoint 5 Tx Interrupt. The `USB_INTRTX.EP5` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	Endpoint 4 Tx Interrupt. The `USB_INTRTX.EP4` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	EP3	Endpoint 3 Tx Interrupt. The `USB_INTRTX.EP3` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 27-66: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (RC/NW)	EP2	Endpoint 2 Tx Interrupt. The `USB_INTRTX.EP2` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	Endpoint 1 Tx Interrupt. The `USB_INTRTX.EP1` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	EP0	Endpoint 0 Tx Interrupt. The `USB_INTRTX.EP0` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Transmit Interrupt Enable Register

The `USB_INTRTXE` register enables interrupts for endpoint 0 and the transmit (Tx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the `USB_INTRTX` register is set.

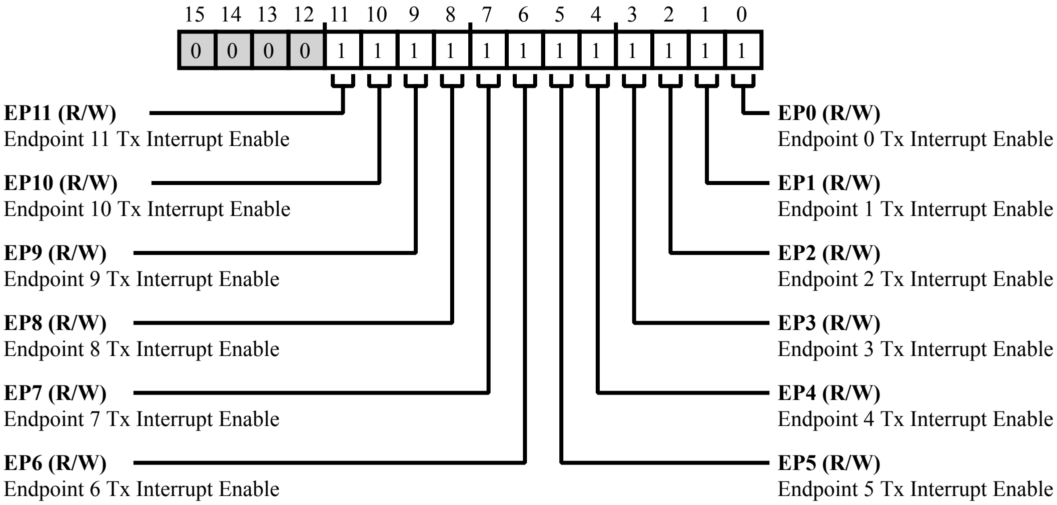

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
EP11 (R/W)
Endpoint 11 Tx Interrupt Enable
EP10 (R/W)
Endpoint 10 Tx Interrupt Enable
EP9 (R/W)
Endpoint 9 Tx Interrupt Enable
EP8 (R/W)
Endpoint 8 Tx Interrupt Enable
EP7 (R/W)
Endpoint 7 Tx Interrupt Enable
EP6 (R/W)
Endpoint 6 Tx Interrupt Enable
EP0 (R/W)
Endpoint 0 Tx Interrupt Enable
EP1 (R/W)
Endpoint 1 Tx Interrupt Enable
EP2 (R/W)
Endpoint 2 Tx Interrupt Enable
EP3 (R/W)
Endpoint 3 Tx Interrupt Enable
EP4 (R/W)
Endpoint 4 Tx Interrupt Enable
EP5 (R/W)
Endpoint 5 Tx Interrupt Enable

Figure 27-91: USB_INTRTXE Register Diagram

Table 27-67: USB_INTRTXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	Endpoint 11 Tx Interrupt Enable. The `USB_INTRTXE.EP11` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	Endpoint 10 Tx Interrupt Enable. The `USB_INTRTXE.EP10` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	Endpoint 9 Tx Interrupt Enable. The `USB_INTRTXE.EP9` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-67: USB_INTRTXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	EP8	Endpoint 8 Tx Interrupt Enable. The `USB_INTRTXE.EP8` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	Endpoint 7 Tx Interrupt Enable. The `USB_INTRTXE.EP7` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	Endpoint 6 Tx Interrupt Enable. The `USB_INTRTXE.EP6` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	Endpoint 5 Tx Interrupt Enable. The `USB_INTRTXE.EP5` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	Endpoint 4 Tx Interrupt Enable. The `USB_INTRTXE.EP4` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	Endpoint 3 Tx Interrupt Enable. The `USB_INTRTXE.EP3` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	Endpoint 2 Tx Interrupt Enable. The `USB_INTRTXE.EP2` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-67: USB_INTRTXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	EP1	Endpoint 1 Tx Interrupt Enable. The `USB_INTRTXE.EP1` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	EP0	Endpoint 0 Tx Interrupt Enable. The `USB_INTRTXE.EP0` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Common Interrupts Register

The `USB_IRQ` register indicates which interrupts are currently active for USB controller system sources. Note that the USB controller automatically clears this register when it is read.

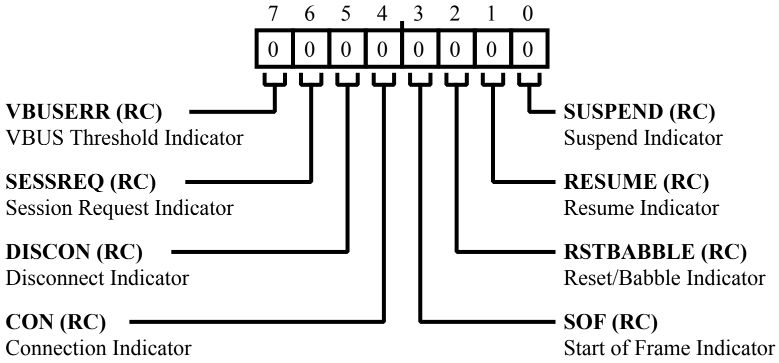

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
VBUSERR (RC)
VBUS Threshold Indicator
SESSREQ (RC)
Session Request Indicator
DISCON (RC)
Disconnect Indicator
CON (RC)
Connection Indicator
SUSPEND (RC)
Suspend Indicator
RESUME (RC)
Resume Indicator
RSTBABBLE (RC)
Reset/Babble Indicator
SOF (RC)
Start of Frame Indicator

Figure 27-92: USB_IRQ Register Diagram

Table 27-68: USB_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	VBUSERR	VBUS Threshold Indicator. The `USB_IRQ.VBUSERR` bit indicates whether the USB controller has detected that the VBUS is below the VBUS valid threshold. This bit is valid only when the USB controller is an A device. Note that the `USB_IRQ.VBUSERR` bit and the `USB_VBUS_CTL.DRVINT` bit share an interrupt source line.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	SESSREQ	Session Request Indicator. The `USB_IRQ.SESSREQ` bit indicates whether the USB controller has detected a session request signal. This bit is valid only when the USB controller is an A device.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	DISCON	Disconnect Indicator. The `USB_IRQ.DISCON` bit indicates whether the USB controller has detected a device disconnect (host mode) or has detected a session end (peripheral mode).	
		0	No Interrupt
		1	Interrupt Pending

Table 27-68: USB_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (RC/NW)	CON	Connection Indicator. The `USB_IRQ.CON` bit indicates whether the USB controller has detected a device connection. This bit is valid only in host mode.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	SOF	Start of Frame Indicator. The `USB_IRQ.SOF` bit indicates whether the USB controller has detected a start of a frame.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	RSTBABBLE	Reset/Babble Indicator. The `USB_IRQ.RSTBABBLE` bit indicates whether the USB controller has detected reset signalling on the bus. In host mode, the USB controller also indicates when the USB controller detects babble. Note that the `USB_IRQ.RSTBABBLE` bit is only active after the first SOF has been sent.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	RESUME	Resume Indicator. The `USB_IRQ.RESUME` bit indicates whether the USB controller has detected resume signaling on the bus while the USB controller is in suspend mode.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	SUSPEND	Suspend Indicator. The `USB_IRQ.SUSPEND` bit indicates whether the USB controller has detected suspend signalling on the bus. This bit is valid only in peripheral mode.	
		0	No Interrupt
		1	Interrupt Pending

Link Information Register

The `USB_LINKINFO` register specifies the PHY-related delays.

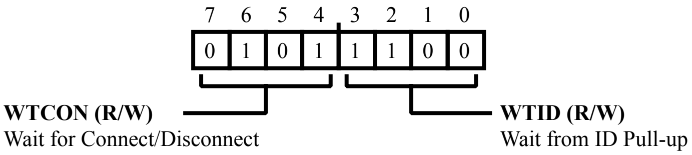

7 6 5 4 3 2 1 0
0 1 0 1 1 1 0 0
WTCON (R/W)
Wait for Connect/Disconnect
WTID (R/W)
Wait from ID Pull-up

Figure 27-93: USB_LINKINFO Register Diagram

Table 27-69: USB_LINKINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	WTCON	Wait for Connect/Disconnect. The `USB_LINKINFO.WTCON` bits set the wait time to be applied to allow the users to connect or disconnect filter.
3:0 (R/W)	WTID	Wait from ID Pull-up. The `USB_LINKINFO.WTID` bits set the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid.

LPM Attribute Register

The USB_LPM_ATTR register defines the link power management (LPM) attributes for LPM transactions and sleep/wake operation. In peripheral mode, the USB_LPM_ATTR register contains values received in the most recent, accepted (ACK'd) LPM transaction. In host mode, the USB_LPM_ATTR register contains values (loaded by software) that set up the next LPM transaction. The USB controller inserts the LPM values within the next LPM transaction.

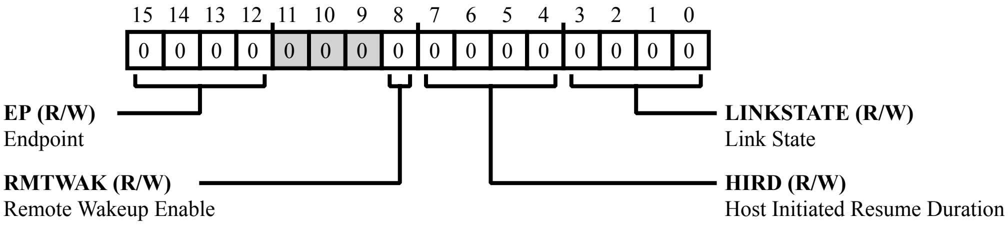


Figure 27-94: USB_LPM_ATTR Register Diagram

Table 27-70: USB_LPM_ATTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EP	Endpoint. The USB_LPM_ATTR.EP bits select the endpoint in the token packet of the LPM transaction.	
8 (R/W)	RMTWAK	Remote Wakeup Enable. The USB_LPM_ATTR.RMTWAK bit enables remote wakeup. This bit is applied on a temporary basis only and is only applied to the current suspend state. After the current suspend cycle, the remote wakeup capability that was negotiated during enumeration applies.	
		0	Disable Remote Wakeup
		1	Enable Remote Wakeup
7:4 (R/W)	HIRD	Host Initiated Resume Duration. The USB_LPM_ATTR.HIRD bits select the host-initiated resume duration. This value is the minimum time that the host drives resume on the bus. The value in this register corresponds to an actual resume time of: Resume Time = 50us + HIRD*75us. This equation produces results in a range of 50us to 1200us.	
3:0 (R/W)	LINKSTATE	Link State. The USB_LPM_ATTR.LINKSTATE bits is value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. (Enumerations not shown are reserved.)	
		1	Sleep State (L1)

LPM Control Register

The `USB_LPM_CTL` register controls link power management (LPM) operations, including LPM enable, NAK, resume, and mode transition.

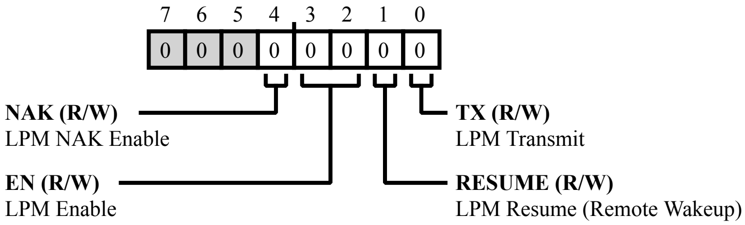


Figure 27-95: USB_LPM_CTL Register Diagram

Table 27-71: USB_LPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	NAK	LPM NAK Enable. The `USB_LPM_CTL.NAK` bit enables (in peripheral mode) a NAK-all-non-LPM transactions mode for all end points, forcing a NAK response to all transactions other than an LPM transaction. This bit only takes effect after the controller has been LPM suspended. In this case, the USB controller continues to NAK, until this bit has been cleared by software.	
		0	Disable LPM NAK
		1	Enable LPM NAK
3:2 (R/W)	EN	LPM Enable. The `USB_LPM_CTL.EN` bits enable (In peripheral mode) LPM operations. The LPM operation may be enabled at different levels, which determine the response of the USB controller to LPM transactions.	
		0	Disable LPM. LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		1	Disable LPM. LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		2	Enable Extended Transactions. LPM is not supported, but extended transactions are supported. The USB controller responds to an LPM transaction with a STALL.
		3	Enable LPM and Extended Transactions. Both LPM and extended transactions are supported. The USB controller responds with a NYET or an ACK as determined by the value of LPMXMT and other conditions.

Table 27-71: USB_LPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	RESUME	LPM Resume (Remote Wakeup). The `USB_LPM_CTL.RESUME` bit initiates resume (remote wakeup). The operation of this bit differs from the `USB_POWER.RESUME` bit in that the LPM resume signal timing is controlled by hardware. When set, the USB controller asserts resume signaling for 50us in host mode or asserts resume signaling for the time specified by the `USB_LPM_ATTR.HIRD` field in device mode. The `USB_LPM_CTL.RESUME` bit is self-clearing.
		0 No Action
		1 LPM Resume
0 (R/W)	TX	LPM Transmit. The `USB_LPM_CTL.TX` bit puts the USB controller in LPM transmit mode. But, this mode operates differently in host mode versus peripheral mode. In peripheral mode, this bit is set by software to instruct the controller to transition to the L1 state upon receipt of the next LPM transaction. This bit is only effective if LPM enable (`USB_LPM_CTL.EN`) is set to 0x3. The LPM transmit enable bit can be set in the same cycle as LPM enable. If the `USB_LPM_CTL.TX` and `USB_LPM_CTL.EN` bits are enabled, the USB controller can respond in the following ways: • If no data is pending (all transmit FIFOs are empty), the USB controller responds with an ACK, clears the `USB_LPM_CTL.TX` bit, and generates a software interrupt. • If data is pending (data resides in at least one transmit FIFO), the USB controller responds with a NYET, does not clear the `USB_LPM_CTL.TX` bit, and generates a software interrupt. In host mode, this bit is set by software to transmit an LPM transaction. This bit is self-clearing. The USB controller clears this bit immediately on receipt of any token or after three timeouts have occurred.
		0 Disable LPM Tx
		1 Enable LPM Tx

LPM Function Address Register

The `USB_LPM_FADDR` register selects the link power management (LPM) function address.

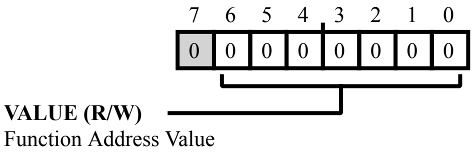


Figure 27-96: USB_LPM_FADDR Register Diagram

Table 27-72: USB_LPM_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The `USB_LPM_FADDR.VALUE` bits hold the LPM function address value that the USB controller places in the LPM payload.

LPM Interrupt Enable Register

The USB_LPM_IEN register enables the link power management (LPM) related interrupts. When an interrupt is enabled in this register and the corresponding interrupt is pending in USB_LPM_IRQ, the USB controller generates the interrupt. When an interrupt is disabled in this register, the corresponding interrupt may be pending in USB_LPM_IRQ, but the USB controller does not generate an interrupt.

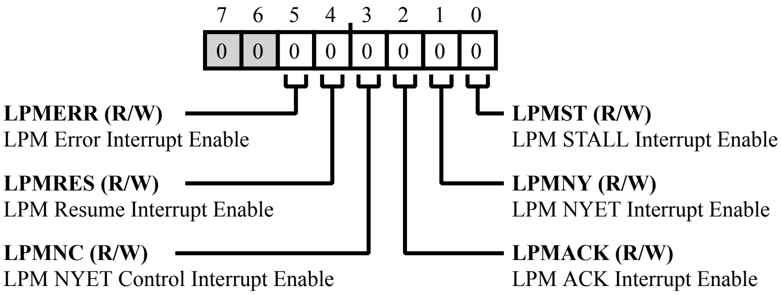

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
LPMERR (R/W)
LPM Error Interrupt Enable
LPMRES (R/W)
LPM Resume Interrupt Enable
LPMNC (R/W)
LPM NYET Control Interrupt Enable
LPMST (R/W)
LPM STALL Interrupt Enable
LPMNY (R/W)
LPM NYET Interrupt Enable
LPMACK (R/W)
LPM ACK Interrupt Enable

Figure 27-97: USB_LPM_IEN Register Diagram

Table 27-73: USB_LPM_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	LPMERR	LPM Error Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	LPMRES	LPM Resume Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	LPMNC	LPM NYET Control Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	LPMACK	LPM ACK Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	LPMNY	LPM NYET Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 27-73: USB_LPM_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	LPMST	LPM STALL Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

LPM Interrupt Status Register

The `USB_LPM_IRQ` register indicates link power management (LPM) related interrupt status. The USB controller clears this register when it is read.

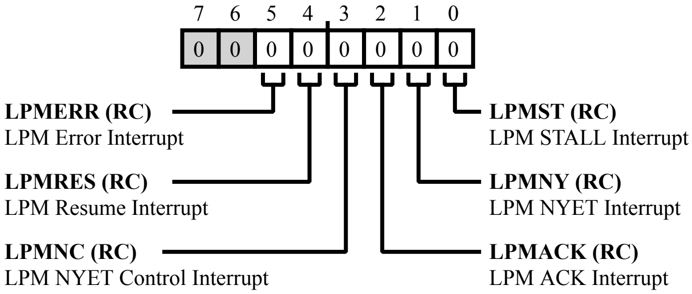

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
LPMERR (RC)
LPM Error Interrupt
LPMRES (RC)
LPM Resume Interrupt
LPMNC (RC)
LPM NYET Control Interrupt
LPMST (RC)
LPM STALL Interrupt
LPMNY (RC)
LPM NYET Interrupt
LPMACK (RC)
LPM ACK Interrupt

Figure 27-98: USB_LPM_IRQ Register Diagram

Table 27-74: USB_LPM_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RC/NW)	LPMERR	LPM Error Interrupt. The `USB_LPM_IRQ.LPMERR` bit indicates an LPM error interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set if an LPM transaction is received that has a `USB_LPM_ATTR.LINKSTATE` field that is not supported. The USB controller responds to the transaction with a STALL. Note that the USB controller updates the `USB_LPM_ATTR` register, so software can observe the non-compliant LPM packet payload. In host mode, this bit is set if the response to a LPM transaction is received with a bit stuff or PID error. No suspend occurs and the state of the device is now unknown.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (RC/NW)	LPMRES	LPM Resume Interrupt. The `USB_LPM_IRQ.LPMRES` bit indicates that the USB controller has been resumed for any reason. This bit is mutually exclusive from the `USB_POWER.RESUME` bit.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 27-74: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (RC/NW)	LPMNC	LPM NYET Control Interrupt. The `USB_LPM_IRQ.LPMNC` bit indicates an LPM NYET control interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET due to data pending in the transmit FIFOs. This interrupt may only occur when the `USB_LPM_CTL.EN` field is set to 11, the `USB_LPM_CTL.TX` field is set to 1, and there is data pending in the transmit FIFOs. In host mode, this bit is set when an LPM transaction has been transmitted, but has failed to complete. The transaction failure is due to a timeout or bit errors in the response for three attempts. 0 No Interrupt Pending 1 Interrupt Pending
2 (RC/NW)	LPMACK	LPM ACK Interrupt. The `USB_LPM_IRQ.LPMACK` bit indicates an LPM ACK interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with an ACK. This interrupt may only occur when the `USB_LPM_CTL.EN` field is set to 11, the `USB_LPM_CTL.TX` field is set to 1, and there is no data pending in the controller transmit FIFOs. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with an ACK. 0 No Interrupt Pending 1 Interrupt Pending
1 (RC/NW)	LPMNY	LPM NYET Interrupt. The `USB_LPM_IRQ.LPMNY` bit indicates an LPM NYET interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET. This interrupt may only occur when the `USB_LPM_CTL.EN` field is set to 11, and the `USB_LPM_CTL.TX` field is set to 0. In host mode, this bit is set when an LPM transaction is transmitted and the device responds with a NYET. 0 No Interrupt Pending 1 Interrupt Pending

Table 27-74: USB_LPM_IRQ Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">0
(RC/NW)</td><td rowspan="3">LPMST</td><td colspan="2">LPM STALL Interrupt.
The <code>USB_LPM_IRQ.LPMST</code> bit indicates an LPM STALL interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode.
This bit is set when an LPM transaction is received, and the USB controller responds with a STALL. This interrupt may only occur when the <code>USB_LPM_CTL.EN</code> field is set to 01.
In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with a STALL.</td></tr>
<tr><td>0</td><td>No Interrupt Pending</td></tr>
<tr><td>1</td><td>Interrupt Pending</td></tr>
</table>

Low-Speed EOF 1 Register

The `USB_LS_EOF1` register defines the minimum time gap allowed between the start of the last transaction and the end of frame for low-speed transactions.

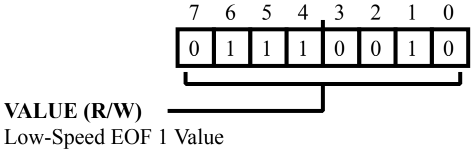


Figure 27-99: USB_LS_EOF1 Register Diagram

Table 27-75: USB_LS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Low-Speed EOF 1 Value. The `USB_LS_EOF1.VALUE` bits set the time before end of frame to stop beginning new transactions (in units of 1.067us) for low-speed transactions. The default setting corresponds to 121.6us.

MPn Receive Function Address Register

The `USB_MP[n]_RXFUNCADDR` register specifies the receive endpoint's target address in host mode. This register is not used in device mode. Note that the `USB_MP[n]_RXFUNCADDR` register does not exist for EP0.

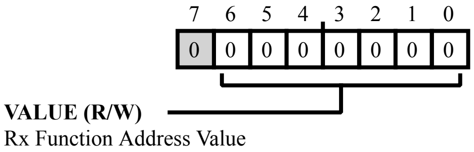


Figure 27-100: USB_MP[n]_RXFUNCADDR Register Diagram

Table 27-76: USB_MP[n]_RXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Rx Function Address Value. The `USB_MP[n]_RXFUNCADDR.VALUE` bits hold the address of the target device for this endpoint.

MPn Receive Hub Address Register

The `USB_MP[n]_RXHUBADDR` register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Note that the `USB_MP[n]_RXHUBADDR` register does not exist for EP0.

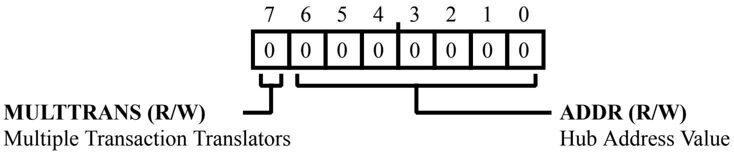


Figure 27-101: USB_MP[n]_RXHUBADDR Register Diagram

Table 27-77: USB_MP[n]_RXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The `USB_MP[n]_RXHUBADDR.MULTTRANS` bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The `USB_MP[n]_RXHUBADDR.ADDR` bits hold the address of the hub to which this device is connected.	

MPn Receive Hub Port Register

The `USB_MP[n]_RXHUBPORT` register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The `USB_MP[n]_RXHUBPORT` register lets the USB controller support SPLIT transactions. Note that the `USB_MP[n]_RXHUBPORT` register does not exist for EP0.

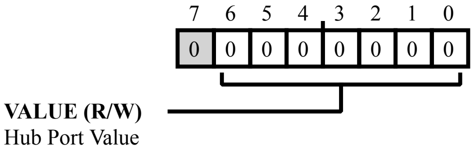


Figure 27-102: USB_MP[n]_RXHUBPORT Register Diagram

Table 27-78: USB_MP[n]_RXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The `USB_MP[n]_RXHUBPORT.VALUE` bits hold the hub port value of the target device for this endpoint.

MPn Transmit Function Address Register

The `USB_MP[n]_TXFUNCADDR` register specifies the transmit endpoint's target address in host mode. This register is not used in device mode. Note that the `USB_MP[n]_TXFUNCADDR` register must be setup for EP0. (The `USB_MP[n]_RXFUNCADDR` register does not exist for EP0.)

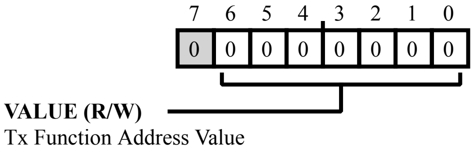


Figure 27-103: USB_MP[n]_TXFUNCADDR Register Diagram

Table 27-79: USB_MP[n]_TXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Tx Function Address Value. The `USB_MP[n]_TXFUNCADDR.VALUE` bits hold the address of the target device for this endpoint.

MPn Transmit Hub Address Register

The USB_MP[n]_TXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Also, note that EP0 only uses the USB_MP[n]_TXHUBADDR register. (The USB_MP[n]_RXHUBADDR register does not exist for EP0.)

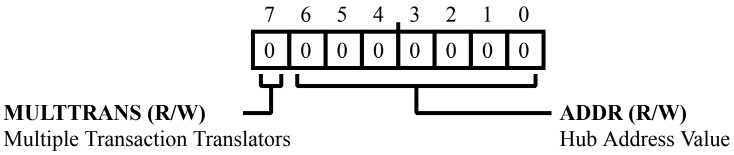


Figure 27-104: USB_MP[n]_TXHUBADDR Register Diagram

Table 27-80: USB_MP[n]_TXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MP[n]_TXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MP[n]_TXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Transmit Hub Port Register

The `USB_MP[n]_TXHUBPORT` register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The `USB_MP[n]_TXHUBPORT` register lets the USB controller support SPLIT transactions. EP0 only uses the `USB_MP[n]_TXHUBPORT` register. (The `USB_MP[n]_RXHUBPORT` register does not exist for EP0.)

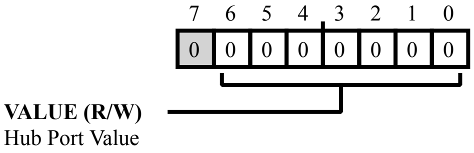


Figure 27-105: USB_MP[n]_TXHUBPORT Register Diagram

Table 27-81: USB_MP[n]_TXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The `USB_MP[n]_TXHUBPORT.VALUE` bits hold the hub port value of the target device for this endpoint.

PHY Control Register

The USB_PHY_CTL register provides access to PHY control features.

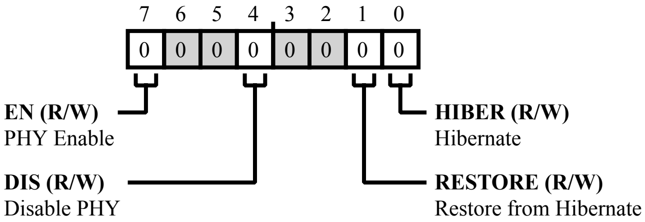


Figure 27-106: USB_PHY_CTL Register Diagram

Table 27-82: USB_PHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EN	PHY Enable. The USB_PHY_CTL.EN bit enables the USB controller PHY. This bit enables the schmitt-trigger inputs on D+ and D- to detect session request protocol. The bit also enables the bias circuits and VBUS comparators to detect when a host is connected. This bit should be set for all USB controller operations.	
4 (R/W)	DIS	Disable PHY. The USB_PHY_CTL.DIS bit disables the PHY, so it draws minimal power.	
		0	Enable USB PHY and 5V protection on USB signals.
		1	Disable USB PHY and 5V protection on USB signals. Caution: When 5V protection is disabled, the absolute max voltage on USB signals is reduced. See the data sheet for details.
1 (R/W)	RESTORE	Restore from Hibernate. The USB_PHY_CTL.RESTORE bit causes the PHY to come out of hibernate and release its latches.	
0 (R/W)	HIBER	Hibernate. The USB_PHY_CTL.HIBER bit causes the PHY to prepare for hibernate. Latches hold the pullup/pulldown state when the core power is removed.	

PLL and Oscillator Control Register

The `USB_PLL_OSC` register provides access to PLL and oscillator-related control features.

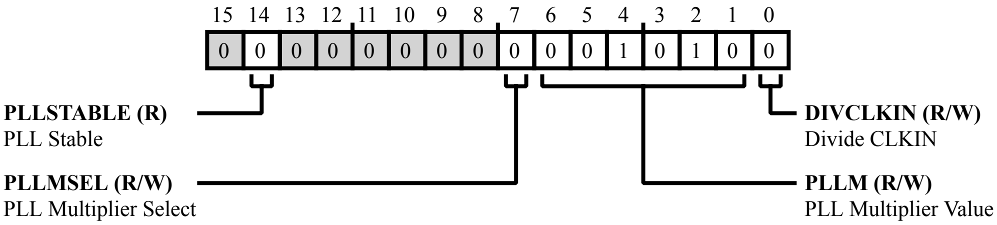

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
PLLSTABLE (R)
PLL Stable
PLLMSEL (R/W)
PLL Multiplier Select
DIVCLKIN (R/W)
Divide CLKIN
PLLM (R/W)
PLL Multiplier Value

Figure 27-107: USB_PLL_OSC Register Diagram

Table 27-83: USB_PLL_OSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PLLSTABLE	PLL Stable. The `USB_PLL_OSC.PLLSTABLE` status bit indicates that the oscillator and PLL clock are stable.
7 (R/W)	PLLMSEL	PLL Multiplier Select. The `USB_PLL_OSC.PLLMSEL` bit directs the PLL to use the PLL multiplier value stored in the `USB_PLL_OSC.PLLM` bits.
6:1 (R/W)	PLLM	PLL Multiplier Value. The `USB_PLL_OSC.PLLM` bit field contains the PLL multiplier. This field should be set such that CLKIN * `USB_PLL_OSC.PLLM` value = 480MHz.
0 (R/W)	DIVCLKIN	Divide CLKIN. The `USB_PLL_OSC.DIVCLKIN` bit enables a divide CLKIN by 2 function for the PLL.

Power and Device Control Register

The `USB_POWER` register controls suspend and resume signaling and some operational aspects of the USB controller.

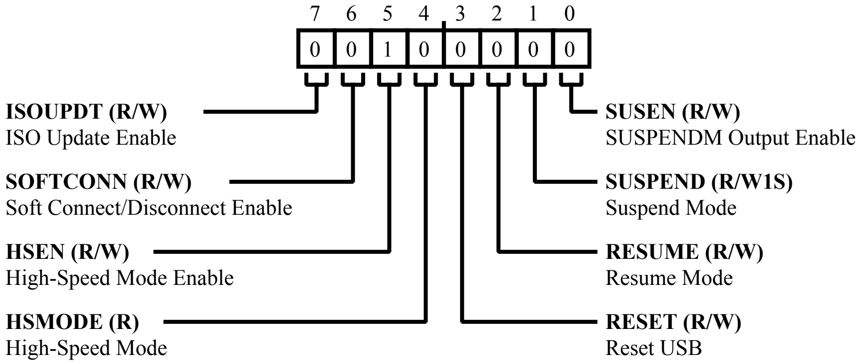

7 6 5 4 3 2 1 0
0 0 1 0 0 0 0 0
ISOUPDT (R/W)
ISO Update Enable
SOFTCONN (R/W)
Soft Connect/Disconnect Enable
HSEN (R/W)
High-Speed Mode Enable
HSMODE (R)
High-Speed Mode
SUSEN (R/W)
SUSPENDM Output Enable
SUSPEND (R/W1S)
Suspend Mode
RESUME (R/W)
Resume Mode
RESET (R/W)
Reset USB

Figure 27-108: USB_POWER Register Diagram

Table 27-84: USB_POWER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	ISOUPDT	ISO Update Enable. The `USB_POWER.ISOUPDT` bit directs the USB controller to waits for an SOF token from the time the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set before sending the packet. If an IN token is received before an SOF token, the USB controller send a zero length data packet. This `USB_POWER.ISOUPDT` bit only affects endpoints performing isochronous transfers. This bit only is valid in peripheral mode (`USB_DEV_CTL.HOSTMODE` = 0).	
		0	Disable ISO Update
		1	Enable ISO Update
6 (R/W)	SOFTCONN	Soft Connect/Disconnect Enable. In peripheral mode, the D+/- lines default to disconnected. Setting this bit enables the D+/- termination resistors. This bit is automatically set when the `USB_DEV_CTL.SESSION` bit is written with 1. The `USB_POWER.SOFTCONN` bit enables USB controller soft connect/disconnect, enabling the termination resistors for `USB_DP` (Data +) and `USB_DM` (Data -) pins. When disabled, these pins are three-stated. Note that `USB_POWER.SOFTCONN` only is valid in peripheral mode (`USB_DEV_CTL.HOSTMODE` = 0).	
		0	Disable Soft Connect/Disconnect
		1	Enable Soft Connect/Disconnect

Table 27-84: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	HSEN	High-Speed Mode Enable. The `USB_POWER.HSEN` bit enables USB controller negotiation for high speed (on devices supporting high-speed mode) when the device is reset by the hub/host. If disabled, the USB controller only operates in full-speed mode. When operating in full-speed mode, this bit should be cleared.	
		0	Disable Negotiation for HS Mode
		1	Enable Negotiation for HS Mode
4 (R/NW)	HSMODE	High-Speed Mode. The `USB_POWER.HSMODE` bit indicates whether or not the USB controller successfully negotiated high-speed mode during a USB controller reset. In peripheral mode (`USB_DEV_CTL.HOSTMODE` = 0), this bit has valid data when the USB controller completes reset. In host mode (`USB_DEV_CTL.HOSTMODE` = 1), this bit has valid data when the `USB_IRQ.RSTBABBLE` bit is cleared, remaining valid for the duration of the session.	
		0	Full-Speed Mode (HS fail during reset)
		1	High-Speed Mode (HS success during reset)
3 (R/W)	RESET	Reset USB. The `USB_POWER.RESET` bit indicates (in both host and peripheral modes) that the USB controller has detected that reset signaling is present on the bus. In peripheral mode (`USB_DEV_CTL.HOSTMODE` = 0), this bit is read only, but in host mode (`USB_DEV_CTL.HOSTMODE` = 1), this bit is read/write, permitting the processor core to set the bit and initiate a USB controller reset.	
		0	No Reset
		1	Reset USB
2 (R/W)	RESUME	Resume Mode. The `USB_POWER.RESUME` bit directs the USB controller to generate resume signaling when the function is in suspend mode (`USB_POWER.SUSPEND` =1). The processor core should clear this bit after 10 ms (a maximum of 15 ms) to end resume signaling. When the USB controller is in host mode (`USB_DEV_CTL.HOSTMODE` = 1), the USB controller automatically sets the `USB_POWER.RESUME` bit when resume signaling from the target is detected while the USB controller is suspended.	
		0	Disable Resume Signaling
		1	Enable Resume Signaling

Table 27-84: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	SUSPEND	Suspend Mode. When the USB controller is in host mode (`USB_DEV_CTL.HOSTMODE` = 1), the `USB_POWER.SUSPEND` bit enables suspend mode. When the USB controller is in peripheral mode (`USB_DEV_CTL.HOSTMODE` = 0), the USB controller sets the `USB_POWER.SUSPEND` bit on entry to suspend mode and clears the bit when the processor reads the `USB_IRQ` register. Note that the USB controller automatically clears this bit if the `USB_POWER.RESUME` bit is set.	
		0	Disable Suspend Mode (Host)
		1	Enable Suspend Mode (Host)
0 (R/W)	SUSEN	SUSPENDM Output Enable. The `USB_POWER.SUSEN` bit enables the SUSPENDM output (internal USB controller signal). When enabled, the SUSPENDM output signal is used by the USB controller PHY to power-down its drivers when the USB controller is not active.	
		0	Disable SUSPENDM Output
		1	Enable SUSPENDM Output

RAM Information Register

The `USB_RAMINFO` register provides information about the width of the USB controller RAM.

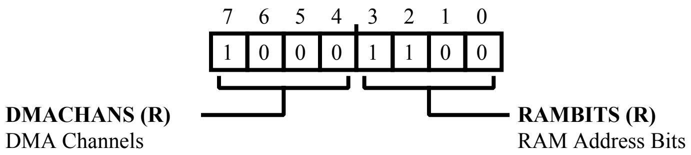


Figure 27-109: USB_RAMINFO Register Diagram

Table 27-85: USB_RAMINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	DMACHANS	DMA Channels. The `USB_RAMINFO.DMACHANS` bits indicate the number of DMA channels.
3:0 (R/NW)	RAMBITS	RAM Address Bits. The `USB_RAMINFO.RAMBITS` bits indicate the number of RAM address bits. The USB controller FIFO RAM is 32-bits wide. The number of bytes in the FIFO RAM may be calculated from the formula: $RAM\_bytes = 2^{(RAM\_Bits+2)}$

EPn Request Packet Count Register

The `USB_RQPKTCNT[n]` register specifies (in host mode) the number of packets to request in a block transfer of one or more bulk packets of size `USB_EP[n]_RXMAXP` from a receive endpoint. This register only applies for receive endpoints 1 through 11 in host mode.

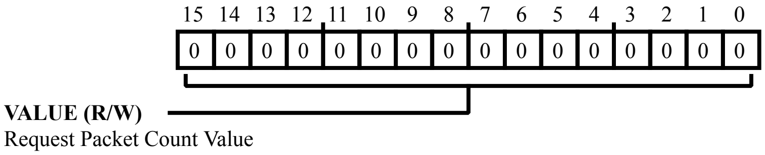


Figure 27-110: USB_RQPKTCNT[n] Register Diagram

Table 27-86: USB_RQPKTCNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Request Packet Count Value. The `USB_RQPKTCNT[n].VALUE` bits specify the number of bulk packets to request in a block transfer from a receive endpoint. This field is used with the auto request feature (`USB_EP[n]_RXCSR_H.AUTOREQ`).

Receive FIFO Address Register

The `USB_RXFIFOADDR` sets the start address for the selected Rx FIFO for endpoints 1-11. There is one receive FIFO address register for each endpoint, except endpoint 0. The `USB_RXFIFOADDR` register is indexed and selected by the `USB_INDEX` register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

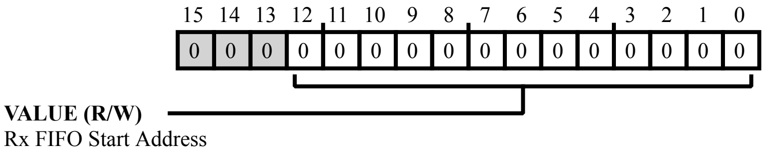


Figure 27-111: USB_RXFIFOADDR Register Diagram

Table 27-87: USB_RXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	VALUE	Rx FIFO Start Address. The `USB_RXFIFOADDR.VALUE` bits hold the start address of the selected endpoint FIFO (selected with the `USB_INDEX` register) in units of 8 bytes, according to the formula: FIFO address = `USB_RXFIFOADDR.VALUE` * 8

Receive FIFO Size Register

The USB_RXFIFOSZ register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed-access to the FIFO (packet) size selection for each Rx endpoint (except endpoint 0).

Note that a value greater than the maximum allowed of 1023 for full-speed USB controller operation produces unpredictable results.

Also, note that the value written to this register should match the programmed maximum individual packet size (MaxPktSize) of the standard endpoint descriptor for the associated endpoint. (See the Universal Serial Bus Specification Revision 2.0, Chapter 9). A mismatch could cause unexpected results. The total amount of data represented by the value written to this register must not exceed the Rx FIFO size, and should not exceed half the FIFO size if double-buffering is required.

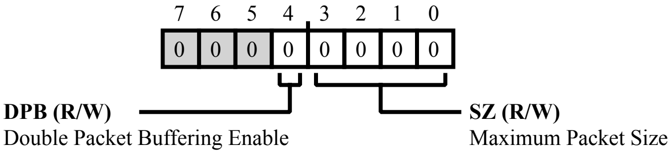


Figure 27-112: USB_RXFIFOSZ Register Diagram

Table 27-88: USB_RXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The USB_RXFIFOSZ.DPB bit enables double packet buffering, doubling the FIFO (packet) size selected with the USB_RXFIFOSZ.SZ field.	
		0	Single Packet Buffering
		1	Double Packet Buffering
3:0 (R/W)	SZ	Maximum Packet Size. The USB_RXFIFOSZ.SZ bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$ If the USB_RXFIFOSZ.DPB is cleared, the FIFO size is FIFOSZ from this formula. If the USB_RXFIFOSZ.DPB is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32

Table 27-88: USB_RXFIFOSZ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

Software Reset Register

The `USB_SOFT_RST` register provides reset controls for the USB controller CLK domain and XCLK domain. The USB controller PHY operates in the controller's XCLK domain, and the USB controller interface to the processor core operates in the controller's CLK domain. Note that for correct operation, both of the reset control bits (`USB_SOFT_RST.RST` and `USB_SOFT_RST.RSTX`) should always be asserted simultaneously.

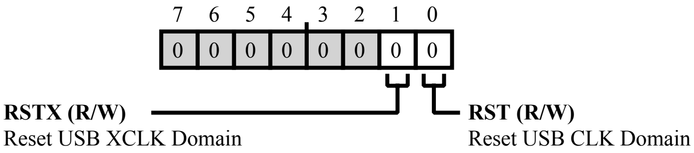


Figure 27-113: USB_SOFT_RST Register Diagram

Table 27-89: USB_SOFT_RST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RSTX	Reset USB XCLK Domain. The `USB_SOFT_RST.RSTX` bit resets logic in the USB XCLK domain. This bit is self-clearing. Note that this bit should always be asserted simultaneously with the `USB_SOFT_RST.RST` bit.	
		0	No Reset
		1	Reset USB XCLK Domain
0 (R/W)	RST	Reset USB CLK Domain. The `USB_SOFT_RST.RST` bit resets logic in the USB CLK domain. This bit is self-clearing. Note that this bit should always be asserted simultaneously with the `USB_SOFT_RST.RSTX` bit.	
		0	No Reset
		1	Reset USB CLK Domain

Testmode Register

The `USB_TESTMODE` register places the USB controller into the test mode state and can also put the USB controller into one of the test modes for high-speed operation. For more information about these modes, see the USB 2.0 specification.

Note that the `USB_TESTMODE` register is not used in normal operation. Only one of the test mode (bits 0-6) selection bits may be set at a time.

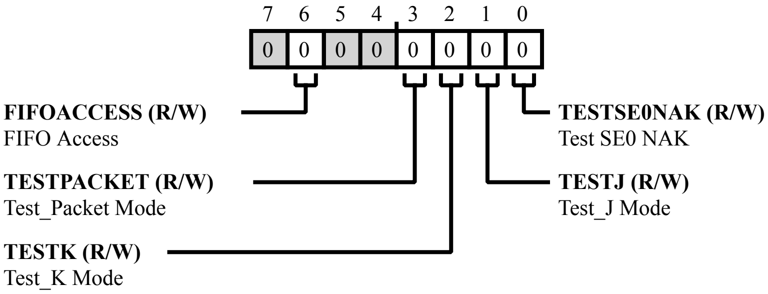


Figure 27-114: USB_TESTMODE Register Diagram

Table 27-90: USB_TESTMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	FIFOACCESS	FIFO Access. The `USB_TESTMODE` bit directs the USB controller to transfer the packet in the endpoint 0 Tx FIFO to the endpoint 0 Rx FIFO. The bit is cleared automatically.
3 (R/W)	TESTPACKET	Test_Packet Mode. The `USB_TESTMODE.TESTPACKET` bit selects Test_Packet test mode, which applies only when the USB controller is in high-speed mode. In this mode, the USB controller repetitively transmits on the bus a 53-byte test packet, whose form is defined in the USB 2.0 Specification, Section 7.1.20. Note that the test packet has a fixed format and must be loaded into the endpoint 0 FIFO before this test mode is entered.
2 (R/W)	TESTK	Test_K Mode. The `USB_TESTMODE.TESTK` bit selects Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1 (R/W)	TESTJ	Test_J Mode. The `USB_TESTMODE.TESTJ` bit selects Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0 (R/W)	TESTSE0NAK	Test SE0 NAK. The `USB_TESTMODE.TESTSE0NAK` bit selects Test_SE0_NAK test mode, which applies only when the USB controller is in high-speed mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

Transmit FIFO Address Register

The USB_TXFIFOADDR register sets the start address for the selected Tx FIFO for endpoints 1-11. There is one transmit FIFO address register for each endpoint, except endpoint 0. The USB_TXFIFOADDR register is indexed and selected by the USB_INDEX register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

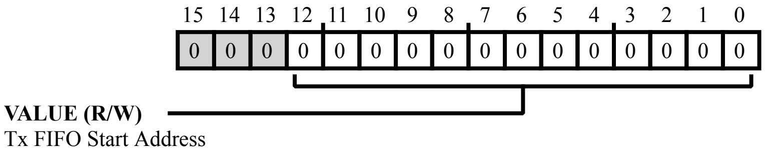


Figure 27-115: USB_TXFIFOADDR Register Diagram

Table 27-91: USB_TXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	VALUE	Tx FIFO Start Address. The USB_TXFIFOADDR.VALUE bits hold the start address of the selected endpoint FIFO (selected with the USB_INDEX register) in units of 8 bytes, according to the formula: FIFO address = USB_TXFIFOADDR.VALUE * 8

Transmit FIFO Size Register

The `USB_TXFIFOSZ` register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed access to the FIFO (packet) size selection for each Tx endpoint (except endpoint 0).

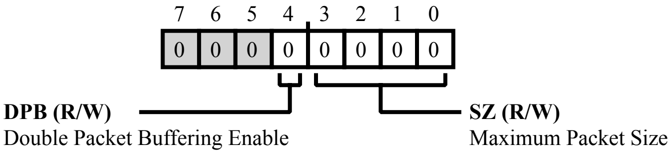


Figure 27-116: USB_TXFIFOSZ Register Diagram

Table 27-92: USB_TXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The `USB_TXFIFOSZ.DPB` bit enables double packet buffering, doubling the FIFO (packet) size selected with the `USB_TXFIFOSZ.SZ` field.	
		0	Single Packet Buffering
		1	Double Packet Buffering
3:0 (R/W)	SZ	Maximum Packet Size. The `USB_TXFIFOSZ.SZ` bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$ If the `USB_TXFIFOSZ.DPB` is cleared, the FIFO size is FIFOSZ from this formula. If the `USB_TXFIFOSZ.DPB` is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048

Table 27-92: USB_TXFIFOSZ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

VBUS Control Register

The `USB_VBUS_CTL` controls USB controller VBUS-related features.

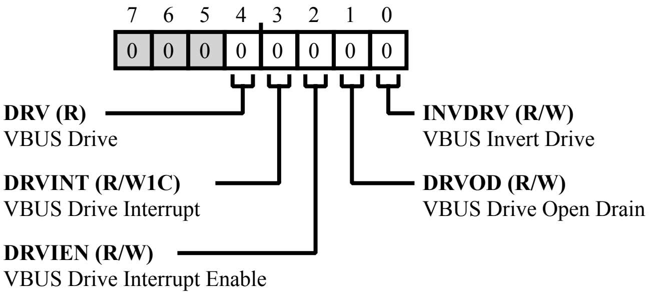

7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
DRV (R)
VBUS Drive
DRVINT (R/W1C)
VBUS Drive Interrupt
DRVIEN (R/W)
VBUS Drive Interrupt Enable
INVDRV (R/W)
VBUS Invert Drive
DRVOD (R/W)
VBUS Drive Open Drain

Figure 27-117: USB_VBUS_CTL Register Diagram

Table 27-93: USB_VBUS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	DRV	VBUS Drive. The `USB_VBUS_CTL.DRV` bit indicates the state of the UTMI+ DrvVBUS signal from the USB controller.
3 (R/W1C)	DRVINT	VBUS Drive Interrupt. The `USB_VBUS_CTL.DRVINT` bit indicates the state of the DrvVBUSInt interrupt.
2 (R/W)	DRVIEN	VBUS Drive Interrupt Enable. The `USB_VBUS_CTL.DRVIEN` bit enables the DrvVBUS interrupt.
1 (R/W)	DRVOD	VBUS Drive Open Drain. The `USB_VBUS_CTL.DRVOD` selects whether the DrvVBUS output is open drain.
0 (R/W)	INVDRV	VBUS Invert Drive. The `USB_VBUS_CTL.INVDRV` bit selects whether the DrvVBUS output is inverted.

VBUS Pulse Length Register

The `USB_VPLEN` register defines the duration of the VBUS pulsing charge for SRP initiation.

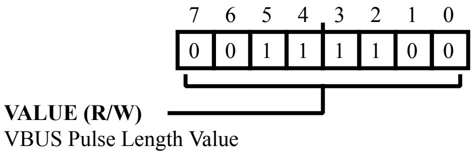


Figure 27-118: USB_VPLEN Register Diagram

Table 27-94: USB_VPLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	VBUS Pulse Length Value. The `USB_VPLEN.VALUE` bits sets the duration of the VBUS pulsing charge in units of 546.1us. The default setting corresponds to 32.77ms. Note that VBUS pulsing was removed in the OTG specification v2.0, section 5.1.4.

28 Media Local Bus (MLB)

Media Local Bus (MediaLB®) is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The MLB topology supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

The MLB module serves as an interface between the MediaLB and the processor, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. It supports up to 64 logical channels with up to 468 bytes of data per MediaLB frame. Transmit and receive data can be transferred between MediaLB and on-chip memory with DMA block transfers.

The MLB supports the MOST25, MOST50 and MOST150 standards and this document assumes familiarity with these standards. For more information, refer to the Media Local Bus specification version 4.2.

- Copyright 1998-2016 Microchip Technology Inc. All rights reserved. Portions of this chapter are included with permission from Microchip Technology, Inc.

Features

The objective of Media Local Bus (MediaLB) is to map all the MOST Network data types (transport methods) into a single low-cost, scalable, and standardized hardware interface between a MediaLB controller and at least one other MediaLB Device. The adoption of MediaLB simplifies the hardware interface, reduces the pin count, and facilitates the design of modular reusable hardware. From a software development perspective, the use of MediaLB relieves the system developer from the complexity of the MOST network, which simplifies software development and enables the design of reusable software for different applications. This simplified, standardized interface shortens time to market and makes software maintenance effortless.

- Compliant to Media Local Bus specification version 4.2
- Support MOST25, MOST50, MOST150 standards
- Support 3-pin and 6-pin modes
  - 6-pin mode supports a data rate of 2048 × FS
  - 3-pin mode support various data rates of 256 × FS, 512 × FS, 1024 × FS

- Support 64 logic channels
- Dedicated PLL for clock recovery and phase alignment
- Dedicated pins for 6-pin mode (LVDS)
- Shared pins for 3-pin mode
- Dedicated internal RAM for data buffering and channel table
- Recovered clock (after division) available out on a shared pin for DAI

NOTE: Only one MLB interface (3 pin or 6 pin mode) can be active at any given time.

MLB Definitions

The following are standard MLB and MOST terms that are used in this chapter.

AGU

Address Generation Units. To access a particular HBI channel the HC must first configure one of two HBI address generation units. AGUs can be configured by writing the HBI command registers, HCMD0 and HCMD1.

CAT

Channel Allocation Table. The Channel Allocation Table (CAT) is comprised of 16 CTR entries. Each 16-bit CAT entry represents a logical connection to or from a transmit/receive device (for example MediaLB or HBI channel).

CDT

Channel Descriptor Table.

FCE

Flow Control Enable bit. The FCE bit is used by MediaLB isochronous Rx channels only.

HBI

Host Bus Interface. The HBI block provides 16-bit parallel slave access to all MOST channels and data types for the external Host Controller (HC). The HBI supports up to 64 independent channels.

HC

Host Controller (external).

HCMDx

HBI Command registers.

HSTSx

HBI status registers.

MFE

Multi-Frame per sub-buffer enable bit. The MFE bit is used by MediaLB synchronous channels only.

PML

Packet Message Length.

Clocking

The MLB controller provides an external clock pin—the media local bus clock. The MLB controller generates the clock. It is synchronized to the MOST network and provides the timing for the entire MLB interface at FS = 48 kHz.

Functional Description

The ***Media LB Block Diagram*** figure shows the MLB high-level architecture. The MLB core serves as an interface between the MediaLB and the processor, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. The MLB core has the following responsibilities.

- Transmit commands and data when functioning as the transmitting device associated with a *Channel Address*
- Receive data and transmit Rx Status responses when functioning as the receiving device associated *Channel Address*
- MLB lock detection
- System channel command handling

The MediaLB device can function as either a MediaLB 3-pin interface (single-ended) or MediaLB 6-pin interface (differential) but only one interface can be active at a time. The MediaLB interfaces are capable of exchanging data at speeds up to 1024 × Fs in 3-pin mode or 2048 × Fs in 6-pin mode.

A set of physical channels for exchanging data over the MediaLB bus is supported. These physical channels (4 bytes in length, or a quadlet) can be grouped into logical channels, where each logical channel is referenced using a channel address and represents a uni-directional datapath between a specific MediaLB device transmitting the data and the MediaLB device(s) receiving the data. The MediaLB 6-pin interface provides support for up to 468 bytes of data per frame. The logical channels, configured by system software, can be any combination of channel types (synchronous, asynchronous, isochronous, or control) and direction (transmit or receive).

ADSP-SC58x MLB Register List

The MediaLB Device Interface Macro 2 (MediaLB DIM 2), also referred to as OS62420, implements the required functionality of a Media Local Bus (MediaLB) device. This logic serves as an interface between the inter-chip MediaLB bus and a customer IC, implementing the physical- and link-layer requirements outlined in the MediaLB Specification.

Table 28-1: ADSP-SC58x MLB Register List

Name	Description
MLB_ACMR0	Peripheral Channel Mask 0 Register
MLB_ACMR1	Peripheral Channel Mask 1 Register
MLB_ACSR0	Peripheral Channel Status 0 Register
MLB_ACSR1	Peripheral Channel Status 1 Register
MLB_ACTL	Bus Control Register
MLB_CTL0	MediaLB Control 0 Register
MLB_CTL1	Control 1 Register
MLB_GCTL	MLB Global Control Register
MLB_HCBR0	HBI Channel Busy 0 Register
MLB_HCBR1	HBI Channel Busy 1 Register
MLB_HCER0	HBI Channel Error 0 Register
MLB_HCER1	HBI Channel Error 1 Register
MLB_HCMR0	HBI Channel Mask 0 Register
MLB_HCMR1	HBI Channel Mask 1 Register
MLB_HCTL	HBI Control Register
MLB_MADR	Memory Interface Address Register
MLB_MCTL	Memory Interface Control Register
MLB_MDAT0	Memory Interface Control Data 0 Register
MLB_MDAT1	Memory Interface Control Data 1 Register
MLB_MDAT2	Memory Interface Control Data 2 Register
MLB_MDAT3	Memory Interface Control Data 3 Register
MLB_MDWE0	Memory Interface Control Data Write Enable 0 Register
MLB_MDWE1	Memory Interface Control Data Write Enable 1 Register
MLB_MDWE2	Memory Interface Control Data Write Enable 2 Register
MLB_MDWE3	Memory Interface Control Data Write Enable 3 Register
MLB_MIEN	Interrupt Enable Register

Table 28-1: ADSP-SC58x MLB Register List (Continued)

Name	Description
`MLB_MS0`	Channel Status 0 Register
`MLB_MS1`	Channel Status 1 Register
`MLB_MSD`	System Data Register
`MLB_MSS`	System Status Register
`MLB_PCTL0`	MediaLB 6-pin Control 0 Register

ADSP-SC58x MLB Interrupt List

Table 28-2: ADSP-SC58x MLB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
162	MLB0_INT0	MLB0 Interrupt 0 channel 0-31		
163	MLB0_INT1	MLB0 Interrupt 1 channel 32-63		
164	MLB0_STAT	MLB0 Status		

Media LB Protocol

The MediaLB topology supports communication among all MediaLB devices, including the MediaLB controller. The bus interface consists of a uni-directional line for clock (MLBC), a bidirectional line for signal information (MLBS), and a bidirectional line for data transfer (MLBD). The MediaLB topology supports one controller connected to one or more devices, where the controller is the interface between the MediaLB devices and the MOST network.

The MediaLB controller includes MediaLB device functionality, and also generates the MediaLB clock (MLBC) that is synchronized to the MOST Network. This generated clock provides the timing for the entire MediaLB interface. The MLBS line is a multiplexed signal which carries channel addresses generated by the MediaLB controller, as well as command and RxStatus bytes from MediaLB devices. The MLBD line is driven by the transmitting MediaLB device and is received by all other MediaLB devices, including the MediaLB controller. The MLBD line carries the actual data (synchronous, asynchronous, control, or isochronous).

Once per MOST network frame, the MLB controller generates a unique frame sync pattern on the `MLB_SIG` line. The end of the frame sync pattern defines the byte boundary and the channel boundary for the `MLB_SIG` and `MLB_DAT` lines of all MLB devices.

The MLB controller manages the arbitration for all the channels on the MLB and grants bandwidth for all the MLB devices. An MLB *physical channel* is defined as four bytes wide, or a quadlet. Physical channels can be grouped into multiple quadlets (which do not have to be consecutive) to form an MLB *logical channel*, which is defined by a unique channel address.

As shown in ***MLB Data Structure***, the MLB controller initiates communication by sending out a channel address on the `MLB_SIG` line for each physical channel. The channel address indicates which MLB device is transmitting and which MLB devices are receiving in the following physical channel. Therefore, four bytes after the controller outputs the channel address on the `MLB_SIG` line, the transmitting device outputs a command byte command on the `MLB_SIG` line and outputs the respective data on the `MLB_DAT` line, concurrently. The MLB command byte contains the type of data currently being transmitted (for example synchronous, asynchronous or control).

The MLB device receiving the channel data outputs a status byte, RxStatus, on the `MLB_SIG` line immediately after the transmitting device outputs the command byte. The status response can indicate that the receiving device is busy and cannot receive the data at present, or the device is ready to receive the data. Since synchronous stream data is sent in a broadcast fashion, receiving devices cannot return a busy status and should not drive RxStatus onto the `MLB_SIG` line.

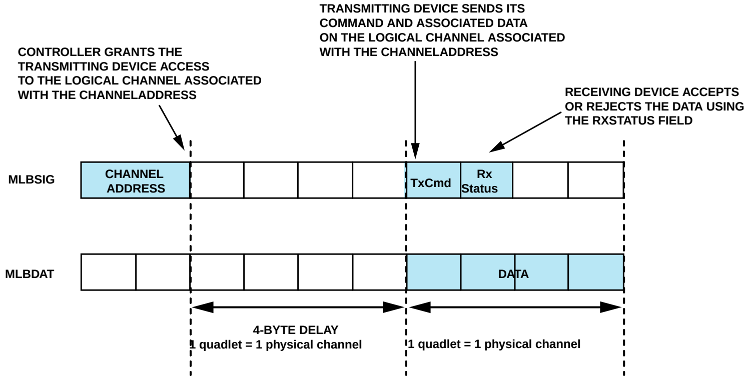

TRANSMITTING DEVICE SENDS ITS COMMAND AND ASSOCIATED DATA ON THE LOGICAL CHANNEL ASSOCIATED WITH THE CHANNELADDRESS
CONTROLLER GRANTS THE TRANSMITTING DEVICE ACCESS TO THE LOGICAL CHANNEL ASSOCIATED WITH THE CHANNELADDRESS
RECEIVING DEVICE ACCEPTS OR REJECTS THE DATA USING THE RXSTATUS FIELD
MLBSIG
CHANNEL ADDRESS
TxCmd
Rx Status
MLBDAT
DATA
4-BYTE DELAY
1 quadlet = 1 physical channel
1 quadlet = 1 physical channel

Figure 28-1: MLB Data Structure

MLB Architectural Concepts

The following sections and the Figure 28-2 Media LB Block Diagram provide information about the MLB architecture.

MediaLB Block Diagram

The ***Media LB Block Diagram*** shows the various blocks within the interface its connections to the processor.

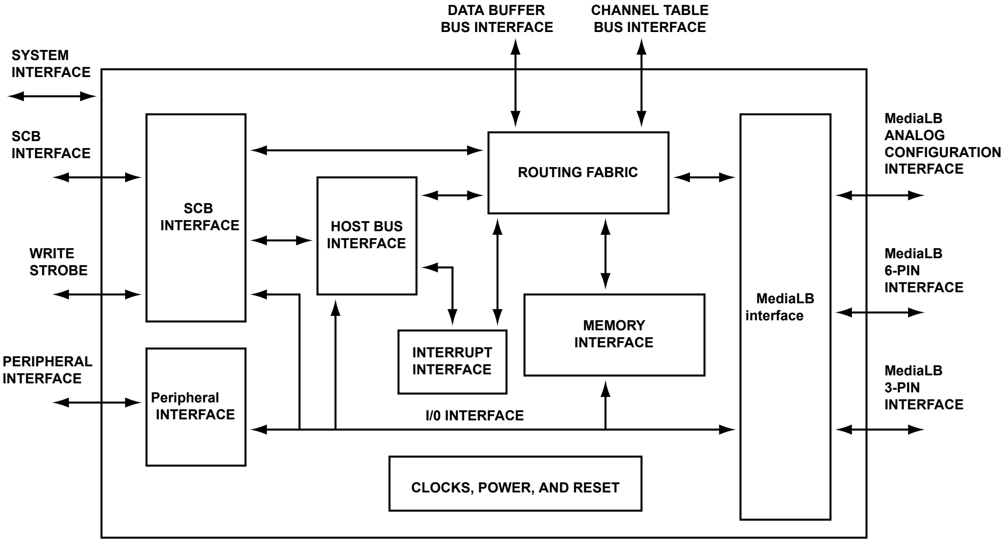

DATA BUFFER BUS INTERFACE
CHANNEL TABLE BUS INTERFACE
SYSTEM INTERFACE
SCB INTERFACE
WRITE STROBE
PERIPHERAL INTERFACE
SCB INTERFACE
HOST BUS INTERFACE
ROUTING FABRIC
INTERRUPT INTERFACE
MEMORY INTERFACE
Peripheral INTERFACE
I/0 INTERFACE
CLOCKS, POWER, AND RESET
MediaLB interface
MediaLB ANALOG CONFIGURATION INTERFACE
MediaLB 6-PIN INTERFACE
MediaLB 3-PIN INTERFACE

Figure 28-2: Media LB Block Diagram

MediaLB Interface

The Media Local Bus (MediaLB) block supports both a MediaLB 3-pin interface and MediaLB 6-pin interface; however, only one MediaLB interface can be active at any given time. Both MediaLB interfaces provide real-time access to all network data types - synchronous, asynchronous, control, and isochronous data.

- MediaLB 3-pin interface – Supports the MediaLB protocol for single-ended 3-pin mode, with a maximum data rate of 1024 FS (49.152 MHz at FS =48 kHz).
- MediaLB 6-pin interface – Supports the MediaLB protocol for high-speed differential 6-pin mode, with a maximum data

  maximum data rate of 2048 FS (98.304 MHz at FS =48 kHz).

MediaLB Channel Address to Logical Channel Mapping

The MediaLB channel addresses are mapped to the logical channels as shown in the ***MediaLB Channel Address to Logical Channel Mapping*** table.

Table 28-3: MediaLB Channel Address to Logical Channel Mapping

Channel Address	Logical Address
0x0002	1
0x0004	2
0x0006	3
...	...

Table 28-3: MediaLB Channel Address to Logical Channel Mapping (Continued)

Channel Address	Logical Address
0x007C	62
0x007E	63
0x01FE	0 (Logical channel 0 is the system channel and is reserved)

Routing Fabric

The Routing Fabric (RF) block manages the flow of data between the MediaLB port and the HBI port. It manages accessing the Channel Table RAM (CTR) and Data Buffer RAM (DBR), which are explained in the following subsections. The routing fabric uses channel descriptors (stored in the CTR) to manage access to dynamic buffers in the DBR.

Data Buffer RAM

The Data Buffer RAM (DBR) is an 8-bit x 16k entries deep, single-port synchronous SRAM and provides dynamic circular buffering between the transmit and receive devices. The size and location of each data buffer is defined by software in the Channel Descriptor Table (CDT), which is located in the Channel Table RAM (CTR), described in following sections.

Channel Table RAM

The Channel Table RAM (CTR) is a 128-bit x 144-entry table that allows system software to dynamically configure channel routing and allocate data buffers in the DBR. The CTR is logically divided into three tables:

- Channel Descriptor Tables
- AHB Descriptor Table (ADT)
- Channel Allocation Table

Address Mapping

The ***CTR Address Mapping*** table shows the address mapping for the CTR.

Table 28-4: CTR Address Mapping

Label	Address	Bits [127:96]	Bits [95:64]	Bits [63:32]	Bits [31:0]
Channel Descriptor Table (CDT)					
CDT	0x00	CDT0[127:0], CL = 0			
	0x01	CDT1[127:0], CL = 1			
	0x02	CDT2[127:0], CL = 2			
	...				
	0x3D	CDT61[127:0], CL = 61			
	0x3E	CDT62[127:0], CL = 62			

Table 28-4: CTR Address Mapping (Continued)

<table>
<tr><th>Label</th><th>Address</th><th colspan="2">Bits [127:96]</th><th colspan="2">Bits [95:64]</th><th colspan="2">Bits [63:32]</th><th colspan="2">Bits [31:0]</th></tr>
<tr><td></td><td>0x3F</td><td colspan="8">CDT63[127:0], CL = 63</td></tr>
<tr><td colspan="10">AHB Descriptor Table (ADT)</td></tr>
<tr><td rowspan="7">ADT</td><td>0x40</td><td colspan="8">ADT0[127:0], CL = 0</td></tr>
<tr><td>0x41</td><td colspan="8">ADT1[127:0], CL = 1</td></tr>
<tr><td>0x42</td><td colspan="8">ADT2[127:0], CL = 2</td></tr>
<tr><td>...</td><td colspan="8"></td></tr>
<tr><td>0x7D</td><td colspan="8">ADT61[127:0], CL = 61</td></tr>
<tr><td>0x7E</td><td colspan="8">ADT62[127:0], CL = 62</td></tr>
<tr><td>0x7F</td><td colspan="8">ADT63[127:0], CL = 63</td></tr>
<tr><td colspan="6">Channel Allocation Table (CAT)</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">CAT for MediaLB</td><td>0x80</td><td>CAT7</td><td>CAT6</td><td>CAT5</td><td>CAT4</td><td>CAT3</td><td>CAT2</td><td>CAT1</td><td>CAT0</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>0x87</td><td>CAT63</td><td>CAT62</td><td>CAT61</td><td>CAT60</td><td>CAT59</td><td>CAT58</td><td>CAT57</td><td>CAT56</td></tr>
<tr><td rowspan="3">CAT for HBI[*1]</td><td>0x88</td><td>CAT71</td><td>CAT70</td><td>CAT69</td><td>CAT68</td><td>CAT67</td><td>CAT66</td><td>CAT65</td><td>CAT64</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>0x8F</td><td>CAT127</td><td>CAT126</td><td>CAT125</td><td>CAT124</td><td>CAT123</td><td>CAT122</td><td>CAT121</td><td>CAT120</td></tr>
</table>

[*1] A fixed relationship exists between ADT entries and HBI CAT entries. When using HBI channel 0 (CAT64), program ADT0. When using HBI channel 1 (CAT65), program ADT1, and so on.

Channel Allocation Table

The *Channel Allocation Table (CAT)* table is comprised of 16 CTR entries (addresses 0x80 0x8F) as shown in the *CTR Entry Map* table. Each 16-bit CAT entry represents a logical connection to or from a transmit or receive device. (for example, MediaLB channel). All entries are indexed according to a fixed physical address assigned to every RX/TX channel as shown in the *CAT Entry Formats* table. The value stored in a CAT entry includes a 6-bit connection label, which provides a pointer to the CDT. To complete a logical channel and form a routing connection, system software must assign the same connection label to both the RX and TX channels.

Table 28-5: CAT Entry Map

Peripheral	TX Channels	RX Channels	CAT Start Index	CAT End Index	Entries
MediaLB	0 to 64	64 TX Channels	0	63	64
HBI	0 to 64	64 TX Channels	64	127	64

The format of a full CAT entry is shown in the *CAT Entry Formats* table, with field descriptions described in the *CAT Field Definitions* table. All reserved bits of a CAT entry field should be written as zero.

Table 28-6: CAT Entry Formats

Channel Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Isochronous	rsvd	FCE	rsvd	RNW	CE	CT[2:0] = 011			rsvd		CL[5:0]					
Asynchronous	rsvd		MT	RNW	CE	CT[2:0] = 010			rsvd		CL[5:0]					
Control	rsvd		MT	RNW	CE	CT[2:0] = 001			rsvd		CL[5:0]					
Synchronous	rsvd	MFE	MT	RNW	CE	CT[2:0] = 000			rsvd		CL[5:0]					

Table 28-7: CAT Field Definitions

Field	Description
CL[5:0]	Connection Label (offset into CDT)
CT[2:0]	Channel Type (others) 111 = Reserved 110 = Reserved 101 = Reserved 100 = Reserved 011 = Isochronous 010 = Asynchronous 001 = Control 000 = Synchronous
CE	Channel Enable. 0 = Disabled, 1 = Enabled
RNW	Read Not Write. 0 = Write, 1 = Read
MT	Mute Enable. 0 = Disabled When set for synchronous channels, the MT bit forces RX channels to write zeros into the channel data buffer, and TX channels to output zeros on the physical interface. When set for asynchronous and control channels, the MT bit causes DMA to halt at a packet boundary. Not valid for isochronous channels.
FCE	Flow Control Enable. 0 = Disabled, 1 = Enabled The FCE bit is used by MediaLB isochronous RX channels only.
MFE	Multi-Frame per sub-buffer enable. 0 = Disabled, 1 = Enabled The MFE bit is used by MediaLB synchronous channels only.
rsvd	Reserved. Software writes a 0 to all reserved bits when the entry is initialized. These bits are read-only after initialization.

Channel Set Up

Data direction is in reference to the DBR. The data direction of CAT entries corresponding to the same channel is reversed for the HBI CAT and the MediaLB CAT.

- For a Tx channel (from the HC to the MediaLB interface):
    - HBI CAT entry: RNW = 0 (write)

  - MediaLB CAT entry: RNW = 1 (read)
- Conversely, for an Rx channel (data from MediaLB to HC):
  - HBI CAT entry: RNW = 1 (read)
  - MediaLB CAT entry: RNW = 0 (write)

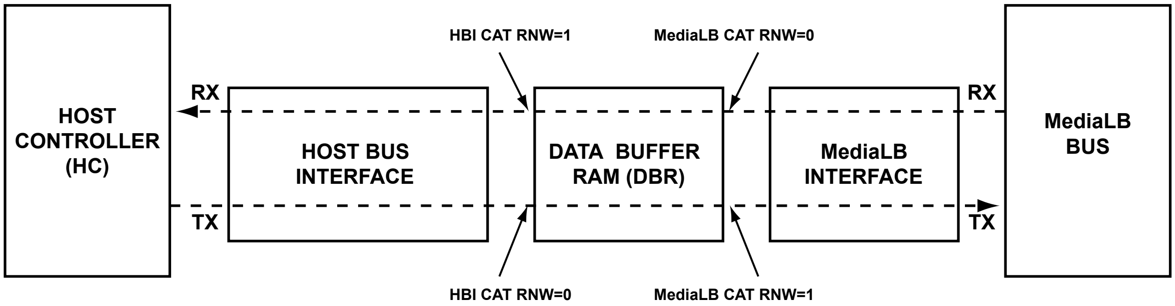

HOST CONTROLLER (HC)
RX
TX
HOST BUS INTERFACE
HBI CAT RNW=1
HBI CAT RNW=0
DATA BUFFER RAM (DBR)
MediaLB CAT RNW=0
MediaLB CAT RNW=1
MediaLB INTERFACE
RX
TX
MediaLB BUS

Figure 28-3: DBR Directional Relationship

Channel Descriptor Tables

The Channel Descriptor Table (CDT) is comprised of 64 CTR entries (addresses 0x00 - 0x3F), as shown in the Table 28-4 CTR Address Mapping table. Each 128-bit CDT entry (also referred to as a channel descriptor) is referenced by a connection label and contains information about a data buffer in the DBR (for example buffer size, address pointers). The format of each CDT entry is dependent on the channel type (synchronous, isochronous, asynchronous, or control).

NOTE: All reserved channel descriptor bits must be written to 0 by software when initialized.

Synchronous Channel Descriptors

The format and field definitions for a synchronous CDT entry are shown in the *Synchronous CDT Entry Format* and *Synchronous CDT Entry Field Definitions* tables.

Table 28-8: Synchronous CDT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WSBC[1:0]		Reserved													
16	RSBC[1:0]		Reserved													
32	Reserved															
48	Reserved															
64	WSTS[3:0]				WPTR[11:0]											
80	RSTS[3:0]				RPTR[11:0]											
96	Reserved				BD[11:0]											
112	Reserved		BA[13:0]													

Table 28-9: Synchronous CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. BD = 4 m bpf 1 where: m = frames per sub-buffer (for MFE = 0, m = 1) bpf = bytes per frame.	RW
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA read address = BA + WPTR.	RWU
RSBC	Read Sub-buffer Counter	Software initializes to 0, hardware updates. Counts the read sub-buffer offset. DMA uses for pointer management.	RWU
WSBC	Write Sub-buffer Counter	Software initializes to 0, hardware updates. Counts the write sub-buffer offset. DMA uses for pointer management.	RWU
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: xxx0 = normal operation (no mute) xxx1 = normal operation (mute) xx0x = idle	RWU
WSTS	Write Status	Software initializes to 0, hardware updates. WSTS States: xxx0 = normal operation (no mute) xxx1 = normal operation (mute) xx0x = idle 1xxx = command protocol error	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

Isochronous Channel Descriptors

The format and field definitions for a synchronous CDT entry are shown in the ***Isochronous Entry Formats*** and ***Isochronous CDT Entry Field Definitions*** tables.

Table 28-10: Isochronous Entry Formats

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Reserved															
16	Reserved															
32	Reserved							BS[8:0]								
48	Reserved															
64	WSTS[3:0]			WPTR[12:0]												
80	RSTS[3:0]			RPTR[12:0]												
96	Reserved			BD[12:0]												
112	BF	Rsvd	BA[13:0]													

Table 28-11: Isochronous CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR.	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. Isochronous buffers must be large enough to hold at least 3 blocks (packets) of data. BD Must be an integer multiple of blocks.	RW
BF	Buffer Full	Software initializes to 0, hardware updates. DMA write hardware sets BF when the buffer is full. DMA read hardware clears BF when the buffer is empty. BF is valid only when buffer is full or empty, otherwise ignore.	RWU
BS	Block Size	BS defines when to begin the DMA to the data buffer. BS = buffer block size in bytes 1. For RX channels, the DMA writes start when the number of empty bytes in the data buffer the block size. For TX channels, the DMA reads start when the number of valid bytes in the data buffer the block size.	RWU
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA write address = BA + WPTR.	RWU

Table 28-11: Isochronous CDT Entry Field Definitions (Continued)

Field	Description	Details	Access
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: xx1 = active xx0 = idle	RWU
WSTS	Write Status	Software initializes to 0, hardware updates. WSTS States: xxx0 =active xxx1 = idle xx0x = command protocol error 1xxx = buffer overflow (FCE = 0 only)	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

Asynchronous/Control Channel Descriptors

The format and field definitions for an Asynchronous/Control CDT entry are shown in the ***Asynchronous/Control CDT Entry Format*** and ***Asynchronous/Control CDT Entry Field Definitions*** tables.

Table 28-12: Asynchronous/Control CDT Entry Format

<table>
<tr><th>Bit Offset</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td colspan="5">WPC[4:0]</td><td colspan="11">Reserved</td></tr>
<tr><td>16</td><td colspan="5">RPC[4:0]</td><td colspan="11">Reserved</td></tr>
<tr><td>32</td><td>Rsvd</td><td colspan="3">WPC[7:5]</td><td colspan="12">Reserved</td></tr>
<tr><td>48</td><td>Rsvd</td><td colspan="3">RPC[7:5]</td><td colspan="12">Reserved</td></tr>
<tr><td>64</td><td colspan="4">WSTS[3:0]</td><td colspan="12">WPTR[11:0]</td></tr>
<tr><td>80</td><td colspan="4">RSTS[3:0]</td><td colspan="12">RPTR[11:0]</td></tr>
<tr><td>96</td><td>RSTS[4]</td><td>WSTS[4]</td><td colspan="2">Rsvd</td><td colspan="12">BD[11:0]</td></tr>
<tr><td>112</td><td colspan="2">Rsvd</td><td colspan="14">BA[13:0]</td></tr>
</table>

Table 28-13: Asynchronous/Control CDT Entry Field Definitions

Field	Description	Details	Access
BA	Buffer Base Address	Can start at any byte in the 16k DBR.	RW
BD	Buffer Depth	BD = size of buffer in bytes 1. Buffer end address = BA + BD. BD = max packet length 1.	RW

Table 28-13: Asynchronous/Control CDT Entry Field Definitions (Continued)

Field	Description	Details	Access
RPC	Read Packet Count	Software initializes to 0, hardware updates. Used with RPC, RPTR and WPTR to determine if the buffer is empty or full.	RWU
WPC	Write Packet Count	Software initializes to 0, hardware updates. Used with RPC, RPTR and WPTR to determine if the buffer is empty or full.	RWU
RPTR	Read Pointer	Software initializes to 0, hardware updates. Counts the read address offset within a buffer. DMA read address = BA + RPTR.	RWU
WPTR	Write Pointer	Software initializes to 0, hardware updates. Counts the write address offset within a buffer. DMA read address = BA + WPTR.	RWU
RSTS	Read Status	Software initializes to 0, hardware updates. RSTS States: x0x00 = idle xx1xx = *ReceiverProtocolError* response received from RX device 1xxxx = *ReceiverBreak* command received from RX device	RWU
WSTS	Write Status	Software initializes to 0, hardware updates. Status States (only valid for DMA pointers associated with the MLB block, not HBI block pointers): xxx0 = idle xxx1 = command protocol error detected xx0x = *AsyncBreak*/*ControlBreak* command received from TX device	RWU
Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.			RWU

AHB Descriptor Table (ADT)

The AHB block manages data exchange between local channel data buffers within MLB module and the system memory buffer. To support system memory buffering, a ping-pong memory structure is implemented on a per channel basis using 128-bit descriptors for AHB Descriptor Table (ADT) entries. The Table 28-4 CTR Address Mapping table provides a complete address map of the CTR, including the location of the ADT.

Each logical channel is assigned a separate 128-bit descriptor, defining the data buffers in the system memory used by the DMA interface for that channel. The descriptors are stored at fixed addresses in the CTR as described in previous section. The ***ADT Field Definitions*** table provides an overview of field definitions for ADT entries.

Table 28-14: ADT Field Definitions

Field	No. of Bits	Description	Access
CE	1	Channel Enable. 0 = Disabled	RW, U
LE	1	Endianness Select. 0 = Big Endian, 1 = Little Endian	RW
PG	1	Page pointer. Software initializes to 0, hardware writes thereafter. 0 = Ping buffer, 1 = Pong buffer	RW, U
RDY1	1	Buffer Ready bit for ping buffer page. 0 = Not ready, 1 = Ready	RW
RDY2	1	Buffer Ready bit for pong buffer page. 0 = Not ready, 1 = Ready	RW
DNE1	1	Buffer Done bit for ping buffer page. 0 = Not done, 1 = Done	R, U, c0
DNE2	1	Buffer Done bit for pong buffer page. 0 = Not done, 1 = Done	R, U, c0
ERR1	1	Error Response detected for ping buffer page. 0 = No error, 1 = Error	R, U, c0
ERR2	1	Error Response detected for pong buffer page. 0 = No error, 1 = Error	R, U, c0
PS1	1	Packet Start bit for ping buffer page. 0 = No packet start, 1 = Packet start Reserved for synchronous and isochronous channels.	RW, U both TX and RX
PS2	1	Packet Start bit for pong buffer page. 0 = No packet start, 1 = Packet start Reserved for synchronous and isochronous channels.	RW, U both TX and RX
MEP1	1	Most Ethernet Packet indicator for ping buffer page. 0 = Not MEP, 1 = MEP. MEP1 only valid for the first page of a segmented buffer. Reserved for control synchronous and isochronous channels.	Rsvd for TX, R, U, c0 for RX
MEP2	1	Most Ethernet Packet indicator for pong buffer page. 0 = Not MEP, 1 = MEP. MEP2 only valid for the first page of a segmented buffer. Reserved for control synchronous and isochronous channels.	Rsvd for TX, R, U, c0 for RX
BD1	11 to 13	Buffer Depth for ping buffer page. 11 or 12 bits for asynchronous and control channels. 13 bits for synchronous and isochronous channels.	RW
BD2	11 to 13	Buffer Depth for pong buffer page. 11 or 12 bits for asynchronous and control channels. 13 bits for synchronous and isochronous channels.	RW
BA1	32	Buffer Base Address for ping buffer page	RW
BA2	32	Buffer Base Address for pong buffer page.	RW
Reserved	varies	Reserved. Software writes a 0 to all these bits when the entry is initialized. Reserved bits are RO after initialization.	RW, U

The ***Ping-Pong System Memory Structure*** figure shows that this system memory structure is similar for all channel types and shows the relationship between the BAn, BDn, and PG descriptor fields.

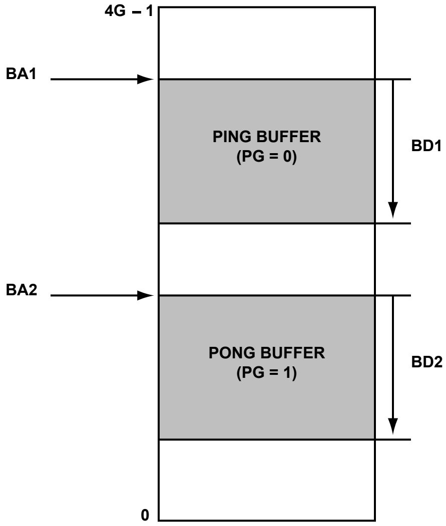


Figure 28-4: Ping-Pong System Memory Structure

Each ADT entry (also referred to as a *Channel Descriptor)* holds a 32-bit BAn field which defines the start of each ping or pong buffer within system memory. The BDn field is used to indicate the size for the respective ping or pong page. The maximum size is 2k entries for asynchronous and control channels and 8k entries for isochronous and synchronous channels.

Synchronous Channel Descriptors

The synchronous buffering scheme allows each ping or pong buffer to contain a single frame or a multiple number of frames. For this reason, the synchronous buffer depth (BDn) must be defined in terms of an integer number (n), frames per sub-buffer (m) and bytes per frame (bpf) of data (for example BDn = n m bpf 1). The ***Synchronous ADT Entry Format*** table shows the format for a synchronous ADT entry. The field definitions are defined in the ***ADT Field Definitions*** table. Each synchronous channel buffer can be up to 8k-bytes deep.

Table 28-15: Synchronous ADT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	BD1[12:0]												
48	RDY2	DNE2	ERR2	BD2[12:0]												
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Isochronous Channel Descriptors

The isochronous buffering scheme allows each ping or pong buffer to contain a single block or a multiple number of blocks. For this reason, the isochronous buffer depth (BDn) must be defined in terms of an integer number (n) and block size (BS) (for example BDn = n (BS + 1) 1).

The ***Isochronous ADT Entry Format*** table shows the format for an isochronous ADT entry. The field definitions are defined in the ***ADT Field Definitions*** table. Each isochronous channel buffer can be up to 8k-bytes deep.

Table 28-16: Isochronous ADT Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	BD1[12:0]												
48	RDY2	DNE2	ERR2	BD2[12:0]												
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Asynchronous and Control Channel Descriptors

Every asynchronous and control packet adheres to the Port Message Protocol (PMP), which designates the first two bytes of each packet as the packet length (PML). Each packet must be no more than 2048-bytes (PML 2048).

Software must set the buffer ready bit (RDYn) for each buffer as it programs the DMA. As hardware processes each buffer, it sets the done bit (DNEn) and generates an interrupt to inform HC. When hardware finishes processing a buffer, it can begin processing another buffer if RDYn is set. The application is responsible for setting up and configuring the channel buffer descriptor prior to every DMA access on the channel.

Two packet buffering modes are supported by hardware for programming the DMA, single-packet mode (`MLB_ACTL.MPB` =0) and multiple-packet mode (`MLB_ACTL.MPB` =1). The MPB is written prior to enabling the channel DMA.

Single Packet Mode. The single-packet mode asynchronous and control buffering scheme supports a maximum of one packet per buffer (for example, ping or pong). Both non-segmented and segmented data packets are allowed while using single-packet mode. Non-segmented packets are exchanged when only one buffer (for example, ping or pong) is needed for packet transfer. Segmented packets are exchanged when a single packet is too long for one buffer and the packet must span multiple buffers.

The ***Single-Packet Asynchronous and Control Entry Format*** table shows the format for single-packet mode asynchronous and control ADT entries. The field definitions are defined in the ***ADT Field Definitions*** table.

Table 28-17: Single-Packet Asynchronous and Control Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	PS1	MEP1	BD1[10:0]										
48	RDY2	DNE2	ERR2	PS2	MEP2	BD2[10:0]										
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Multiple Packet Mode. The multiple-packet mode asynchronous and control buffering scheme supports more than one packet per system memory buffer, as shown in the ***Asynchronous/Control CDT Entry Format*** table. Multiple-packet mode reduces the interrupt rate for packet channels at the cost of increasing buffering and latency.

For TX packet channels in multiple-packet mode, software sets the packet start bit (PSn) for every buffer. Setting PSn informs hardware that the first two bytes of the buffer contains the port message length (PML) of the first packet. After the first packet, hardware keeps track of where packets start and end within the current buffer. Software should not write to PSn while the buffer is active (RDYn = 1 and DNEn = 0). For TX packet channels, the buffer is done (DNEn = 1) when the last byte of the last packet in the buffer is read from system memory. Software should set the buffer depth to contain the exact number of complete packets for that buffer. Segmented buffers are not supported for TX packet channels in multiple-packet mode.

NOTE: The PS1 and PS2 bits are only valid for TX channels. Set PS1 and PS2 = 1 at the start of the buffer.

Table 28-18: Multiple-Packet Asynchronous and Control Entry Format

Bit Offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CE	LE	PG	Reserved												
16	Reserved															
32	RDY1	DNE1	ERR1	PS1	BD1[11:0]											
48	RDY2	DNE2	ERR2	PS2	BD2[11:0]											
64	BA1[15:0]															
80	BA1[31:16]															
96	BA2[15:0]															
112	BA2[31:16]															

Interrupt Interface Block

The interrupt interface raises an interrupt when specific changes to HBI channel descriptors occur, including:

- For asynchronous and control read/write channels:
  - a packet is available to read in the channel buffer, or
  - sufficient empty space is available in the channel buffer to accept a requested packet write
- For isochronous read/write channels:
  - the number of valid bytes in the channel buffer exceeds the block size, or
  - the number of empty bytes in the channel buffer exceeds the block size

Operating Modes

The following sections describe the operating modes of the MLB interface. The channel type selection enables the logical channels to operate in synchronous, asynchronous, isochronous, or control channels.

NOTE: The logical channels can be any combination of channel type (for example synchronous, asynchronous, or control) and direction (transmit or receive).

Isochronous Data Exchange

An isochronous HBI channel is initially opened and synchronized with HCMD0.CMD[2:0]= 010. For isochronous channels, no further synchronization is required from the HBI perspective; however, an optional resynchronization command is available for HC flexibility. Setting HCMD0.CMD[2:0]= 011 reinitializes the address pointer within the data buffer, ensuring that subsequent data exchange with the channel is aligned at an isochronous packet boundary. When the HC must close an isochronous channel before it has read or written an entire data packet, setting HCMD0.CMD[2:0]= 000 reopens the channel without synchronizing the address pointer in the buffer. This action allows reading and writing to continue where the HC previously stopped.

For isochronous data transmission, the ***Exchanging Isochronous Data on an HBI Channel*** figure shows the flow control.

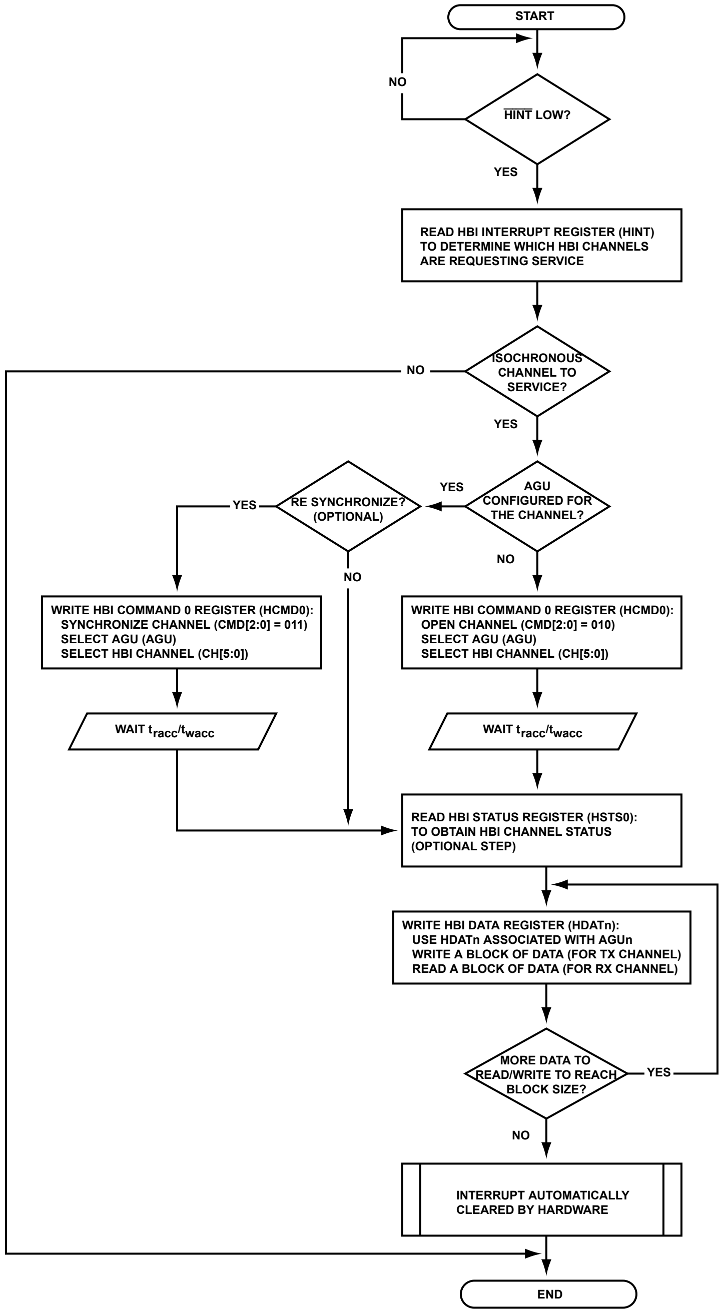

START
NO
HINT LOW?
YES
READ HBI INTERRUPT REGISTER (HINT) TO DETERMINE WHICH HBI CHANNELS ARE REQUESTING SERVICE
NO
ISOCHRONOUS CHANNEL TO SERVICE?
YES
YES
RE SYNCHRONIZE? (OPTIONAL)
YES
AGU CONFIGURED FOR THE CHANNEL?
NO
NO
WRITE HBI COMMAND 0 REGISTER (HCMD0): SYNCHRONIZE CHANNEL (CMD[2:0] = 011) SELECT AGU (AGU) SELECT HBI CHANNEL (CH[5:0])
WRITE HBI COMMAND 0 REGISTER (HCMD0): OPEN CHANNEL (CMD[2:0] = 010) SELECT AGU (AGU) SELECT HBI CHANNEL (CH[5:0])
WAIT t_racc/t_wacc
WAIT t_racc/t_wacc
READ HBI STATUS REGISTER (HSTS0): TO OBTAIN HBI CHANNEL STATUS (OPTIONAL STEP)
WRITE HBI DATA REGISTER (HDATn): USE HDATn ASSOCIATED WITH AGUn WRITE A BLOCK OF DATA (FOR TX CHANNEL) READ A BLOCK OF DATA (FOR RX CHANNEL)
MORE DATA TO READ/WRITE TO REACH BLOCK SIZE?
YES
NO
INTERRUPT AUTOMATICALLY CLEARED BY HARDWARE
END

Figure 28-5: Exchanging Isochronous Data on an HBI Channel

Asynchronous and Control Data Exchange

An asynchronous or control HBI channel is initially opened using HCMD0.CMD[2:0]= 010. Occasionally, the HC may need to close a packet channel before it has completed writing or reading the current packet of data (for example, the HC needs to use this AGU to service another HBI channel). In this case, the HC can reopen the previous channel with HCMD0.CMD[2:0]= 000. This situation allows the HC to continue writing or reading a packet from the point it left off. In this situation, the PML is already known and the packet length is not reread or rewritten by the HC.

During the reading of a packet, if the HC sets HCMD0.CMD[2:0]= 010 for the channel before the last byte of the packet is read, an internal hardware pointer is reset to the beginning of the packet buffer. This situation requires that the HC reread the PML (from HSTS1) and reread the packet data (from HDATn) from the beginning. In the same manner, if the HC resets HCMD0.CMD[2:0]= 010 for the channel before the last byte of a packet is written, an internal hardware pointer is reset to the beginning of the packet buffer. This situation requires that the HC rewrite the PML (to HCMD1) and rewrite the packet data (to HDATn) from the beginning. Any previous packet data in the buffer is overwritten.

Frame synchronization is not supported for asynchronous channels.

Synchronous Data Exchange

The MLB core provides two modes of operation; standard and multi-frame per sub-buffer which provide flexibility for implementing synchronous channels. Channels configured for standard mode require less buffer space, but have higher interrupt rates and more stringent latency requirements. Channels setup for multi-frame per sub-buffer mode require more buffer space, but have lower interrupt rates and less stringent latency requirements.

To set up a channel in multi-frame per sub-buffer mode:

1. Program the `MLB_CTL0.FCNT` bit field to select the number of frames per sub-buffer.
2. Program the CAT to enable multi-frame sub-buffering (MFE= 1) for each particular channel.
3. Set the buffer depth in the CDT: BD= 4 × m × bpf 1 where: m = frames per sub-buffer, bpf = bytes per frame.
4. Repeat for additional synchronous channels

A sample synchronous data buffer is shown in the ***Synchronous Data Buffer Structure*** figure. Each data buffer contains four sub-buffers and each sub-buffer contains space for 1 to 64 frames of data, determined by the `MLB_CTL0.FCNT` bits.

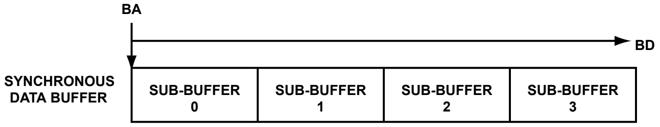


Figure 28-6: Synchronous Data Buffer Structure

Data Transfer

Two modes of operation are supported for transferring channel data between the MLB and internal memory. DMA allows the multi-channel DMA engine to manage data transfers without core intervention. Core driven mode (I/O mode) allows software to manage the transfer of data between MLB and internal memory.

NOTE: All hardware channels must use the same data transfer method. Mixed mode operation where hardware channels operate in both I/O mode and DMA mode is not supported.

DMA

The processor supports DMA mode which uses only INCR8, INCR4 and SINGLE beat bursts. Program the `MLB_ACTL.DMAMODE` = 1 to use DMA mode.

Programming Model

This section provides general guidelines for programming the MediaLB interface.

Channel Initialization

The software flow required to initialize a channel must be performed in order to ensure proper operation.

Configure the Hardware

1. Initialize the CTR and registers.
    a. Set all the CTR (CAT, CDT, and ADT) bits to 0.
    b. Set all bits of all registers to 0.
2. Configure the MediaLB interface.
    a. Select 3-pin or 6-pin MediaLB operation: `MLB_CTL0.PEN` =0 (3-pin), `MLB_CTL0.PEN` =1 (6-pin).
    b. Select MediaLB clock speed via `MLB_CTL0.CLK`.
    c. Set MediaLB enable via `MLB_CTL0.EN`.
3. Configure the HBI interface.
    a. Set `MLB_HCMR0` and `MLB_HCMR1` = 0xFFFFFFFF to activate all channels.
    b. Set the HBI enable bit: `MLB_HCTL.EN` =1.

Program the CAT and the CDT

1. Initialize all bits of the CAT to 0.
2. Select a logical channel: N = 1 – 63.
3. Program the CDT for channel N.
    a. Set the 14-bit base address (BA)
    b. Set the 12-bit or 13-bit buffer depth (BD): BD = buffer depth in bytes 1
    - For synchronous channels: (BD + 1) = 4 frames per sub-buffer (m) bytes-per-frame (bpf)
    - For isochronous channels: (BD + 1) mod (BS + 1) = 0

- For asynchronous channels: (BD + 1) max packet length (1024 for a MOST Data Packet (MDP)
- 1536 for a MOST Ethernet Packet (MEP)
- For control channels: (BD + 1) max packet length (64)

4. Program the CAT for the inbound DMA.
   a. For TX channels (to MediaLB) HBI is the inbound DMA
   b. For RX channels (from MediaLB) MediaLB is the inbound DMA
   c. Set the channel direction: RNW = 0
   d. Set the channel type: CT[2:0] = 010 (asynchronous), 001 (control), 011 (isochronous), or 000 (synchronous)
   e. Set the connection label: CL[5:0] = N
   f. If CT[2:0] = 000 (synchronous), set the mute bit (MT = 1)
   g. Set the channel enable: CE = 1
   h. Set all other bits of the CAT to 0
5. Program the CAT for the outbound DMA.
   a. For TX channels (to MediaLB) HBI is the outbound DMA
   b. For RX channels (from MediaLB) MediaLB is the outbound DMA
   c. Set the channel direction: RNW = 1
   d. Set the channel type: CT[2:0] = 010 (asynchronous), 001 (control), 011 (isochronous), or 000 (synchronous)
   e. Set the channel label: CL[5:0] = N
   f. If CT[2:0] = 000 (synchronous), set the mute bit (MT = 1)
   g. Set the channel enable: CE = 1
   h. Set all other bits of the CAT to 0
6. Repeat steps 2 through 5 to initialize all logical channels.

Program the ADT

1. Initialize all bits of the ADT to 0
2. Select a logical channel: N = 1 – 63
3. Program the AMBA AHB block ping page for channel N
   a. Set the 32-bit base address (BA1)

   b. Set the 11-bit buffer depth (BD1): BD1 = buffer depth in bytes – 1
      - For synchronous channels: (BD1 + 1) = n frames per sub-buffer (m) bytes-per-frame (bpf)
      - For isochronous channels: (BD1 + 1) mod (BS + 1) = 0
      - For asynchronous channels: 5 (BD1 + 1) 4096 (max packet length)
      - For control channels: 5 (BD1 + 1) 4096 (max packet length)
   c. For asynchronous and control Tx channels set the packet start bit (PS1) iff the page contains the start of the packet
   d. Clear the page done bit (DNE1)
   e. Clear the error bit (ERR1)
   f. Set the page ready bit (RDY1)
4. Program the AMBA AHB block pong page for channel N
   a. Set the 32-bit base address (BA2)
   b. Set the 11-bit buffer depth (BD2): BD2 = buffer depth in bytes – 1
      - For synchronous channels: (BD2 + 1) = n frames per sub-buffer (m) bytes-per-frame (bpf)
      - For isochronous channels: (BD2 + 1) mod (BS + 1) = 0
      - For asynchronous channels: 5 (BD2 + 1) 4096 (max packet length)
      - For control channels: 5 (BD2 + 1) 4096 (max packet length)
   c. For asynchronous and control TX channels set the packet start bit (PS2) iff the page contains the start of the packet
   d. Clear the page done bit (DNE2)
   e. Clear the error bit (ERR2)
   f. Set the page ready bit (RDY2)
5. Select Big Endian (LE = 0) or Little Endian (LE = 1)
6. Select the active page: PG = 0 (ping), PG = 1 (pong)
7. Set the channel enable (CE) bit for all active logical channels
8. Repeat steps 2 through 7 for all active logical channels.

Service

After initialization, each channel will require periodic servicing. Use the procedures in the following sections to service DMA and MLB interrupts and to poll for MLB system commands.

Servicing the DMA Channel Interrupts

1. Program the `MLB_ACMR0`/`MLB_ACMR1` registers to enable interrupts from all active DMA channels.
2. Select the status clear method. `MLB_ACTL.SCE` =0 (hardware clears on read), `MLB_ACTL.SCE` = 1 (software writes a 1 to clear).
3. Select 1 or 2 interrupt signals. Configure the `MLB_ACTL.SMX` bit =0 (one interrupt for channels 0 through 31 and another interrupt for channels 32 through 63 on). Configure the `MLB_ACTL.SMX` bit =1 (single interrupt for all channels).
4. Wait for an interrupt.
5. Read the `MLB_ACSR0`/`MLB_ACSR1` registers to determine which channel or channels are causing the interrupt.
6. If the `MLB_ACTL.SCE` bit =1, write the results of step 5 back to the `MLB_ACSR0` and `MLB_ACSR1` registers to clear the interrupt.
7. Select a logical channel (N = 0 – 63) with an interrupt to service.
8. Read the ADT entry for channel N to:
   a. Determine the active page (ping or pong) via the PG bit
   b. Determine which page(s) are done via the DNEn bits
   c. Determine which channels encountered an AHB error via the ERRn bit
   d. Determine which asynchronous and control Rx channel pages contain a packet start via the PSn bit (extract the PML)
9. Repeat steps 6 through 8 for all channels with pending interrupts
10. Repeat steps 4 through 9 while there are active channels.

Servicing the MediaLB Status Interrupts

1. Select the MediaLB channel status register (`MLB_MS0`/`MLB_MS1`) to be cleared by software, writing a 0 to the appropriate bits.
2. Program the `MLB_MIEN` register to enable protocol error interrupts for all active MediaLB channels. (`MLB_MIEN.CTXPE` =1, `MLB_MIEN.CRXPE` =1, `MLB_MIEN.ATXPE` =1, `MLB_MIEN.ARXPE` =1, `MLB_MIEN.SYNCPE` =1, and `MLB_MIEN.ISOCPE` =1).
3. Wait for an interrupt on the `MLB_INT0`/`MLB_INT1` signal.
4. Read the `MLB_MS0`/`MLB_MS1` registers to determine which channel(s) are causing the interrupt.
5. Read the RSTS/WSTS of the appropriate CDT(s) to determine the interrupt type.

6. Clear the RSTS/WSTS errors to ensure that the current status of channel operations is reflected in the register:

Option	Description
For synchronous RX channels	WSTS[3] = 0
For synchronous TX channels	RSTS[3] = 0
For isochronous RX channels	WSTS[2:1] = 00
For isochronous TX channels	RSTS[2:1] = 00
For asynchronous and control RX channels	WSTS[4] = 0 and WSTS[2] = 0
For asynchronous and control TX channels	RSTS[4] = 0 and RSTS[2] = 0

Polling for MediaLB System Commands

The MediaLB System status (`MLB_MSS`) register is used to detect a system command received from the MediaLB controller. The processor's peripheral automatically sends the appropriate system response to the MediaLB controller. The procedure for the application is:

1. The application periodically polls the `MLB_MSS` register.
2. Clear by writing a 0 to the appropriate bit in the `MLB_MSS` register after the application finishes the service.
3. If `MLB_MSS.SWSYSCMD` = 1, read the `MLB_MSD` register to receive the system data sent from MediaLB controller.

ADSP-SC58x MLB Register Descriptions

MediaLB Device Interface Macro 2 (MLB) contains the following registers.

Table 28-19: ADSP-SC58x MLB Register List

Name	Description
`MLB_ACMR0`	Peripheral Channel Mask 0 Register
`MLB_ACMR1`	Peripheral Channel Mask 1 Register
`MLB_ACSR0`	Peripheral Channel Status 0 Register
`MLB_ACSR1`	Peripheral Channel Status 1 Register
`MLB_ACTL`	Bus Control Register
`MLB_CTL0`	MediaLB Control 0 Register
`MLB_CTL1`	Control 1 Register
`MLB_GCTL`	MLB Global Control Register
`MLB_HCBR0`	HBI Channel Busy 0 Register
`MLB_HCBR1`	HBI Channel Busy 1 Register

Table 28-19: ADSP-SC58x MLB Register List (Continued)

Name	Description
MLB_HCER0	HBI Channel Error 0 Register
MLB_HCER1	HBI Channel Error 1 Register
MLB_HCMR0	HBI Channel Mask 0 Register
MLB_HCMR1	HBI Channel Mask 1 Register
MLB_HCTL	HBI Control Register
MLB_MADR	Memory Interface Address Register
MLB_MCTL	Memory Interface Control Register
MLB_MDAT0	Memory Interface Control Data 0 Register
MLB_MDAT1	Memory Interface Control Data 1 Register
MLB_MDAT2	Memory Interface Control Data 2 Register
MLB_MDAT3	Memory Interface Control Data 3 Register
MLB_MDWE0	Memory Interface Control Data Write Enable 0 Register
MLB_MDWE1	Memory Interface Control Data Write Enable 1 Register
MLB_MDWE2	Memory Interface Control Data Write Enable 2 Register
MLB_MDWE3	Memory Interface Control Data Write Enable 3 Register
MLB_MIEN	Interrupt Enable Register
MLB_MS0	Channel Status 0 Register
MLB_MS1	Channel Status 1 Register
MLB_MSD	System Data Register
MLB_MSS	System Status Register
MLB_PCTL0	MediaLB 6-pin Control 0 Register

Peripheral Channel Mask 0 Register

The `MLB_ACMR0` register allows control over which channel(s) generate interrupts on MLB_INT[1:0]. All of the bits in this register default to 0 (masked) so the HC must initially write to the `MLB_ACMR0` register to enable interrupts. Each bit of this register is read/write accessible.

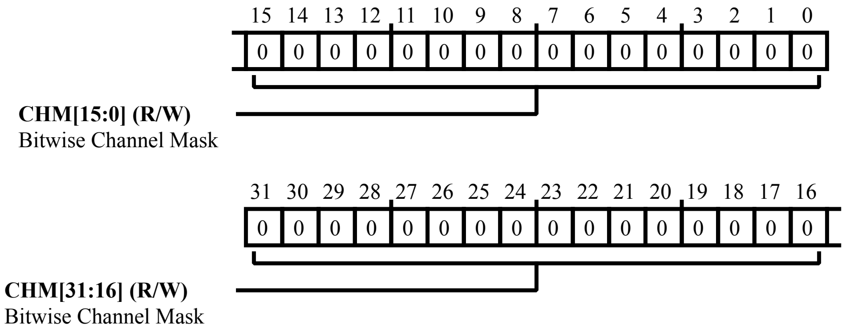

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[15:0] (R/W)
Bitwise Channel Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[31:16] (R/W)
Bitwise Channel Mask

Figure 28-7: MLB_ACMR0 Register Diagram

Table 28-20: MLB_ACMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W)	CHM	Bitwise Channel Mask. The `MLB_ACMR0.CHM` bit field masks or unmasks channels 31-0.	
		0	Mask interrupt for Channel 31:0 (bitwise; all channels shown)
		4294967295	Unmask interrupt for Channel 31:0 (bitwise; all channels shown)

Peripheral Channel Mask 1 Register

The `MLB_ACMR1` register allows control over which channel(s) generate interrupts on MLB_INT[1:0]. All of the bits in this register default to 0 (masked) so the HC must initially write to the `MLB_ACMR1` register to enable interrupts. Each bit of this register is read/write accessible.

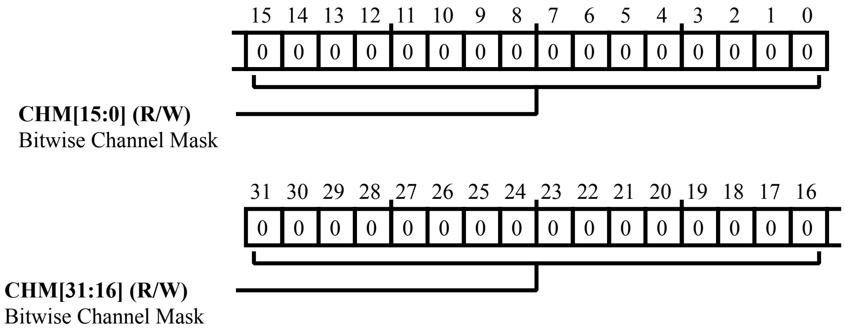

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[15:0] (R/W)
Bitwise Channel Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[31:16] (R/W)
Bitwise Channel Mask

Figure 28-8: MLB_ACMR1 Register Diagram

Table 28-21: MLB_ACMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W)	CHM	Bitwise Channel Mask. The `MLB_ACMR1.CHM` bit field masks or unmasks channels 63-32.	
		0	Mask interrupt for Channel 63:32 (bitwise; all channels shown)
		4294967295	Unmask interrupt for Channel 63:32 (bitwise; all channels shown)

Peripheral Channel Status 0 Register

The `MLB_ACSR0` register contains interrupt bits for each of the 64 physical channels. When a bit in this register is set, it indicates that the corresponding physical channel has an interrupt pending.

A peripheral interrupt is triggered when either DNEn or ERRn is set within the Bus Channel Descriptor. The HC is notified of the channel interrupt via MLB_INT[1:0]. When an interrupt occurs in ACCUSER (for channels 31 to 0) MLB_INT[0] is set. When an interrupt occurs in `MLB_ACSR1` (for channels 63 to 32) MLB_INT[1] is set.

Interrupts in the `MLB_ACSR0` and `MLB_ACSR1` registers can be optionally multiplexed onto a single interrupt signal, MLB_INT[0], if the `MLB_ACTL.SMX` bit = 1.

If the `MLB_ACTL.SCE` bit =0, hardware automatically clears the interrupt bit(s) after the HC reads the peripheral channel status registers. Alternatively, if the `MLB_ACTL.SCE` bit =1, software must write a 1 to the appropriate bit(s) of the peripheral channel status registers to clear the interrupt(s).

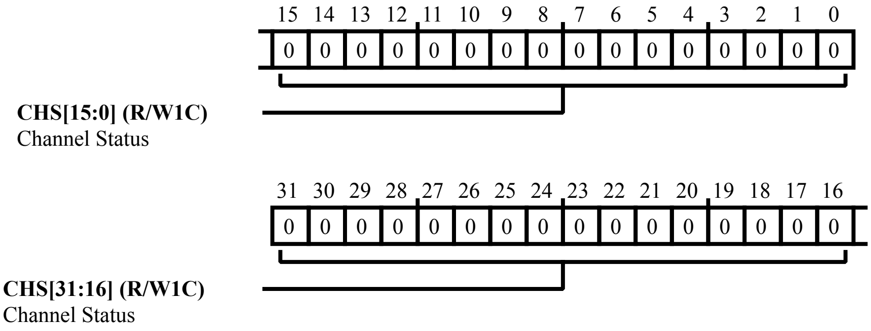

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHS[15:0] (R/W1C)
Channel Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHS[31:16] (R/W1C)
Channel Status

Figure 28-9: MLB_ACSR0 Register Diagram

Table 28-22: MLB_ACSR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	CHS	Channel Status. The `MLB_ACSR0.CHS` bit field indicates channel status for channels 31-0.	
		0	No interrupt on Channel 31:0 (bitwise; all channels shown)
		4294967295	Interrupt on Channel 31:0 (bitwise; all channels shown)

Peripheral Channel Status 1 Register

The MLB_ACSR1 register contains interrupt bits for each of the 64 physical channels. When a bit in this register is set, it indicates that the corresponding physical channel has an interrupt pending.

A peripheral interrupt is triggered when either DNEn or ERRn is set within the Bus Channel Descriptor. The HC is notified of the channel interrupt via MLB_INT[1:0]. When an interrupt occurs in ACCUSER (for channels 31 to 0) MLB_INT[0] is set. When an interrupt occurs in MLB_ACSR1 (for channels 63 to 32) MLB_INT[1] is set.

Interrupts in the MLB_ACSR1 and MLB_ACSR0 registers can be optionally multiplexed onto a single interrupt signal, MLB_INT[0], if the `MLB_ACTL.SMX` bit = 1.

If the `MLB_ACTL.SCE` bit =0, hardware automatically clears the interrupt bit(s) after the HC reads the peripheral channel status registers. Alternatively, if the `MLB_ACTL.SCE` bit =1, software must write a 1 to the appropriate bit(s) of peripheral channel status registers to clear the interrupt(s).

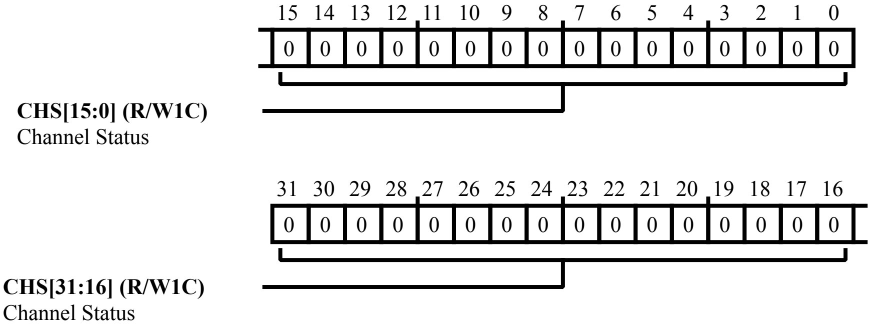

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHS[15:0] (R/W1C)
Channel Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHS[31:16] (R/W1C)
Channel Status

Figure 28-10: MLB_ACSR1 Register Diagram

Table 28-23: MLB_ACSR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	CHS	Channel Status. The `MLB_ACSR1.CHS` bit field indicates channel status for channels 63-32.	
		0	No interrupt on Channel 63:32 (bitwise; all channels shown)
		4294967295	Interrupt on Channel 63:32 (bitwise; all channels shown)

Bus Control Register

The `MLB_ACTL` register is written by the HC to configure the block for channel interrupts. This register contains bits to select the packet buffering mode and the DMA mode. This register also contains bits that are used to multiplex channel interrupts onto a single interrupt signal and to select the method of clearing channel interrupts.

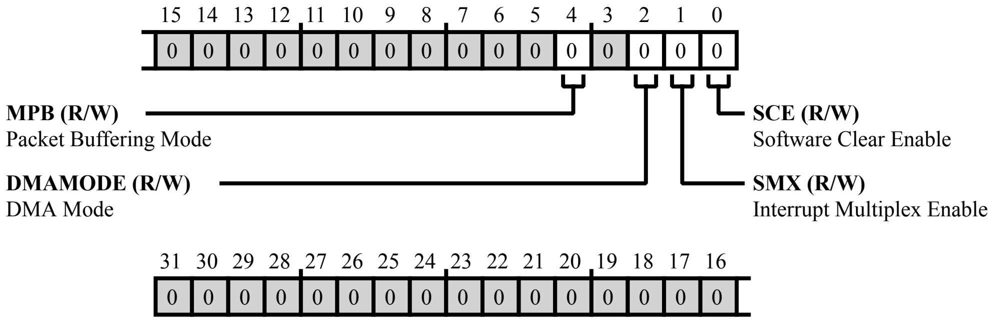

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MPB (R/W)
Packet Buffering Mode
DMAMODE (R/W)
DMA Mode
SCE (R/W)
Software Clear Enable
SMX (R/W)
Interrupt Multiplex Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-11: MLB_ACTL Register Diagram

Table 28-24: MLB_ACTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	MPB	Packet Buffering Mode. The `MLB_ACTL.MPB` bit selects whether the buffering mode is single-packet or multiple-packet.	
		0	single-packet mode
		1	multiple-packet mode
2 (R/W)	DMAMODE	DMA Mode. The `MLB_ACTL.DMAMODE` bit selects between DMA mode 1 and 0. DMA Mode 0 uses incrementing bursts of an unspecified length. This allows the block to perform single beat transfers as well as an incrementing (INCR) burst of unspecified length up to the maximum specified burst length (8 beats). DMA Mode 1 uses only INCR8, INCR4, and SINGLE beat bursts. The hburst[2:0] signal encodes the burst transfer used for a DMA operation.	
		0	DMA Mode 0
		1	DMA Mode 1

Table 28-24: MLB_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	SMX	Interrupt Multiplex Enable. The `MLB_ACTL.SMX` bit selects whether ACSR0 generates an interrupt on MLB_INT[0] and ACSR1 generates an interrupt on MLB_INT[1] or ACSR0 and ACSR1 generate an interrupt on MLB_INT[0] only.	
		0	ACSR0 generates an interrupt on MLB_INT[0]; ACSR1 generates an interrupt on MLB_INT[1]
		1	ACSR0 and ACSR1 generate an interrupts on MLB_INT[0] only
0 (R/W)	SCE	Software Clear Enable. The `MLB_ACTL.SCE` bit selects whether hardware clears the interrupt after a ACSRn register read or software clears the interrupt.	
		0	Hardware clears interrupt after a ACSRn register read
		1	Software clears interrupt

MediaLB Control 0 Register

The `MLB_CTL0` register contains bit that enable the module and provide module status. Note that the maximum speed for ML6-pin mode is 2048 x FS.

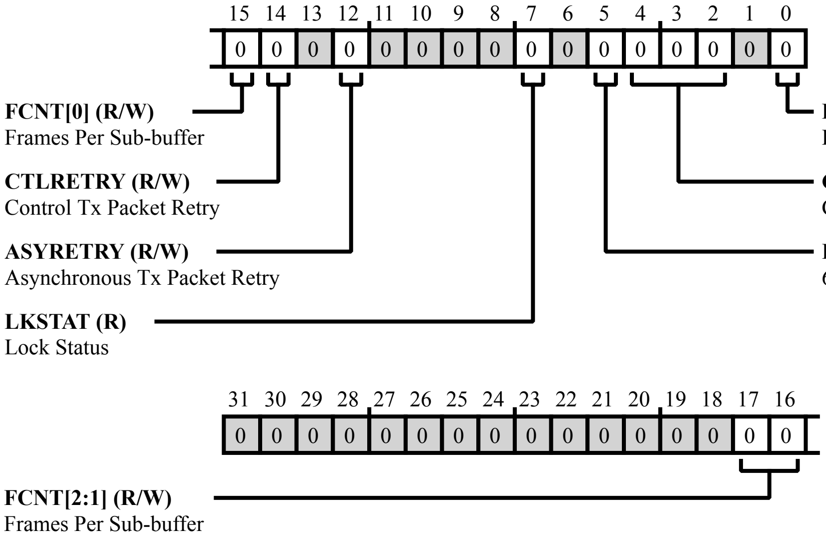

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FCNT[0] (R/W)
Frames Per Sub-buffer
CTLRETRY (R/W)
Control Tx Packet Retry
ASYRETRY (R/W)
Asynchronous Tx Packet Retry
LKSTAT (R)
Lock Status
EN (R/W)
Enable
CLK (R/W)
Clock Speed Select
PEN (R/W)
6-pin Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FCNT[2:1] (R/W)
Frames Per Sub-buffer

Figure 28-12: MLB_CTL0 Register Diagram

Table 28-25: MLB_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	FCNT	Frames Per Sub-buffer. The `MLB_CTL0.FCNT` bit field configures the frames per sub-buffer on synchronous channels.	
		0	1 Frame per sub-buffer (Operation is the same as Standard mode)
		1	2 frames per sub-buffer
		2	4 frames per sub-buffer
		3	8 frames per sub-buffer
		4	16 frames per sub-buffer
		5	32 frames per sub-buffer
		6	64 frames per sub-buffer
14 (R/W)	CTLRETRY	Control Tx Packet Retry. When the `MLB_CTL0.CTLRETRY` bit is set, a control packet that is flagged with a Break or Protocol error by the receiver is retransmitted. When cleared, a control packet that is flagged with a Break or Protocol error by the receiver is skipped.	

Table 28-25: MLB_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	ASYRETRY	Asynchronous Tx Packet Retry. When the `MLB_CTL0.ASYRETRY` bit is set, an asynchronous packet that is flagged with a Break or Protocol error by the receiver is retransmitted. When cleared, an asynchronous packet that is flagged with a Break or ProtocolError by the receiver is skipped.
7 (R/NW)	LKSTAT	Lock Status. When the `MLB_CTL0.LKSTAT` bit is set (=0), the MediaLB block is synchronized to the incoming MediaLB frame with the following conditions. • If `MLB_CTL1.LOCK` =0 (unlocked), `MLB_CTL1.LOCK` is set after a FRAMESYNC is detected at the same position for three consecutive frames. • If `MLB_CTL1.LOCK` =1 (locked), `MLB_CTL1.LOCK` is cleared after not receiving a FRAMESYNC at the expected time for two consecutive frames. In this case FRAMESYNC patterns occurring at locations other than the expected one are ignored.
5 (R/W)	PEN	6-pin Enable. The `MLB_CTL0.PEN` bit configures the MLB for 6-pin or 3-pin mode. 0 3-pin interface enabled 1 6-pin interface enabled
4:2 (R/W)	CLK	Clock Speed Select. The `MLB_CTL0.CLK` bit field sets the clock speed. 0 256 x Fs (MLB_CTL.PEN = 0) 1 512 x Fs (MLB_CTL.PEN = 0) 2 1024 x Fs (MLB_CTL.PEN = 0) 3 2048 x Fs (MLB_CTL.PEN = 1) 4 Reserved 5 Reserved 6 Reserved 7 Reserved
0 (R/W)	EN	Enable. When the `MLB_CTL0.EN` bit is set (=1), MediaLB clock, signal, and data are received and transmitted on the appropriate MediaLB pins.

Control 1 Register

The MLB_CTL1 register contains bits that provide lock status and control system commands.

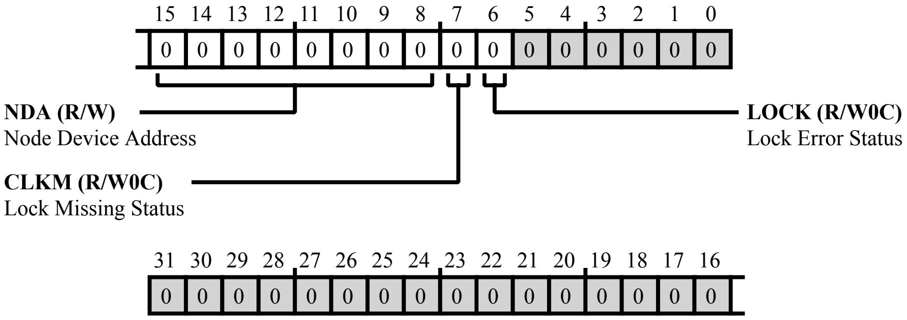

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NDA (R/W)
Node Device Address
CLKM (R/W0C)
Lock Missing Status
LOCK (R/W0C)
Lock Error Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-13: MLB_CTL1 Register Diagram

Table 28-26: MLB_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	NDA	Node Device Address. The `MLB_CTL1.NDA` bit field is used for system commands directed to individual MediaLB nodes.
7 (R/W0C)	CLKM	Lock Missing Status. The `MLB_CTL1.CLKM` bit is set when the MediaLB clock is not toggling at the pin. This bit is cleared by software.
6 (R/W0C)	LOCK	Lock Error Status. The `MLB_CTL1.LOCK` bit is set when the MediaLB is unlocked. This bit is cleared by software.

MLB Global Control Register

The `MLB_GCTL` register contains bits that manage the MLB clock.

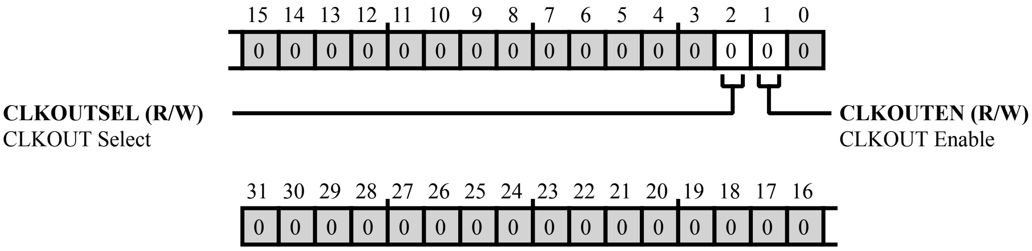

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKOUTSEL (R/W)
CLKOUT Select
CLKOUTEN (R/W)
CLKOUT Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-14: MLB_GCTL Register Diagram

Table 28-27: MLB_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CLKOUTSEL	CLKOUT Select. The `MLB_GCTL.CLKOUTSEL` bit selects either the MLB 3 pin clock or the 6 pin clock as MLBCLKOUT.	
		0	MLB 3 pin clock is selected for MLBCLKOUT
		1	MLB 6 pin clock is selected for MLBCLKOUT
1 (R/W)	CLKOUTEN	CLKOUT Enable.	

HBI Channel Busy 0 Register

The HC can determine which channel(s) are busy by reading the `MLB_HCBR0` register. An HBI channel is busy if:

- it is currently loaded into one of the two AGUs
- the channel is enabled, CE = 1 from the Channel Allocation Table
- the DMA is active

When an HBI channel is busy, hardware may write back its local copy of the channel descriptor at any time. System software should not write a CDT descriptor for a channel that is busy. Only two HBI channels can be busy at any given time. Each bit of this register is read-only.

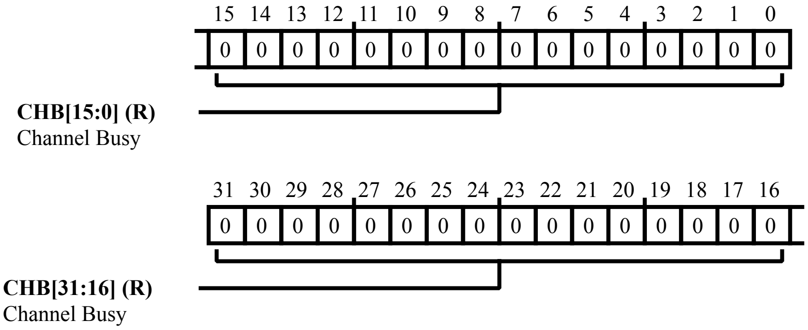

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHB[15:0] (R)
Channel Busy
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHB[31:16] (R)
Channel Busy

Figure 28-15: MLB_HCBR0 Register Diagram

Table 28-28: MLB_HCBR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CHB	Channel Busy. The `MLB_HCBR0.CHB` bit field contains the bitwise channel busy bit for channels 31:0. When a bit is cleared (=0) the channel is idle. When a bit is set (=1) the channel is busy.

HBI Channel Busy 1 Register

The HC can determine which channel(s) are busy by reading the `MLB_HCBR1` register. An HBI channel is busy if:

- it is currently loaded into one of the two AGUs
- the channel is enabled, CE = 1 from the Channel Allocation Table
- the DMA is active

When an HBI channel is busy, hardware may write back its local copy of the channel descriptor at any time. System software should not write a CDT descriptor for a channel that is busy. Only two HBI channels can be busy at any given time. Each bit of this register is read-only.

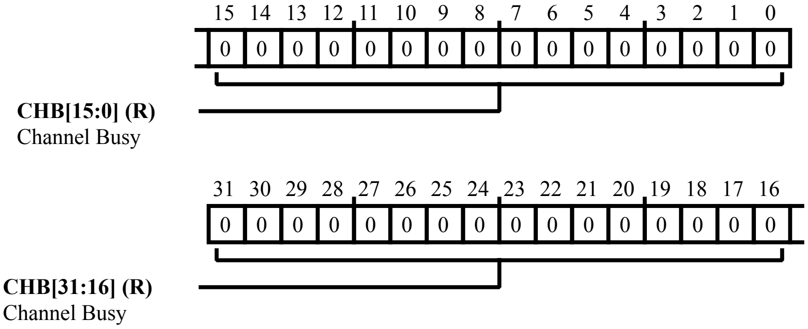

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHB[15:0] (R)
Channel Busy
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHB[31:16] (R)
Channel Busy

Figure 28-16: MLB_HCBR1 Register Diagram

Table 28-29: MLB_HCBR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CHB	Channel Busy. The `MLB_HCBR1.CHB` bit field contains the bitwise channel busy bit for channels 63:32. When a bit is cleared (=0) the channel is idle. When a bit is set (=1) the channel is busy.

HBI Channel Error 0 Register

The MLB_HCER0 register indicates which channels (channels 31:0) have encountered fatal errors.

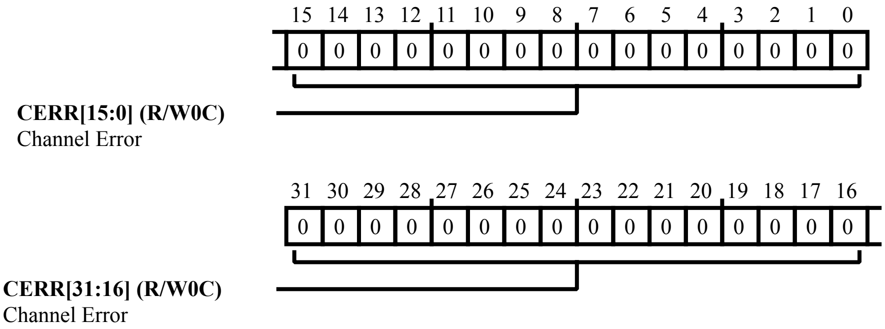

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CERR[15:0] (R/W0C)
Channel Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CERR[31:16] (R/W0C)
Channel Error

Figure 28-17: MLB_HCER0 Register Diagram

Table 28-30: MLB_HCER0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	CERR	Channel Error. The `MLB_HCER0.CERR` bit field reports bitwise errors for channels 31:0.

HBI Channel Error 1 Register

The `MLB_HCER1` register indicates which channel(s) have encountered fatal errors.

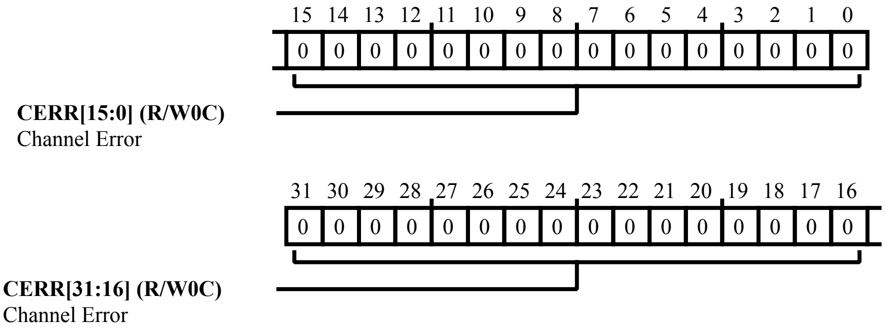

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CERR[15:0] (R/W0C)
Channel Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CERR[31:16] (R/W0C)
Channel Error

Figure 28-18: MLB_HCER1 Register Diagram

Table 28-31: MLB_HCER1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	CERR	Channel Error. The `MLB_HCER1.CERR` bite field reports bitwise errors for channels 63:32.

HBI Channel Mask 0 Register

The `MLB_HCMR0` register controls which channels (for channels 31:0) are able to generate an HBI interrupt.

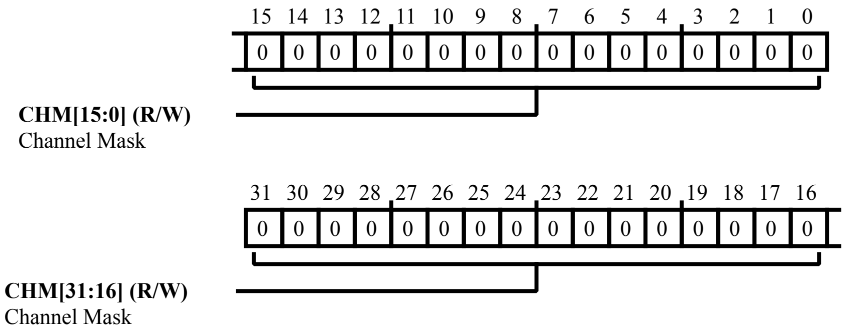

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[15:0] (R/W)
Channel Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[31:16] (R/W)
Channel Mask

Figure 28-19: MLB_HCMR0 Register Diagram

Table 28-32: MLB_HCMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Channel Mask. The `MLB_HCMR0.CHM` bit field contains the bitwise channel mask bit for channels 31:0. When a bit is cleared (=0) the channel is masked. When set (=1) the channel is unmasked.

HBI Channel Mask 1 Register

The MLB_HCMR1 register controls which channels (for channels 63:32) are able to generate an HBI interrupt.

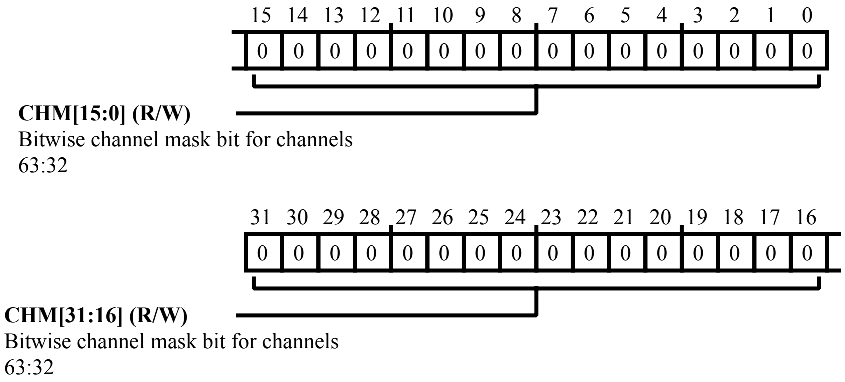

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[15:0] (R/W)
Bitwise channel mask bit for channels 63:32
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHM[31:16] (R/W)
Bitwise channel mask bit for channels 63:32

Figure 28-20: MLB_HCMR1 Register Diagram

Table 28-33: MLB_HCMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CHM	Bitwise channel mask bit for channels 63:32. The `MLB_HCMR1.CHM` bit field contains the bitwise channel mask bit for channels 63:32. When a bit is cleared (=0) the channel is masked. When set (=1) the channel is unmasked.

HBI Control Register

The `MLB_HCTL` register controls and monitors general operation of the HBI block through the Address Generation Units) by reading and writing the register through the I/O interface. Each bit of this register is read/write.

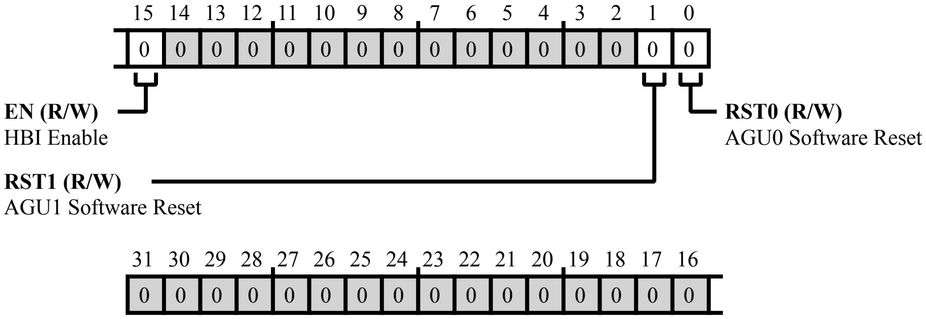

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
HBI Enable
RST0 (R/W)
AGU0 Software Reset
RST1 (R/W)
AGU1 Software Reset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-21: MLB_HCTL Register Diagram

Table 28-34: MLB_HCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	EN	HBI Enable. Setting the `MLB_HCTL.EN` bit enables the HBI.
1 (R/W)	RST1	AGU1 Software Reset. Setting the `MLB_HCTL.RST1` bit resets AGU1.
0 (R/W)	RST0	AGU0 Software Reset. Setting the `MLB_HCTL.RST0` bit resets AGU0.

Memory Interface Address Register

The MLB_MADR register contains bit fields that contain the Channel Table RAM (CTR) or Data Buffer RAM (DBR) addresses. It also contains bits that set target location and read-not-write parameters.

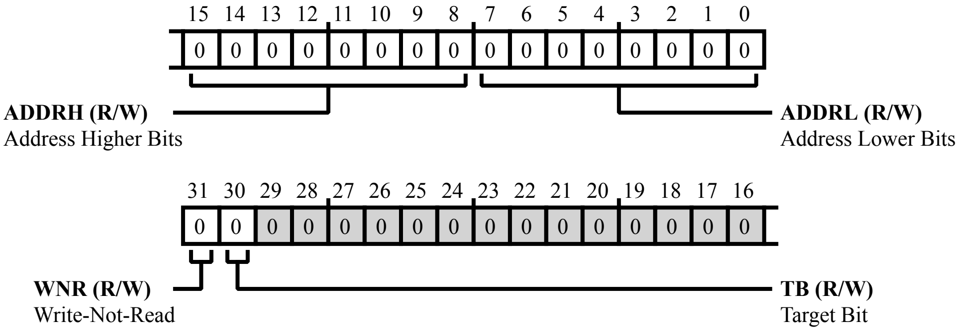

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRH (R/W)
Address Higher Bits
ADDRL (R/W)
Address Lower Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WNR (R/W)
Write-Not-Read
TB (R/W)
Target Bit

Figure 28-22: MLB_MADR Register Diagram

Table 28-35: MLB_MADR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	WNR	Write-Not-Read. The `MLB_MADR.WNR` bit selects Write-Not-Read.	
		0	Read
		1	Write
30 (R/W)	TB	Target Bit. The `MLB_MADR.TB` bit sets the target location.	
		0	Channel Table RAM (CTR)
		1	Data Buffer RAM (DBR)
15:8 (R/W)	ADDRH	Address Higher Bits. The `MLB_MADR.ADDRH` bit field contains the DBR address of the 8-bit entry (bits [13:8]).	
7:0 (R/W)	ADDRL	Address Lower Bits. The `MLB_MADR.ADDRL` bit field contains the CTR address of the 128-bit entry or the DBR address of the 8-bit entry (bits [7:0]).	

Memory Interface Control Register

The `MLB_MCTL` register contains a bit that indicates that the data transfer is complete.

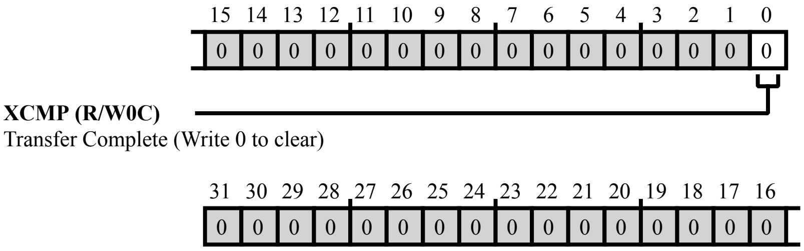

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
XCMP (R/W0C)
Transfer Complete (Write 0 to clear)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-23: MLB_MCTL Register Diagram

Table 28-36: MLB_MCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W0C)	XCMP	Transfer Complete (Write 0 to clear). The `MLB_MCTL.XCMP` bit indicates that the data transfer is complete.

Memory Interface Control Data 0 Register

The `MLB_MDAT0` register contains Channel Table RAM (CTR) data (bits [31:0] of 128-bit entry) or Data Buffer RAM (DBR) data (bits [7:0] of 8-bit entry).

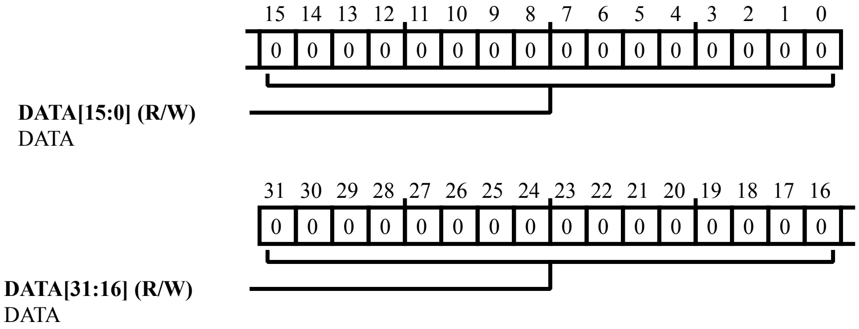

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
DATA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
DATA

Figure 28-24: MLB_MDAT0 Register Diagram

Table 28-37: MLB_MDAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The `MLB_MDAT0.DATA` bit field contains CTR data or DBR data.

Memory Interface Control Data 1 Register

The `MLB_MDAT1` register contains Channel Table RAM (CTR) data (bits [63:32] of 128-bit entry).

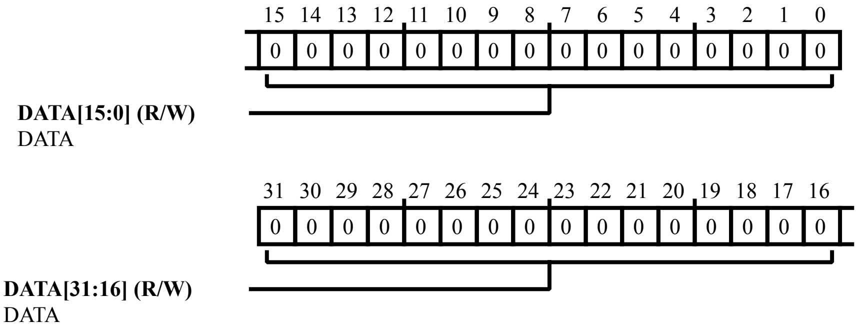

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
DATA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
DATA

Figure 28-25: MLB_MDAT1 Register Diagram

Table 28-38: MLB_MDAT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The `MLB_MDAT1.DATA` bit field contains CTR data.

Memory Interface Control Data 2 Register

The `MLB_MDAT2` register contains Channel Table RAM (CTR) data (bits [95:64] of 128-bit entry).

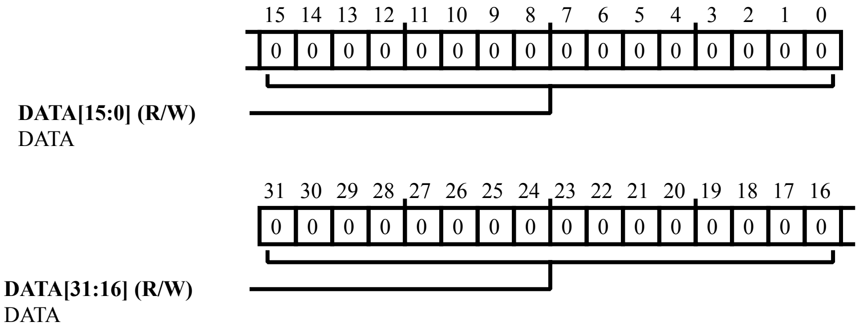

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
DATA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
DATA

Figure 28-26: MLB_MDAT2 Register Diagram

Table 28-39: MLB_MDAT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The `MLB_MDAT2.DATA` bit field contains CTR data.

Memory Interface Control Data 3 Register

The `MLB_MDAT3` register contains Channel Table RAM (CTR) data (bits [127:96] of 128-bit entry).

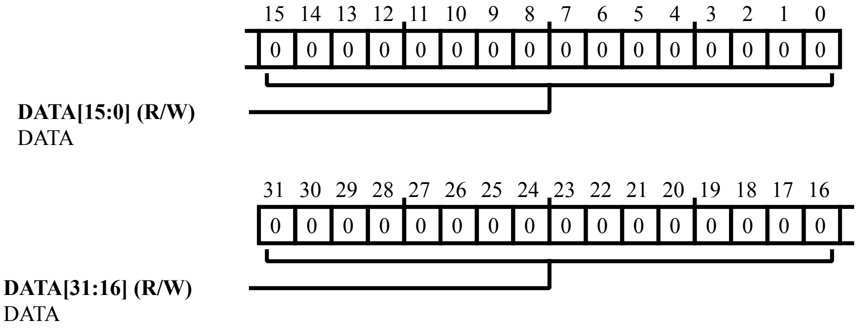

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[15:0] (R/W)
DATA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATA[31:16] (R/W)
DATA

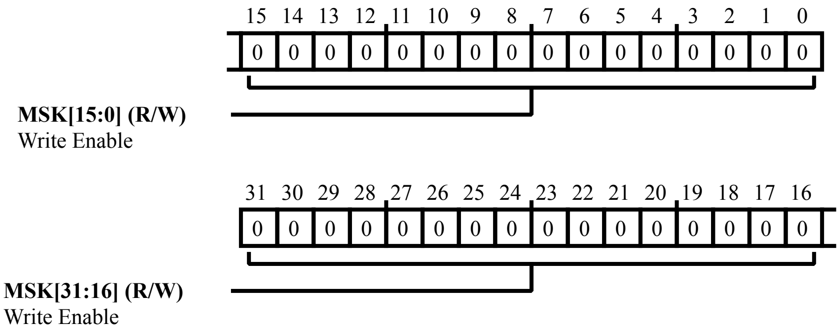
Figure 28-27: MLB_MDAT3 Register Diagram

Table 28-40: MLB_MDAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	DATA. The `MLB_MDAT3.DATA` bit field contains CTR data.

Memory Interface Control Data Write Enable 0 Register

The `MLB_MDWE0` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [31:0]. When cleared (=0) the bit is disabled, when set (=1) the bit is enabled.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[15:0] (R/W)
Write Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[31:16] (R/W)
Write Enable

Figure 28-28: MLB_MDWE0 Register Diagram

Table 28-41: MLB_MDWE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Write Enable. The `MLB_MDWE0.MSK` bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 1 Register

The `MLB_MDWE1` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [63:32]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

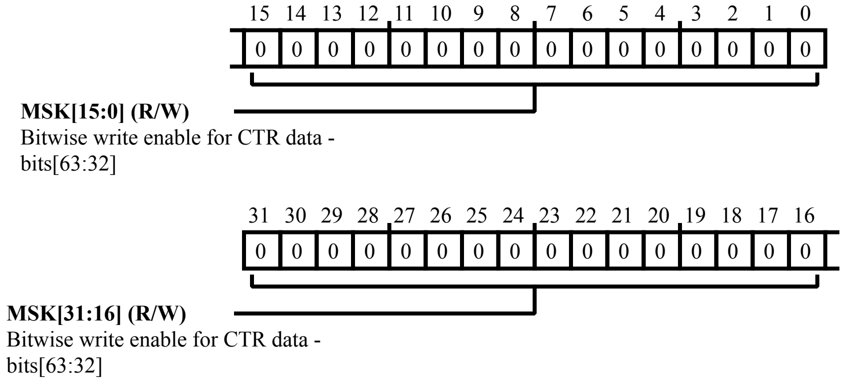

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[15:0] (R/W)
Bitwise write enable for CTR data - bits[63:32]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[31:16] (R/W)
Bitwise write enable for CTR data - bits[63:32]

Figure 28-29: MLB_MDWE1 Register Diagram

Table 28-42: MLB_MDWE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[63:32]. The `MLB_MDWE1.MSK` bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 2 Register

The `MLB_MDWE2` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [95:64]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

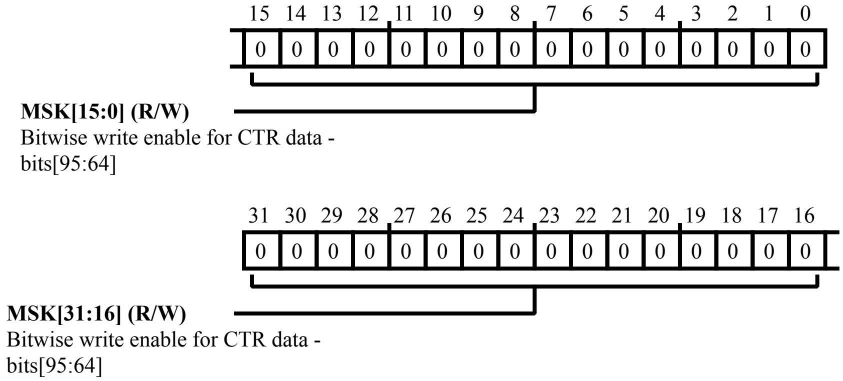

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[15:0] (R/W)
Bitwise write enable for CTR data - bits[95:64]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[31:16] (R/W)
Bitwise write enable for CTR data - bits[95:64]

Figure 28-30: MLB_MDWE2 Register Diagram

Table 28-43: MLB_MDWE2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[95:64]. The `MLB_MDWE2.MSK` bit field contains the bitwise write enable for CTR data.

Memory Interface Control Data Write Enable 3 Register

The `MLB_MDWE3` register contains the bitwise write enable for Channel Table RAM (CTR) data bits [127:96]. When cleared (=0), the bit is disabled. When set (=1), the bit is enabled.

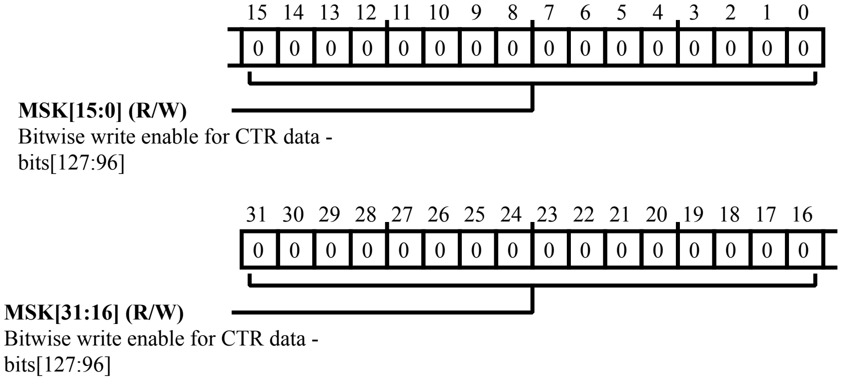

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[15:0] (R/W)
Bitwise write enable for CTR data - bits[127:96]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSK[31:16] (R/W)
Bitwise write enable for CTR data - bits[127:96]

Figure 28-31: MLB_MDWE3 Register Diagram

Table 28-44: MLB_MDWE3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSK	Bitwise write enable for CTR data - bits[127:96]. The `MLB_MDWE3.MSK` bit field contains the bitwise write enable for CTR data.

Interrupt Enable Register

The MLB_MIEN register is used to enable various interrupt conditions.

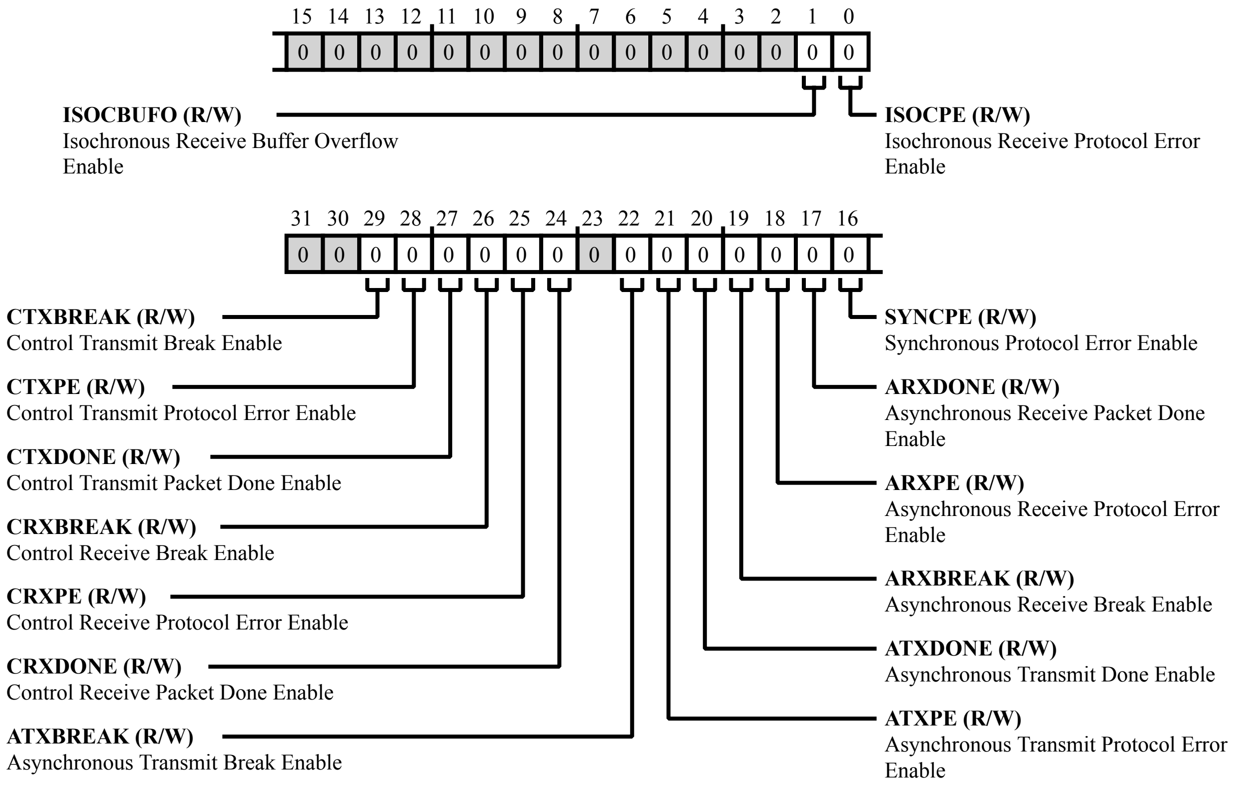

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ISOCBUFO (R/W)
Isochronous Receive Buffer Overflow Enable
ISOCPE (R/W)
Isochronous Receive Protocol Error Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CTXBREAK (R/W)
Control Transmit Break Enable
CTXPE (R/W)
Control Transmit Protocol Error Enable
CTXDONE (R/W)
Control Transmit Packet Done Enable
CRXBREAK (R/W)
Control Receive Break Enable
CRXPE (R/W)
Control Receive Protocol Error Enable
CRXDONE (R/W)
Control Receive Packet Done Enable
ATXBREAK (R/W)
Asynchronous Transmit Break Enable
SYNCPE (R/W)
Synchronous Protocol Error Enable
ARXDONE (R/W)
Asynchronous Receive Packet Done Enable
ARXPE (R/W)
Asynchronous Receive Protocol Error Enable
ARXBREAK (R/W)
Asynchronous Receive Break Enable
ATXDONE (R/W)
Asynchronous Transmit Done Enable
ATXPE (R/W)
Asynchronous Transmit Protocol Error Enable

Figure 28-32: MLB_MIEN Register Diagram

Table 28-45: MLB_MIEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	CTXBREAK	Control Transmit Break Enable. When the MLB_MIEN.CTXBREAK bit is set, a ReceiverBreak response received from the receiver on a control Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
28 (R/W)	CTXPE	Control Transmit Protocol Error Enable. When the MLB_MIEN.CTXPE bit is set, a ProtocolError generated by the receiver on a control Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.
27 (R/W)	CTXDONE	Control Transmit Packet Done Enable. When the MLB_MIEN.CTXDONE bit is set, a packet transmitted with no errors on a control Tx channel causes the appropriate channel bit in the MLB_MS0 or MLB_MS1 registers to be set.

Table 28-45: MLB_MIEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	CRXBREAK	Control Receive Break Enable. When the `MLB_MIEN.CRXBREAK` bit is set, a ControlBreak command received from the transmitter on a control Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
25 (R/W)	CRXPE	Control Receive Protocol Error Enable. When the `MLB_MIEN.CRXPE` bit is set, a ProtocolError detected on a control Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
24 (R/W)	CRXDONE	Control Receive Packet Done Enable. When the `MLB_MIEN.CRXDONE` bit is set, a packet received with no errors on a control Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
22 (R/W)	ATXBREAK	Asynchronous Transmit Break Enable. When the `MLB_MIEN.ATXBREAK` bit is set, a ReceiverBreak response received from the receiver on an asynchronous Tx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
21 (R/W)	ATXPE	Asynchronous Transmit Protocol Error Enable. When the `MLB_MIEN.ATXPE` bit is set, a ProtocolError generated by the receiver on an asynchronous Tx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
20 (R/W)	ATXDONE	Asynchronous Transmit Done Enable. When the `MLB_MIEN.ATXDONE` bit is set, a packet transmitted with no errors on an asynchronous Tx channel channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
19 (R/W)	ARXBREAK	Asynchronous Receive Break Enable. When the `MLB_MIEN.ARXBREAK` bit is set, a AsyncBreak command received from the transmitter on an asynchronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
18 (R/W)	ARXPE	Asynchronous Receive Protocol Error Enable. When the `MLB_MIEN.ARXPE` bit is set, a protocol error detected on an asynchronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.
17 (R/W)	ARXDONE	Asynchronous Receive Packet Done Enable. When the `MLB_MIEN.ARXDONE` bit is set, a packet received with no errors on an asynchronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.

Table 28-45: MLB_MIEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SYNCPE	Synchronous Protocol Error Enable. When the `MLB_MIEN.SYNCPE` bit is set, a protocol error detected on a synchronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set. This occurs only when isochronous flow control is disabled.
1 (R/W)	ISOCBUFO	Isochronous Receive Buffer Overflow Enable. When the `MLB_MIEN.ISOCBUFO` bit is set, a buffer overflow on an isochronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set. This occurs only when isochronous flow control is disabled.
0 (R/W)	ISOCPE	Isochronous Receive Protocol Error Enable. When the `MLB_MIEN.ISOCPE` bit is set, a protocol error detected on an isochronous Rx channel causes the appropriate channel bit in the `MLB_MS0` or `MLB_MS1` registers to be set.

Channel Status 0 Register

The `MLB_MS0` register indicates the channel status for MediaLB channels 31 to 0. Channel status bits are set by hardware and cleared by software. Status is only set if the appropriate bits in the `MLB_MIEN` register are set.

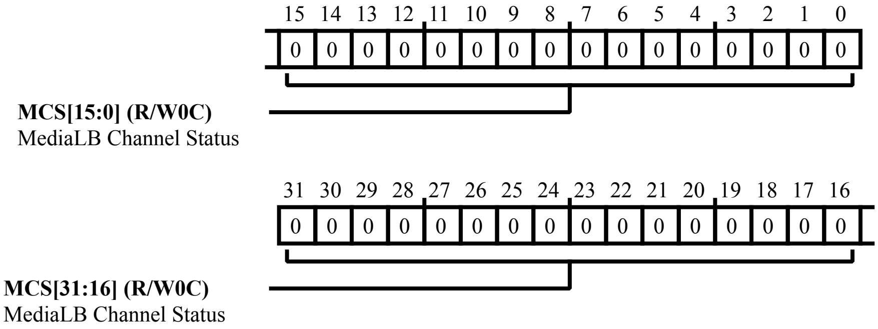

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MCS[15:0] (R/W0C)
MediaLB Channel Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MCS[31:16] (R/W0C)
MediaLB Channel Status

Figure 28-33: MLB_MS0 Register Diagram

Table 28-46: MLB_MS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	MCS	MediaLB Channel Status. The `MLB_MS0.MCS` bit field indicates the MediaLB channel status for channels 31 to 0.

Channel Status 1 Register

The `MLB_MS1` register indicates the channel status for MediaLB channels 32 to 63. Channel status bits are set by hardware and cleared by software. Status is only set if the appropriate bits in the `MLB_MIEN` register are set.

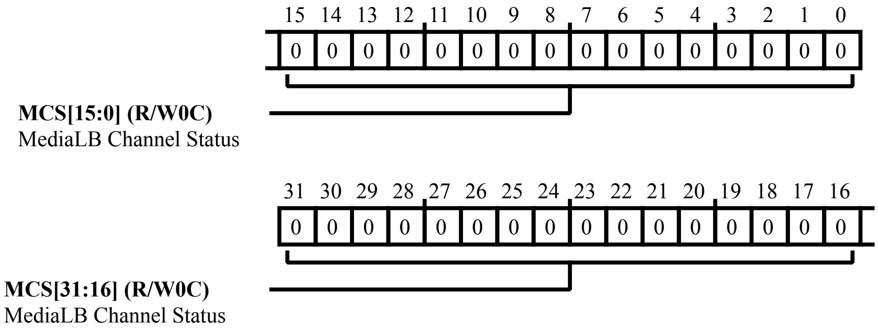

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MCS[15:0] (R/W0C)
MediaLB Channel Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MCS[31:16] (R/W0C)
MediaLB Channel Status

Figure 28-34: MLB_MS1 Register Diagram

Table 28-47: MLB_MS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W0C)	MCS	MediaLB Channel Status. The `MLB_MS1.MCS` bit field indicates the MediaLB channel status for channels 63 to 32.

System Data Register

The `MLB_MSD` register allows system software to receive control information from the MLB controller. The `MLB_MSD` register is updated once per frame by the hardware during the MLB system channel.

The `MLB_MSD` register is loaded with the data from the MLBDAT_IN signal during the system channel quadlet. System software must read this register before the start of the next MLB frame to prevent the current data from being lost.

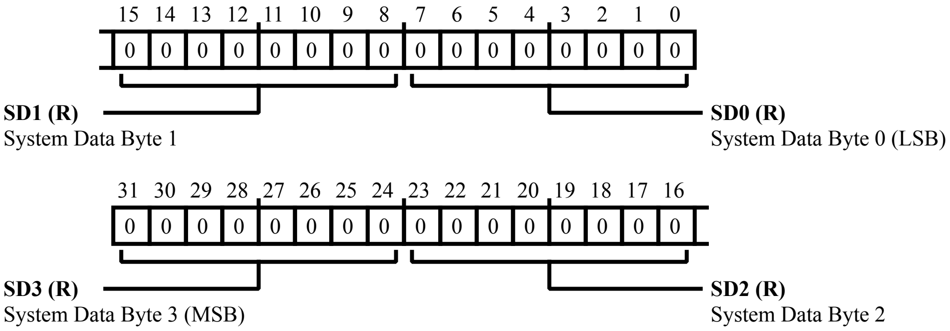

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SD1 (R)
System Data Byte 1
SD0 (R)
System Data Byte 0 (LSB)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SD3 (R)
System Data Byte 3 (MSB)
SD2 (R)
System Data Byte 2

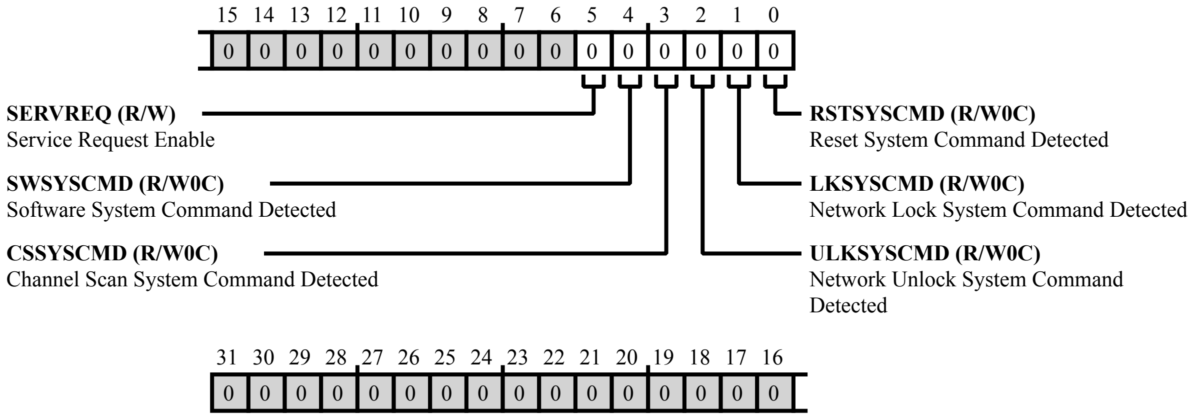
Figure 28-35: MLB_MSD Register Diagram

Table 28-48: MLB_MSD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	SD3	System Data Byte 3 (MSB). The `MLB_MSD.SD3` bits are updated with MediaLB Data [31:24] when a MediaLB software system command is received in the system quadlet. If the `MLB_MSS.SWSYSCMD` bit is already set, then SD3 is not updated.
23:16 (R/NW)	SD2	System Data Byte 2. The `MLB_MSD.SD2` bits are updated with MediaLB Data [23:16] when a MediaLB software system command is received in the system quadlet. If the `MLB_MSS.SWSYSCMD` bit is already set, then SD2 is not updated.
15:8 (R/NW)	SD1	System Data Byte 1. The `MLB_MSD.SD1` bits are updated with MediaLB Data [15:8] when a MediaLB software system command is received in the system quadlet. If the `MLB_MSS.SWSYSCMD` bit is already set, then SD1 is not updated.
7:0 (R/NW)	SD0	System Data Byte 0 (LSB). The `MLB_MSD.SD0` bits are updated with MediaLB Data [7:0] when a MediaLB software system command is received in the system quadlet. If the `MLB_MSS.SWSYSCMD` bit is already set, then SD0 is not updated.

System Status Register

The MLB_MSS register allows system software to monitor and control the status of the MLB network. The register is updated once per frame by hardware during the MLB system channel. The bits of this register are not valid until the processor is locked to the MLB interface (except for the bits associated with MLB lock and unlock, SDMU and SDML). System software must service events before the start of the next MLB frame to prevent the current frame status from being lost.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SERVREQ (R/W)
Service Request Enable
SWSYSCMD (R/W0C)
Software System Command Detected
CSSYSCMD (R/W0C)
Channel Scan System Command Detected
RSTSYSCMD (R/W0C)
Reset System Command Detected
LKSYSCMD (R/W0C)
Network Lock System Command Detected
ULKSYSCMD (R/W0C)
Network Unlock System Command Detected
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-36: MLB_MSS Register Diagram

Table 28-49: MLB_MSS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SERVREQ	Service Request Enable. When the `MLB_MSS.SERVREQ` bit set, the MediaLB block responds with a device present, request service system response if a matching channel scan system command is detected. When cleared, the MediaLB block responds with a device present system response.
4 (R/W0C)	SWSYSCMD	Software System Command Detected. The `MLB_MSS.SWSYSCMD` bit indicates that a software system command was detected (in the system quadlet). The `MLB_MSS.SWSYSCMD` bit is set by hardware, cleared by software. Data is stored in the MLB_MSD register for this command.
3 (R/W0C)	CSSYSCMD	Channel Scan System Command Detected. The `MLB_MSS.CSSYSCMD` bit indicates that a channel scan system command was detected (in the system quadlet). The `MLB_MSS.CSSYSCMD` bit is set by hardware, cleared by software. If the node address specified in the data quadlet matches the value in the `MLB_CTL1.NDA` bit field, the device responds either device present or device present, request service system response in the next system quadlet.

Table 28-49: MLB_MSS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W0C)	ULKSYSCMD	Network Unlock System Command Detected. The `MLB_MSS.ULKSYSCMD` bit indicates a network unlock system command was detected (in the system quadlet). The `MLB_MSS.ULKSYSCMD` bit is by hardware, cleared by software.
1 (R/W0C)	LKSYSCMD	Network Lock System Command Detected. The `MLB_MSS.LKSYSCMD` bit indicates a network lock system command was detected (in the system quadlet). The `MLB_MSS.LKSYSCMD` bit is set by hardware and cleared by software.
0 (R/W0C)	RSTSYSCMD	Reset System Command Detected. The `MLB_MSS.RSTSYSCMD` bit indicates a reset system command was detected (in the system quadlet). The `MLB_MSS.RSTSYSCMD` bit is set by hardware, cleared by software.

MediaLB 6-pin Control 0 Register

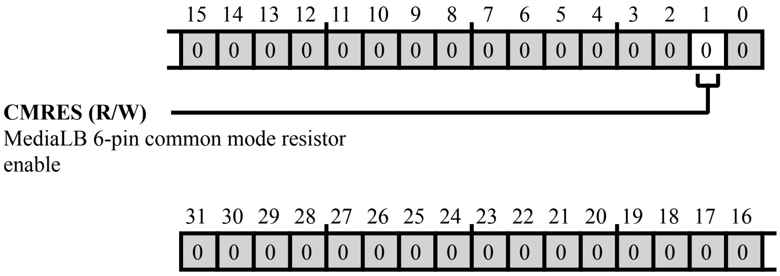

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CMRES (R/W)
MediaLB 6-pin common mode resistor enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 28-37: MLB_PCTL0 Register Diagram

Table 28-50: MLB_PCTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	CMRES	MediaLB 6-pin common mode resistor enable. This signal is no longer used.

29 PCI Express

Peripheral Component Interconnect Express (PCIe) is a high performance, general purpose I/O interconnect defined for a wide variety of computing and communication platforms.

PCIe retains some desirable attributes of the PCI interface, such as its usage model, load-store architecture, and software interfaces, whereas its parallel bus implementation is replaced by a highly scalable, fully serial interface. PCI Express takes advantage of recent advances in point-to-point interconnects, switch-based technology, and packetized protocol to deliver new levels of performance and features.

The PCIe Dual Mode core implements the three PCI Express protocol layers; the transaction layer, data link layer, and the MAC portion of the physical layer. It also implements the application-dependent functionality of the PCI Express transaction layer for packet transmission, which is located between the application logic and the PCI Express protocol layers. As shown in the ***PCIe Layers*** figure, a complete PCI Express Port solution includes the core, an analog PHY macro, and application logic to source and sink data. The physical layer is split across the PIPE such that the MAC functionality (LTSSM) is in the core and the PHY functionality is implemented in the PIPE-compliant PHY. The PIPE interface is internal to the core, and the external interface for the PHY consists of one transmit and one receive differential pair. Otherwise, the PHY module resides outside of the core, interfacing through the standard PIPE.

The PCIe interface is in compliance with PCI-SIG standards. If you require detailed information on PCIe protocol, refer the corresponding PCI-SIG specifications. The PCIe controller follows the following standards.

- PCI Express Base 3.0 Specification, revision 1.0
- PIPE Specification for PCI Express, Version 4.0, Intel Corporation

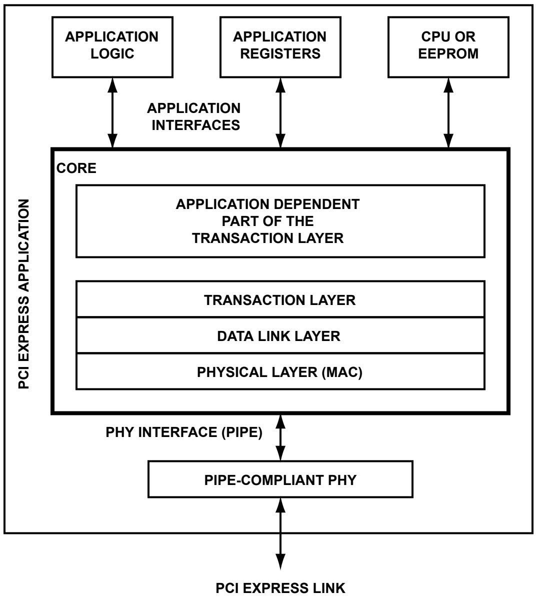

APPLICATION LOGIC
APPLICATION REGISTERS
CPU OR EEPROM
APPLICATION INTERFACES
CORE
APPLICATION DEPENDENT PART OF THE TRANSACTION LAYER
TRANSACTION LAYER
DATA LINK LAYER
PHYSICAL LAYER (MAC)
PCI EXPRESS APPLICATION
PHY INTERFACE (PIPE)
PIPE-COMPLIANT PHY
PCI EXPRESS LINK

Figure 29-1: PCIe Layers

PCIe Features

The PCI Express core is compliant with the PCI Express Gen 2 standard. The following features are supported:

- Dual operation mode: Root Complex (RC) and End Point (EP)
- Supports a single bidirectional x1 link interface
- Supports 2.5 Gbps or 5.0 Gbps per lane per direction
- 3.3 V I/O power supply (The PHY can also support a 2.5 V I/O)
- Maximum outbound payload size of 256 bytes
- Maximum inbound payload size of 256 bytes
- Maximum remote read request size of 256 bytes
- Polarity inversion on receive
- Single virtual channel (VC)
- Single traffic class (TC)
- Single function in End Point (EP) mode
- ECRC generation and checking
- PCI device power management
- PCI Express active state power management (ASPM) state L0s and L1

- PCI Express link power management
- PCI Express advanced error reporting
- PCI Express messages for both transmit and receive
- Filtering for posted, non-posted, and completion traffic
- Configurable BAR filtering, I/O filtering, configuration filtering, and completion lookup/timeout
- Access to configuration space registers and external application memory-mapped registers through local bus controller
- Legacy interrupts reception (RC) and generation (EP)
- MSI reception (RC) and generation (EP)
- Tx-to-Rx PIPE loopback
- IEEE 1149.6 (JTAG) boundary scan

Functional Description

The following sections provide information on the function of the PCIe.

ADSP-SC58x PCIE Register List

The following table lists the registers used by the PCIe interface.

Table 29-1: ADSP-SC58x PCIE Register List

Name	Description
PCIE_ACK_ASPMCTL_[n]	Acknowledge Frequency and L0-L1 ASPM Control Register
PCIE_ACK_TMR_[n]	Acknowledge Latency Timer and Replay Timer Register
PCIE_ADVERR_CAPBCTL_[n]	Advanced Error Capabilities and Control Register
PCIE_ADVERR_EXTCAPB_[n]	Advanced Error Reporting Extended Capability Header Register
PCIE_APP_CORERR_MSG	Correctable Error Message Requester ID Register
PCIE_APP_CTL	Application Control Register
PCIE_APP_DIAGCTL	Diagnostic Control Register
PCIE_APP_DIAGSTAT	Diagnostic Status Register
PCIE_APP_FERR_MSG	Fatal Error Message Requester ID Register
PCIE_APP_INTSTAT	Application Interrupts Status Register
PCIE_APP_NFERR_MSG	Non Fatal Error Message Requester ID Register
PCIE_APP_PMACK_MSG	Power Management Acknowledge Message Requester ID Register
PCIE_APP_PMEVT_MSG	Power Management Event Message Requester ID Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_APP_PMTOFF_MSG	Power Management Turn Off Message Requester ID Register
PCIE_APP_STAT	Application Status Register
PCIE_APP_UNLK_MSG	Unlock Message Requester ID Register
PCIE_APP_VNDR_MSG	Vendor Message Requester ID Register
PCIE_APP_VNDR_MSG_HDR0	Vendor Message Header Bytes 8 to 11 Register
PCIE_APP_VNDR_MSG_HDR1	Vendor Message Header Bytes 12 to 15 Register
PCIE_AUX_CLKFREQ_[n]	Auxiliary Clock Frequency Control Register
PCIE_BERR_RESP_[n]	Peripheral and SCB Bridge Slave Error Response Register
PCIE_BUSMGR_WDOG_[n]	Bus Order Manager Watchdog Off Register
PCIE_BUS_MULOB_DECOMP_[n]	Bus Multiple Outbound Decomposition SubReq Control Register
PCIE_CACHE_CTL1_[n]	ACE Cache Coherency Control Register 1
PCIE_CACHE_CTL3_[n]	ACE Cache Coherency Control Register 3
PCIE_CAPB_NPTR_[n]	PCIe Capabilities, ID, Next Pointer Register
PCIE_CORERR_MSK_[n]	Correctable Error Mask Register
PCIE_CORERR_STAT_[n]	Correctable Error Status Register
PCIE_DEV_CAPB2_[n]	Device Capabilities 2 Register
PCIE_DEV_CTLSTAT2_[n]	Device Control 2 and Status 2 Register
PCIE_DEV_CTLSTAT_[n]	Device Control and Status Register
PCIE_DMARD_ARBWGT_LO_[n]	DMA Read Arbitration Weight Low Off Register
PCIE_DMARD_CH01_IMWR_[n]	DMA Read Channel 1 and 0 IMWr Data Register
PCIE_DMARD_CTL1_[n]	DMA Control 1 Read Channel Register
PCIE_DMARD_DAR_HI_[n]	DMA Destination Address High Read Channel Register
PCIE_DMARD_DAR_LO_[n]	DMA Destination Address Low Read Channel Register
PCIE_DMARD_DONE_IMWR_HI_[n]	DMA Read Done IMWr Address High Register
PCIE_DMARD_DONE_IMWR_LO_[n]	DMA Read Done IMWr Address Low Register
PCIE_DMARD_DRBL_[n]	DMA Read Doorbell Register
PCIE_DMARD_EN_[n]	DMA Read Engine Enable Register
PCIE_DMARD_ERRSTAT_HI_[n]	DMA Read Error Status High Register
PCIE_DMARD_ERRSTAT_LO_[n]	DMA Read Error Status Low Register
PCIE_DMARD_ICLR_[n]	DMA Read Interrupt Clear Register
PCIE_DMARD_IMSK_[n]	DMA Read Interrupt Mask Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_DMARD_IMWRABRT_HI_[n]	DMA Read Abort IMWr Address High Register
PCIE_DMARD_IMWRABRT_LO_[n]	DMA Read Abort IMWr Address Low Register
PCIE_DMARD_ISTAT_[n]	DMA Read Interrupt Status Register
PCIE_DMARD_LLP_HI_[n]	DMA LLP High Read Channel Register
PCIE_DMARD_LLP_LO_[n]	DMA LLP Low Read Channel Register
PCIE_DMARD_LLSTERR_EN_[n]	DMA Read Linked List Error Enable Register
PCIE_DMARD_SAR_HI_[n]	DMA SAR High Read Channel Register
PCIE_DMARD_SAR_LO_[n]	DMA SAR Low Read Channel Register
PCIE_DMARD_XFRSZ_[n]	DMA Transfer Size Read Channel Register
PCIE_DMAWR_ABRT_IMWR_HI[n]	DMA Write Abort IMWr Address High Register
PCIE_DMAWR_ABRT_IMWR_LO[n]	DMA Write Abort IMWr Address Low Register
PCIE_DMAWR_ARBWGT_LO_[n]	DMA Write Channel Arbitration Weight Low Off Register
PCIE_DMAWR_CH01_IMWR_[n]	DMA Write Channel 1 and 0 IMWr Data Register
PCIE_DMAWR_CTL1_[n]	DMA Control 1 Write Channel Register
PCIE_DMAWR_DAR_HI_[n]	DMA Dar High Write Channel Register
PCIE_DMAWR_DAR_LO_[n]	DMA DAR Low Write Channel Register
PCIE_DMAWR_DONE_IMWR_HI_[n]	DMA Write Done IMWr Interrupt Address High Register
PCIE_DMAWR_DONE_IMWR_LO_[n]	DMA Write Done IMWr Address Low Register
PCIE_DMAWR_DRBL_[n]	DMA Write Doorbell Register
PCIE_DMAWR_EN_[n]	DMA Write Engine Enable Register
PCIE_DMAWR_ERRSTAT_[n]	DMA Write Error Status Register
PCIE_DMAWR_ICLR_[n]	DMA Write Interrupt Clear Register
PCIE_DMAWR_IMSK_[n]	DMA Write Interrupt Mask Register
PCIE_DMAWR_ISTAT_[n]	DMA Write Interrupt Status Register
PCIE_DMAWR_LLP_HI_[n]	DMA LLP High Write Channel Register
PCIE_DMAWR_LLP_LO_[n]	DMA LLP Low Write Channel Register
PCIE_DMAWR_LLSTERR_EN_[n]	DMA Write Linked List Error Enable Register
PCIE_DMAWR_PREQ_TMR_[n]	DMA Write Posted Request Deadlock Timer Register
PCIE_DMAWR_SAR_HI_[n]	DMA SAR High Write Channel Register
PCIE_DMAWR_SAR_LO_[n]	DMA SAR Low Write Channel Register
PCIE_DMAWR_XFRSZ_0[n]	DMA Transfer Size Write Channel Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_DMA_CTL_[n]	DMA Number of Channels Register
PCIE_DMA_VWPRT_SEL_[n]	DMA Channel Context Index Register
PCIE_EP_BAR0_MASK[n]	Endpoint Base Address Mask Register 0
PCIE_EP_BAR0_[n]	Endpoint Base Address Register 0
PCIE_EP_BAR1_MASK[n]	Endpoint Base Address Mask Register 1
PCIE_EP_BAR1_[n]	Endpoint Base Address Register 1
PCIE_EP_BAR2_MASK[n]	Endpoint Base Address Mask Register 2
PCIE_EP_BAR2_[n]	Endpoint Base Address Register 2
PCIE_EP_BAR3_MASK[n]	Endpoint Base Address Mask Register 3
PCIE_EP_BAR3_[n]	Endpoint Base Address Register 3
PCIE_EP_BAR4_MASK[n]	Endpoint Base Address Mask Register 4
PCIE_EP_BAR4_[n]	Endpoint Base Address Register 4
PCIE_EP_BAR5_MASK[n]	Endpoint Base Address Mask Register 5
PCIE_EP_BAR5_[n]	End Point Base Address Register 5
PCIE_EP_CAPBPTR_[n]	Capability Pointer Register
PCIE_EP_CCRID_[n]	Class Code and Revision ID Register
PCIE_EP_CFG_[n]	End Point Configuration Register
PCIE_EP_CRDBPTR_[n]	CardBus CIS Pointer Register
PCIE_EP_DEVCAPB_[n]	Device Capabilities Register
PCIE_EP_ID_[n]	Device ID and Vendor ID Register
PCIE_EP_PINLN_INT_[n]	Interrupt Line and Pin Register
PCIE_EP_ROMCFG_[n]	End Point Expansion ROM Base Address Register
PCIE_EP_SSVID_[n]	Subsystem ID and Subsystem Vendor ID Register
PCIE_EP_STATCMD_[n]	Command and Status Register
PCIE_ERRSRC_ID_[n]	Error Source Identification Register
PCIE_FILTMSK2_[n]	Filter Mask 2 Register
PCIE_GEN2_CTL_[n]	Link Width and Speed Change Control Register
PCIE_HDRLOG0_[n]	Header Log Register 0
PCIE_HDRLOG1_[n]	Header Log Register 1
PCIE_HDRLOG2_[n]	Header Log Register 2
PCIE_HDRLOG3_[n]	Header Log Register 3

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_IATU_CTL1_INB_[n]	iATU Region Control 1 Inbound Register
PCIE_IATU_CTL1_OUTB_[n]	iATU Region Control 1 Outbound Register
PCIE_IATU_CTL2_INB_[n]	iATU Region Control 2 Inbound Register
PCIE_IATU_CTL2_OUTB_[n]	iATU Region Control 2 Register Outbound
PCIE_IATU_CTL3_INB_[n]	iATU Region Control 3 Register
PCIE_IATU_CTL3_OUTB_[n]	iATU Region Control 3 Register
PCIE_IATU_LBADDR_INB_[n]	IATU Lower Base Inbound Address Register
PCIE_IATU_LBADDR_OUTB_[n]	IATU Lower Base Outbound Address Register
PCIE_IATU_LMTADDR_INB_[n]	IATU Inbound Limit Address Register
PCIE_IATU_LMTADDR_OUTB_[n]	IATU Outbound Limit Address Register
PCIE_IATU_LTADDR_INB_[n]	IATU Lower Target Address Inbound Register
PCIE_IATU_LTADDR_OUTB_[n]	IATU Lower Target Address Outbound Register
PCIE_IATU_UBADDR_INB_[n]	IATU Upper Base Address Inbound Register
PCIE_IATU_UBADDR_OUTB_[n]	IATU Upper Base Address Outbound Register
PCIE_IATU_UTADDR_INB_[n]	iATU Upper Target Address Inbound Register
PCIE_IATU_UTADDR_OUTB_[n]	iATU Upper Target Address Outbound Register
PCIE_IATU_VWPRT_[n]	IATU View Port Register
PCIE_LANE_SKEW_[n]	Lane Skew Register
PCIE_LNK_CAPB2_[n]	Link Capabilities 2 Register
PCIE_LNK_CAPB_[n]	Link Capabilities Register
PCIE_LNK_CTLSTAT2_[n]	Link Control 2 and Status 2 Register
PCIE_LNK_CTLSTAT_[n]	Link Control and Status Register
PCIE_MISC_CTL1_[n]	DBI Read-Only Write Enable Register
PCIE_MSI_CAPB_NPTR_[n]	MSI Capability ID, Next Pointer and Control Register
PCIE_MSI_CAPB_OFF4_[n]	MSI Capability Offset Register
PCIE_MSI_CAPB_OFF8_[n]	MSI Capability Offset Register
PCIE_MSI_CAPB_OFFC_[n]	MSI Capability Offset Register
PCIE_MSI_CTL_LADDR_[n]	MSI Controller Lower Address Register
PCIE_MSI_CTL_UADDR_[n]	MSI Controller Upper Address Register
PCIE_MSI_GPIO_IO_[n]	MSI Controller General-Purpose IO Register
PCIE_MSI_IEN0_[n]	MSI Controller Interrupt 0 Enable Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_MSI_IEN1_[n]	MSI Controller Interrupt 1 Enable Register
PCIE_MSI_IEN2_[n]	MSI Controller Interrupt 2 Enable Register
PCIE_MSI_IEN3_[n]	MSI Controller Interrupt 3 Enable Register
PCIE_MSI_IEN4_[n]	MSI Controller Interrupt 4 Enable Register
PCIE_MSI_IEN5_[n]	MSI Controller Interrupt 5 Enable Register
PCIE_MSI_IEN6_[n]	MSI Controller Interrupt 6 Enable Register
PCIE_MSI_IEN7_[n]	MSI Controller Interrupt 7 Enable Register
PCIE_MSI_IMSK0_[n]	MSI Controller Interrupt 0 Mask Register
PCIE_MSI_IMSK1_[n]	MSI Controller Interrupt 1 Mask Register
PCIE_MSI_IMSK2_[n]	MSI Controller Interrupt 2 Mask Register
PCIE_MSI_IMSK3_[n]	MSI Controller Interrupt 3 Mask Register
PCIE_MSI_IMSK4_[n]	MSI Controller Interrupt 4 Mask Register
PCIE_MSI_IMSK5_[n]	MSI Controller Interrupt 5 Mask Register
PCIE_MSI_IMSK6_[n]	MSI Controller Interrupt 6 Mask Register
PCIE_MSI_IMSK7_[n]	MSI Controller Interrupt 7 Mask Register
PCIE_MSI_ISTAT0_[n]	MSI Controller Interrupt 0 Status Register
PCIE_MSI_ISTAT1_[n]	MSI Controller Interrupt 1 Status Register
PCIE_MSI_ISTAT2_[n]	MSI Controller Interrupt 2 Status Register
PCIE_MSI_ISTAT3_[n]	MSI Controller Interrupt 3 Status Register
PCIE_MSI_ISTAT4_[n]	MSI Controller Interrupt 4 Status Register
PCIE_MSI_ISTAT5_[n]	MSI Controller Interrupt 5 Status Register
PCIE_MSI_ISTAT6_[n]	MSI Controller Interrupt 6 Status Register
PCIE_MSI_ISTAT7_[n]	MSI Controller Interrupt 7 Status Register
PCIE_PHY_TXDEEMPH	TX De-emphasis Parameters Register
PCIE_PHY_TXSWING	TX Launch Amplitude Register
PCIE_PLCTL_[n]	Port Logic Link Control Register
PCIE_PLDBG0_[n]	Port Logic Debug0 Register
PCIE_PLDBG1_[n]	Port Logic Debug1 Register
PCIE_PL_FRC_[n]	Port Force Link Register
PCIE_PL_PHYCTL_[n]	Port Control PHY Control Register
PCIE_PL_PHYSTAT_[n]	Port Control PHY Status Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_PMT_CAPB_[n]	Power Management Capabilities Register
PCIE_PMT_CTLSTAT_[n]	Power Management Control and Status Register
PCIE_QSTAT_[n]	Queue Status Register
PCIE_RC_BRDG_ILPCTL_[n]	Interrupt Bridge Line and Pin Control Register
PCIE_RC_CAPBPTR_[n]	Capability Pointer Register.
PCIE_RC_CCRID_[n]	Class Code and Revision ID Register
PCIE_RC_CFG_[n]	Root Complex Configuration Register
PCIE_RC_CTLCAPB_[n]	Root Control and Capabilities Register
PCIE_RC_DEVCAPB_[n]	Device Capabilities Register
PCIE_RC_ERRCMD_[n]	Root Complex Error Command Register
PCIE_RC_ERRSTAT_[n]	Root Error Status Register
PCIE_RC_ID_[n]	Device ID and Vendor ID Register
PCIE_RC_IOBL_[n]	Root Complex I/O Base and Limit Upper 16 bits Register
PCIE_RC_MBL_[n]	Memory Base and Memory Limit Register
PCIE_RC_PREFMBL_[n]	Prefetchable Memory Base and Limit Register
PCIE_RC_PREF_BUPP_[n]	Prefetchable Base Upper 32 Bits Register
PCIE_RC_PREF_LMT_UP_[n]	Prefetchable Limit Upper 32 Bits Register
PCIE_RC_ROMCFG_[n]	Root Complex Expansion ROM Base Address Register
PCIE_RC_SECSTAT_[n]	Secondary Status and I/O Base and Limit Register
PCIE_RC_STATCMD_[n]	Command and Status Register
PCIE_RC_STAT_[n]	Root Status Register
PCIE_RC_TMRLAT_[n]	Latency Timer Register
PCIE_RSCK_CTL	Reset and Clock Control Register
PCIE_RSCK_STAT	Reset and Clock Status
PCIE_SYM_TMRFILT1_[n]	Symbol Timer Filter 1 Off
PCIE_TMRCTL_MFN_[n]	Timer Control and Max Function Number Register
PCIE_TXCOMP_FCCSTAT_[n]	Transmit Completion FC Credit Status Register
PCIE_TX_NPST_FCCSTAT_[n]	Transmit Non-Posted FC Credit Status Register
PCIE_TX_PST_FCCSTAT_[n]	Transmit Posted FC Credit Status Register
PCIE_UNCOR_ERRMSK_[n]	Uncorrectable Error Mask Register
PCIE_UNCOR_ERRSEV_[n]	Uncorrectable Error Severity Register

Table 29-1: ADSP-SC58x PCIE Register List (Continued)

Name	Description
`PCIE_UNCOR_ERRSTAT_[n]`	Uncorrectable Error Status Register
`PCIE_VC0_COMP_RXQCTL_[n]`	Segmented-Buffer VC0 Completion Receive Queue Control Register
`PCIE_VC0_NPST_RXQCTL_[n]`	Segmented-Buffer VC0 Non-Posted Receive Queue Control Register
`PCIE_VC0_PST_RXQCTL_[n]`	Segmented-Buffer VC0 Posted Receive Queue Control Register
`PCIE_VC_TXARB1_[n]`	Virtual Channel Transmit Arbitration Register 1
`PCIE_VC_TXARB2_[n]`	Virtual Channel Transmit Arbitration Register 2
`PCIE_VEND_DLLP_[n]`	Vendor Specific DLLP Register

ADSP-SC58x PCIE Interrupt List

Table 29-2: ADSP-SC58x PCIE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
243	PCIE0_RESET	PCIE0 Reset		
244	PCIE0_STAT	PCIE0 Status		
245	PCIE0_DMA	PCIE0 DMA Completion		

ADSP-SC58x PCIE Trigger List

Table 29-3: ADSP-SC58x PCIE Trigger List Masters

Trigger ID	Name	Description	Sensitivity
135	`PCIE0_DMA`	PCIE0 DMA Done	

Table 29-4: ADSP-SC58x PCIE Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
		None	

Definitions

The following definitions are helpful when using the PCIE module.

BAR

Base Address register

CPL

Completion

DBI

Data Bus Interface

DLLP

Data Link Layer Packet

DSP

Downstream Port

EP

PCIe Endpoint

iATU

Internal Address Translation Unit

Inbound Traffic

Transactions that enter the PCIe port from the wire side and are sent out onto the master SCB interface.

LOS

Loss of signal

NP

Non Posted

Outbound Traffic

Transactions that enter the PCIe port from the slave SCB interface and are sent out onto the PCIe wire

PCIe Host

The processor running the PCIe software on the RC device

PCIe software

Software that configures and controls the whole PCIe system.

PIPE

Standard interface between the PCIe PHY and the core

TLP

Transaction Layer Packet

USP

Upstream Port

VC

Virtual Channel

VMI

Vendor Message Interface

Top-level Block Diagram and Description

The ADI implementation of the PCIe interface is shown in the ***PCIe Interface Block Diagram*** below, and comprises the modules described in the following section.

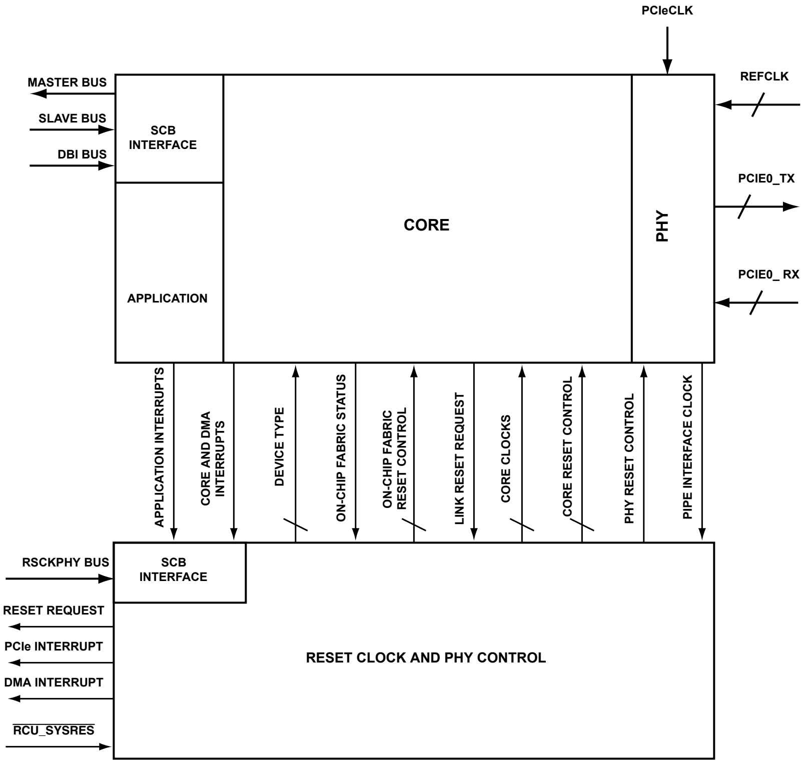

PCIeCLK
MASTER BUS
SLAVE BUS
DBI BUS
SCB INTERFACE
APPLICATION
CORE
PHY
REFCLK
PCIE0_TX
PCIE0_ RX
APPLICATION INTERRUPTS
CORE AND DMA INTERRUPTS
DEVICE TYPE
ON-CHIP FABRIC STATUS
ON-CHIP FABRIC RESET CONTROL
LINK RESET REQUEST
CORE CLOCKS
CORE RESET CONTROL
PHY RESET CONTROL
PIPE INTERFACE CLOCK
RSCKPHY BUS
SCB INTERFACE
RESET REQUEST
PCIe INTERRUPT
DMA INTERRUPT
RCU_SYSRES
RESET CLOCK AND PHY CONTROL

Figure 29-2: PCIe Interface Block Diagram

Module-level Block Diagrams and Descriptions

The following modules are integrated in the PCIe port:

- Reset clock control and PHY control (RSCKPHY). Controls the PCIe port's clocking and reset functions. It can be accessed by on-chip fabric masters regardless of the dual-mode core and PHY status.
- Application (APP). ADSP-SC58x specific application registers and logic.
- The PCIe core and PCIe PHY. Form the heart of the PCIe interface and implement the PCIe protocol stack, and the physical pins, connected to each other through the PIPE interface.

Registers and memory space are accessed over the following system cross bar (SCB) ports:

- DBI SCB slave interface. PCI, PCIe, port logic and application registers.
- Slave SCB interface. PCIe memory space across the PCIe link.
- Master SCB port. On-chip address space accessed from a PCIe partner over the PCIe link.
- RSCKPHY SCB port. RSCKPHY and PCIe PHY registers.

Reset Clock and PHY Control Module (RSCKPHY)

The RSCKPHY block includes the clock and reset circuitry and PCIe PHYs control and status registers. It also drives the bus to configure the PCIe core to either a Root Complex (RC) or an End Point (EP) port. The whole PCIe port is reset by the `RCU_SYSRES` signal and enabled by software.

While booting through the PCIe port is not one of the default ADSP-SC58x boot modes, applications can choose to implement a second stage boot loader kernel that configures and enables the PCIe port to load a custom application into the ADSP-SC58x memory.

The RSCKPHY block generates all clocks and reset signals for the PCIe core and PCIe PHY.

RSCKPHY System Cross Bar (SCB) Port

On-chip masters can access the RSCKPHY MMRs through a dedicated SCB port in the SYSCLK clock domain.

PCIe Reset

All on-chip modules are reset if a system reset (`RCU_SYSRES` asserted) event occurs. The PCIe APP, Core, PHY and Core's SCB Interface are taken out of reset only by software. After a Hardware or System Reset only the RSCKPHY block is out of reset. The `PCIE_RSCK_CTL.PCIERES` bit controls the assertion and deassertion of the PCIe cold reset signal. When the `PCIE_RSCK_CTL.PCIERES` bit is cleared, the core, application, PHY and the core's SCB interface are brought out of reset and their clocks are enabled. To satisfy the PHYs Power-On-Reset timing, RSCKPHY delays the deassertion of its cold reset output by at least 25000 SYSCLK cycles.

The PCIe core and PCIe PHY mode of operation (Root Complex or End Point) is determined by the value of the `PCIE_RSCK_CTL.DEVTYP` bit field at the time the `PCIE_RSCK_CTL.PCIERES` bit is cleared (both fields can be written at the same time). Writing values other than 0x0 or 0x4 to the `PCIE_RSCK_CTL.DEVTYP` bit field is not allowed.

To reset the PCIe port (warm or hot reset), the PCIe core asserts its Link Reset Request signal to make RSCKPHY assert Reset Request signal. This signal is used as an interrupt. Writing 1 to the `PCIE_RSCK_CTL.DBISLVDIS` bit enables the PCIe port reset. The application software may also decide to assert the Power Reset signals by setting the `PCIE_RSCK_CTL.PCIERES` bit.

Clocking

The PHY uses external reference clock (REFCLK_P and REFCLK_N). These are the external reference 100 MHz differential clocks. If the PHYs input clock has been stable and running for more than 10 µsec, the additional delay can be disabled by clearing the `PCIE_RSCK_CTL.RESDELEN` bit. Software can request to remove the reference clock by setting the `PCIE_RSCK_CTL.RFCKRMREQ` bit. The `PCIE_RSCK_STAT.RFCKRMEN` bit shows when the reference clock can be removed.

PCLK is generated by the PHY TX PLL from the reference clock. Its frequency changes when the PCIe port transitions from one frequency to another. CORE_CLK is the primary clock for the core. It is identical to PCLK.

The SCB interface synchronizes between the SCB buses in the SYSCLK domain and the core in the CORE_CLK domain. From the PCIe perspective there are no restrictions on SYSCLK other than changing frequencies when the interface is idle.

AUX_CLK is the clock source for parts of the Power Management Controller module. It is implemented as a low frequency (f_{SYSCLK}/16) clock derived from SYSCLK.

Power Management

An EP needs to send a Power Management Event (PM_PME) message to the RC to wake up from a power down state. For the message to be sent, the Power Management Controller (PMC) in the PCIe core module needs to wake up and the PCIe link brought to its L0 state. This process is triggered by the application software writing 1 to the `PCIE_RSCK_CTL.PMPMEREQ` bit.

It is the application software responsibility not to write 1 to the `PCIE_RSCK_CTL.PMPMEREQ` bit until the previous message has been transmitted.

Reset

The ***Reset Summary*** table provides descriptions of the PCIe reset states.

Table 29-5: Reset Summary

Reset State	Triggered By	Description
System Reset	Reset Control Unit	This is a full-chip reset state. It is triggered by Hard or System Reset events (See the RCU chapter of the HRM). Resets the Reset Clock and PHY Control, Application, Core, Core's SCB Interface and PHY.
Cold Reset	`PCIE_RSCK_CTL.PCIERES` bit	This state is entered if `PCIE_RSCK_CTL.PCIERES` is set. All the PCIe modules except RSCKPHY are reset. Resets the Application, Core, Core's SCB Interface and PHY. To exit Cold Reset software must clear `PCIE_RSCK_CTL.PCIERES`
Warm Reset	Core	This state is entered if the PCIe core requests a link reset. All the PCIe modules except RSCKPHY and sticky bits and registers are reset. Resets the Application, Core, Core's SCB Interface and PHY except sticky bits.
Hot Reset	PCIe Link	This state is entered if a downstream port needs to reset an upstream port asserting a downstream reset request. All the PCIe modules except RSCKPHY and sticky bits and registers are reset. Resets the Application, Core, Core's SCB Interface and PHY except sticky bits.

Responding to a Warm or Hot Reset

During normal operation when the link has to be reset, the PCIe core asserts the link reset request output. The PCIe core and PHY is not reset until that PHY has had a chance to transmit the EIOS.

When the link reset request signal is asserted, the SCB bridge operates as follows:

System cross bar bridge slave interface (including DBI slave interface):

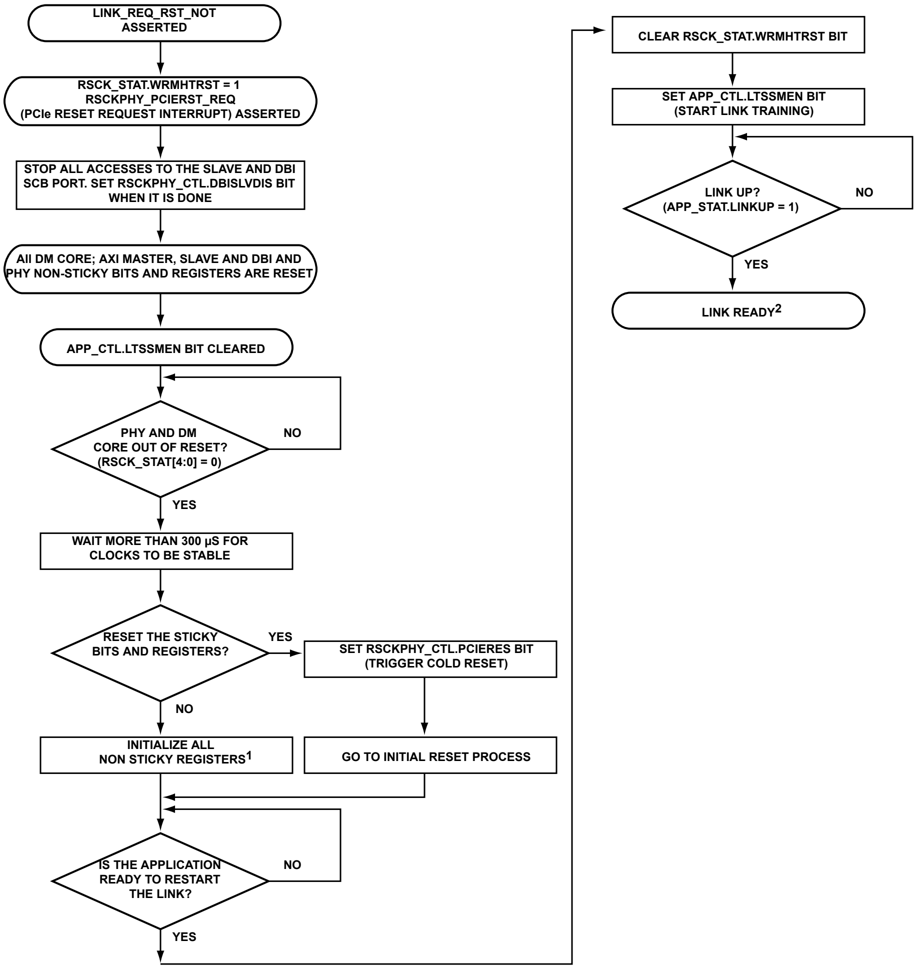
- Keeps accepting any new read/write requests but does not forward them to the PCIe link or application registers.
- For write requests:
  - Allows the application SCB master to complete all write data bursts and dumps data internally in the bridge.
  - On the completion of write data burst, generates a write response with slave error status.
- Generates a slave error response for all SCB beats of read requests.

  For more details, see PCI Express Controller DM Port Core Databook “Generating a Warm Reset (Link Down Reset)” and “Generating and Processing Hot Resets (Training Resets)”.

System cross bar bridge master interface:

- Stops generating any new read and write requests.
- Waits for (and dumps) the completion of all responses from the application for every outstanding command.

The reset request signal is asserted in response to link reset requests assertion. It is the responsibility of the application software to stop all accesses to the slave SCB and DBI SCB ports. Once all accesses are stopped, software must write 1 to the `PCIE_RSCK_CTL.DBISLVDIS` bit to reset the PCIe core. The application software should implement the process shown in the ***Hot Reset Process*** figure.


LINK_REQ_RST_NOT ASSERTED
RSCK_STAT.WRMHTRST = 1 RSCKPHY_PCIERST_REQ (PCIe RESET REQUEST INTERRUPT) ASSERTED
STOP ALL ACCESSES TO THE SLAVE AND DBI SCB PORT. SET RSCKPHY_CTL.DBISLVDIS BIT WHEN IT IS DONE
All DM CORE; AXI MASTER, SLAVE AND DBI AND PHY NON-STICKY BITS AND REGISTERS ARE RESET
APP_CTL.LTSSMEN BIT CLEARED
PHY AND DM CORE OUT OF RESET? (RSCK_STAT[4:0] = 0)
NO
YES
WAIT MORE THAN 300 µS FOR CLOCKS TO BE STABLE
RESET THE STICKY BITS AND REGISTERS?
YES
SET RSCKPHY_CTL.PCIERES BIT (TRIGGER COLD RESET)
NO
INITIALIZE ALL NON STICKY REGISTERS1
GO TO INITIAL RESET PROCESS
IS THE APPLICATION READY TO RESTART THE LINK?
NO
YES
CLEAR RSCK_STAT.WRMHTRST BIT
SET APP_CTL.LTSSMEN BIT (START LINK TRAINING)
LINK UP? (APP_STAT.LINKUP = 1)
NO
YES
LINK READY2

NOTES:
1. Internal Address Translation and Embedded DMA registers have to be reprogrammed, and the custom configuration space setting must be re-written.
2. If configured as an EP, the application logic must wait for the PCIE_EP_STATCMD_n.BME bit to be set in order to access the PCIe address space.
If configured as a RC, the application logic must wait for the PCIE_RC_STATCMD_[n].MSE and the PCIE_RC_STATCMD_[n].IOE bits to be set in order to generate any MEM or I/O requests.

Figure 29-3: Hot Reset Process

The RSCKPHY block resets the PCIe core and PCIe_PHY modules.

Hot reset is similar to a warm reset. The upstream port requests the RSCKPHY to reset the core in response to a reset request sent by the downstream port. If the PCIe core is configured as a root complex port, the application may request the core to reset upstream devices by writing 1 (across the DBI port) to the `PCIE_APP_CTL.INITDSRST` bit. Writing one to this bit makes the APP module assert its APP_INIT_RST to the PCIe core.

In this case, the process involves the RC and EP devices on both sides of the PCIe link. Writing to the `PCIE_APP_CTL.INITDSRST` bit occurs in the RC device. The reset request is transmitted over the PCIe link to the EP device. Then the EP core starts the hot reset sequence.

The application can send new requests to the core only after verifying that the link is up again.

Application Module (APP)

The APP module implements the application-specific control and status logic and registers. Registers are mapped to the PCIe core configuration space. The SCB master accesses these registers across the DBI bus. When the port is configured as an End Point (EP), the remote PCIe device can also access registers across the PCIe link.

This module enables the application software running on the local core to handle PCIe interrupts and messages, and to access general link information. This entire module resides in the PCIe core and is reset when the PCIe core is reset.

APP Registers Access

Application registers are mapped to the upper DBI address or configuration space. Local cores can access these registers regardless of the PCIe core configuration. In EP mode, these registers can also be accessed from the PCIe wire using CFG requests or with BAR matched MEM requests.

The following limitations apply to SCB accesses:

- SCB transfers to the APP address space must be 32-bits wide. Errors are not reported when the size is not equal to 32 bits
- Byte access within a 32-bit double word is not supported
- Zero-byte writes should not be issued

Accessing the ADI-specific PCIe registers enables software to control PCIe features such as Message Signaled Interrupts (MSI) and power management messages, and to debug the PCIe core.

Interrupts When Configured as a PCIe End Point (EP)

The APP enables the application software to signal an interrupt to the RC over the PCIe serial link. Interrupts are asserted in one of two ways, as message signaled interrupts or as an INTA assert message. Accessing APP registers is enabled as long as the core is out of reset, regardless of the PCIe core configuration. The application software should not trigger an interrupt when the PCIe core is configured as RC.

The `PCIE_APP_STAT.MSIEN` `PCIE_APP_STAT.MSIEN` bit reflects the state of the MSI enable bit in the MSI control register. If `PCIE_APP_STAT.MSIEN` is set, the interrupt request is converted into an MSI message. The `PCIE_APP_STAT.OUTSTMSI` bit reflects the status of the previous Interrupt request when the `PCIE_APP_STAT.MSIEN` bit is set.

The `PCIE_APP_STAT.OUTSTMSI` bit signals that an MSI interrupt has been requested but the PCIe core has not accepted it.

The PCIe core grants the MSI request by asserting MSI_GRANT. The `PCIE_APP_STAT.OUTSTMSI` bit and `PCIE_APP_STAT.MSIGRANT` bits are cleared and set, respectively.

If `PCIE_APP_STAT.MSIEN` is cleared, the interrupt request is converted into a PCI legacy INTA assert message. The `PCIE_APP_STAT.OUTSTINTA` bit reflects the status of the previous interrupt request when `PCIE_APP_STAT.MSIEN` is cleared.

The `PCIE_APP_STAT.OUTSTINTA` bit signals that an INTA interrupt has been requested. The `PCIE_APP_STAT.OUTSTINTA` bit does not indicate that the RC device is processing the interrupt.

Interrupts when Configured as an RC

An MSI can be received in two ways. If the SCB MSI controller is enabled, MSIs do not reach the master SCB port and generate a PCIe interrupt instead. MSI status and control registers are part of the port logic registers. If the SCB MSI controller is disabled, MSIs reach the master SCB port as a write transaction to a predetermined address.

When configured as an RC, the PCIe core accepts Assert_INTx and Deassert_INTx messages from the downstream component and signal the APP module to trigger a PCIe interrupt. Incoming interrupts are registered in the `PCIE_APP_INTSTAT` (legacy interrupts status) register. The RC can also write directly to the APP registers to trigger an interrupt deassert message on the EP device.

Inbound Messages

Inbound messages are dropped and not forwarded to the master SCB interface. Instead, these messages are registered and a PCIe interrupt triggered to report the reception of a message. Message interrupts can be individually enabled by setting the appropriate interrupt enable bit in the `PCIE_APP_CTL` register. Each type of message has a corresponding bit in the `PCIE_APP_INTSTAT` register. These bits are set to signal the reception of a message and must be cleared to receive a new message of the same type.

Table 29-6: Inbound Messages Support

Message Type	Interrupt Enable **`PCIE_APP_CTL`** Register	Interrupt Status **`PCIE_APP_INTSTAT`** Register	Header 4th Double Word	Header 3rd Double Word	Requester ID (BDF)
Vendor	VENIEN	VENDMSG	APP_VNDR_MSG_HDR0	APP_VNDR_MSG_HDR1	APP_VNDR _MSG
Unlock	UNLKIEN	UNLK	Not Applicable	Not Applicable	APP_UNLK _MSG
Correctable Error	CORERRIEN	CORERR	Not Applicable	Not Applicable	APP_CORERR _MSG
Non Fatal Error	NFTLERRIEN	NFTLERR	Not Applicable	Not Applicable	APP_NFERR_MSG
Fatal Error	FERRIEN	FTLERR	Not Applicable	Not Applicable	APP_FERR _MSG
Power Management Event	PMEIEN	PME	Not Applicable	Not Applicable	APP_PMEVT_MSG
Power Management Acknowledge	PMTOACKIEN	PMTOACK	Not Applicable	Not Applicable	APP_PMACK_MSG
Power Management Turn Off	PMTOFFIEN	PMTOFF	Not Applicable	Not Applicable	APP_PMTOFF _MSG

Outbound Messages

In addition to MSI and PCI legacy interrupts, the transmission of power management turn-off or unlock messages is triggered by writing 1 to the corresponding `PCIE_APP_CTL.PMOFFREQ` or `PCIE_APP_CTL.UNLKREQ` bits. Power management enable messages are triggered by writing 1 to the `PCIE_RSCK_CTL.PMPMEREQ` bit. These two messages can be triggered when the PCIe core is configured as an RC. It is the responsibility of the application software to not start a new message until the previous message has been transmitted. The core does not return an acknowledgment or grant signal.

Diagnostics

APP also includes a diagnostic feature that enables the application software to inject LCRC and ECRC errors into the datapath of the core.

To inject an LCRC error, the application software must write a 1 to the `PCIE_APP_DIAGCTL.LCRCGEN` bit and initiate a packet transmission. To inject an ECRC error, the application software must write a 1 to the `PCIE_APP_DIAGCTL.ECRCERR` bit and initiate a packet transmission. The `PCIE_APP_DIAGSTAT.LCRC` bit is set when LCRC error packets are transmitted. The `PCIE_APP_DIAGSTAT.ECRC` bit is set when ECRC error packets are transmitted.

This handshake between control and status allows the application to control a specific packet injection with a CRC or ECRC error. Inverting the last bit of an LCRC or ECRC value generates the LCRC and ECRC errors.

Architectural Concepts

The following sections describe PCIe architectural features supporting register and address space accesses, configuration, and PHY control.

Register Accesses (PCIe Link and SCB DBI Port)

The PCIe registers are used to configure and control the PCIe port. The registers are accessed:

- across the DBI SCB interface or over the PCIe link using memory read or write requests, or
- using CFG requests when the DM core is configured as an EP

Accesses over the DBI SCB port must be 32 bits wide with addresses aligned to the data width (address [1:0] = 00).

These registers are divided into two groups. The first group of registers is implemented inside the PCIe core. It includes the PCI configuration header, PCI standard capability structures, PCIe extended capability structures and port logic. The second group of registers is implemented inside the APP module. It includes the application-specific registers.

PCIe link configuration transfers access the PCIe core registers when the registers address is less than 0xC00. It transfers using APP registers when the address is greater than or equal to 0xC00. The PCI configuration header, PCI standard capability structures, PCIe are PCIe core configuration registers specified by the PCI Express specification revision 3.0. The PCI Express 3.0 specification does not specify the port logic registers. These registers are accessed as memory-mapped registers or normally using CFG requests. A PCIe master can also access the PCIe registers using memory accesses to the local DBI SCB address space.

On-chip SCB masters have no concept of MEM, I/O, or CFG transfers. The PCIe core registers are mapped to the lower 4 KB addresses of the PCIe DBI region (DBI address [13:12] = 00). To enable writes to some read-only and shadow registers, PCIe registers are also mapped onto the 4 KB region (DBI write address [13:12] = 01). APP registers are mapped to the higher 4 KB addresses of the PCIe DBI region (address [13:12] = 1x).

SOC Address Space Accesses

The SCB master interface enables a remote PCIe device to read and write to a slave connected to the on-chip fabric. The ***PCIe Address Space*** table provides details on the address space for the processor.

Table 29-7: PCIe Address Space

Name	Base Address	End Address
PCIE-DBI*1	0x310B_8000	0x310B_AFFF
PCIE-RSCKPHY	0x310B_B000	0x310B_BFFF

*1 In the ADSP-SC58x, the PCIe and application registers are mapped as follows:

PCIe core registers 0x310B_8000 to 0x310B_8FFF

PCIe core write enabled registers 0x310B_9000 to 0x310B_9FFF

APP registers 0x310B_A000 to 0x310B_AFFF

System Cross Bar (SCB) Master Features

- Address bus width is 32-bit
- Data bus width is 32-bit
- Only incremental bursts are supported
- Maximum allowable burst length is 16 Beats
- Graceful flush and reset when link is down
- No exclusive access support
- Only posted writes are expected, therefore the master SCB port ignores the write response from the SCB slave.
- PCIe Advanced Error Reporting (AER) is not supported. AER is only supported with respect to the native PCIe core. Errors detected by the bridge are not reported as part of AER.
- Read requests above the 256 bytes limit are not supported

PCIe Address Space Accesses

The System Cross Bar slave interface enables a master on the on-chip fabric to read and write through the System Cross Bar bridge to a remote PCIe device. (In the ADSP-SC58x processor, the PCIe address space size is 256 MB mapped to addresses 0x5000_0000 to 0x5FFF_FFFF). The Internal Address Translation Unit (iATU) enables the mapping of these 256 MB to anywhere in the PCIe memory space.

Table 29-8: ADSP-SC58x System Component Memory Address Map

ADSP-SC58x System Component Memory Address Map			ARM View	System View	SHARC View			
			All Accesses	All Accesses	Data Accesses		Code Execution	
			Byte	Byte	Byte	NW	VISA	Non-VISA
Memory	Sub-division	Group range -->	0000_0000- FFFF_FFFF	2000_0000- FFFF_FFFF	2000_0000- FFF_FFFF	100_0000- 1FFF_FFFF	80_0000- FF_FFFF	40_0000- 7F_FFFF
PCIe Data	256MB	5000_0000	5007_FFFF	5000_0000	5007_FFFF	5000_0000	5007_FFFF	200_0000
		5008_0000	5017_FFFF	5008_0000	5017_FFFF	5008_0000	5017_FFFF	202_0000
		5018_0000	57FF_FFFF	5018_0000	57FF_FFFF	5018_0000	57FF_FFFF	206_0000
		5800_0000	5FFF_FFFF	5800_0000	5FFF_FFFF	5800_0000	5FFF_FFFF	N/A

SCB Slave Port Features

- Address bus width is 32-bit
- Data bus width is 32-bit
- Only incremental bursts are supported
- Maximum allowable burst length is 16 Beats
- Graceful flush and reset when link is down
- No exclusive access support
- No write interleaving with reads
- No write interleaving with writes of different AWIDs
- Outbound write burst with zero or partial (but contiguous) write strobes set is only supported for the first and last beats of a burst.
- All accesses are memory accesses (always posted)
- Write responses are generated by the PCIe core after the write request and are received
- PCIe Advanced Error Reporting (AER) is not supported. AER is only supported with respect to the native PCIe core. Errors detected by the bridge are not reported as part of AER.
- Read requests above the 256 Bytes limit are not supported

Message Signaled Interrupts (MSI) Controller

The SCB Interface block provides an optional programmable MSI controller to detect and terminate inbound MSI requests. Rather than propagating MSI MWr TLPs onto the SCB bus through the master interface; the MSI packets are captured and terminated in the SCB Interface block and an interrupt is signaled locally.

The MSI Controller is programmed with an address that is used as the system MSI address. When an inbound (received) MWr request is passed to the SCB interface and matches the specified MSI address, as well as the conditions specified for an MSI memory write request, then an MSI interrupt is detected. This memory write request is not driven onto the SCB master interface. The MSI Controller decodes the MSI MWr data payload to determine which endpoint device (EP) sent the MSI and which interrupt vector the MSI corresponds to. When a valid interrupt has been decoded, the `PCIE_APP_INTSTAT.MSI` bit is set. This bit remains set when any MSI interrupt is pending. It is only de-asserted when there is no MSI interrupt pending. The controller has a full set of MSI registers.

Features

- MSI interrupt controller only enabled in RC mode when the `PCIE_RSCK_CTL.DEVTYP` bit field =0x4. It is inactive in EP mode.
- The MSI interrupt controller provides support for up to eight EPs. Each supported EP has a set of interrupt enable, mask, and status registers.
- Guarantees correct SCB ordering with respect to other inbound posted writes by generating the MSI interrupt only after an SCB slave acknowledges responses of previous posted TLPs.
- A maximum of 32 interrupts are supported per EP.

MSI Request Detection Criteria

An MSI interrupt request is detected when the core receives a MWr TLP that satisfies the following conditions:

- Header attributes bits are zero. No snoop (NS) and relaxed ordering (RO) must be zero.
- Length field is 0x01 to indicate a payload of one double word.
- First byte enable (FBE) must be such that it is enabling the first two bytes (16 bits) of the payload.
- Last byte enable (LBE) is 4b0000.
- TLP address corresponds to systems chosen MSI address as programmed in the MSI Controller Address register (`PCIE_MSI_CTL_LADDR_[n]` and `PCIE_MSI_CTL_UADDR_[n]`). This register is not the MSI Lower 32 Bits Address Register which is part of the PCI Express MSI capability register structure In addition to these criteria the MWr must also pass the receive filtering rules.

The following table shows the MSI Controller software register map. The tables show the address's offset. The MSI Controller registers base address is 0x310B8700.

Table 29-9: MSI Controller Software Registers

Byte Offset	Name	Description	Reset	Access
+0x120	PCIE_MSI_CTL_LADDR_[n]	MSI Controller Address Register.	0x00000000	R/W
	PCIE_MSI_CTL_UADDR_[n]	MSI Controller Upper Address Register.	0x00000000	R/W
	PCIE_MSI_IEN0_[n]	MSI Controller Interrupt#0 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK0_[n]	MSI Controller Interrupt#0 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT0_[n]	MSI Controller Interrupt#0 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN1_[n]	MSI Controller Interrupt#1 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK1_[n]	MSI Controller Interrupt#1 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT1_[n]	MSI Controller Interrupt#1 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN2_[n]	MSI Controller Interrupt#2 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK2_[n]	MSI Controller Interrupt#2 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT2_[n]	MSI Controller Interrupt#2 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN3_[n]	MSI Controller Interrupt#3 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK3_[n]	MSI Controller Interrupt#3 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT3_[n]	MSI Controller Interrupt#3 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN4_[n]	MSI Controller Interrupt#4 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK4_[n]	MSI Controller Interrupt#4 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT4_[n]	MSI Controller Interrupt#4 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN5_[n]	MSI Controller Interrupt#5 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK5_[n]	MSI Controller Interrupt#5 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT5_[n]	MSI Controller Interrupt#5 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN6_[n]	MSI Controller Interrupt#6 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK6_[n]	MSI Controller Interrupt#6 Mask Register.	0x00000000	R/W
	PCIE_MSI_ISTAT6_[n]	MSI Controller Interrupt#6 Status Register.	0x00000000	R/W
	PCIE_MSI_IEN7_[n]	MSI Controller Interrupt#7 Enable Register.	0x00000000	R/W
	PCIE_MSI_IMSK7_[n]	MSI Controller Interrupt#7 Mask Register.	0x00000000	R/W

Table 29-9: MSI Controller Software Registers (Continued)

Byte Offset	Name	Description	Reset	Access
	PCIE_MSI_ISTAT7_[n]	MSI Controller Interrupt#7 Status Register.	0x00000000	R/W
	PCIE_MSI_GPIO_IO_[n]	MSI Controller General Purpose IO Register.	0x00000000	R/W

Internal Address Translation (iATU)

The Address Translation Unit is used for mapping internal/external memory regions to external/internal memory regions. If an outbound transaction falls into one of the enabled iATU regions, TLP headers are created replacing the default values by the corresponding fields in the regions iATU registers. Without address translation, application addresses are passed straight through to the SCB master/slave interfaces.

Inbound Address Translation Features

- Match mode operation for MEM, I/O, CFG, and MSG TLPs. No translation for completions.
- Selectable BAR Match mode operation for I/O and MEM TLPs.
-
    - TLPs destined for PCIe or APP registers in an upstream port are not translated.
    - TLPs that are not error-free (ECRC, malformed and so on) are not translated.
- Programmable TLP header field matching.
- Up to 5 address regions programmable for location and size.
- Programmable enable/disable per region.
- Automatic FMT field translation between three DWORDS and four DWORDS for 64-bit addresses.
- Invert Address Matching mode to translate accesses outside of a successful address match.
- ECAM Configuration Shift mode to allow a 256 MB CFG1 space to be located anywhere in the 64-bit address space.
- The minimum region size is 64 kB.

Outbound Address Translation Features

- Address Match mode operation for MEM and I/O, CFG, and MSG TLPs. No translation for completions.
- Supports type translation through TLP type header field replacement for MEM or I/O types to MSG/CFG types.
    - Includes posted to non-posted translation (for example, MWr to CfgWr0)
    - No translation from completions
- Programmable TLP header field replacement.

- Up to 4 address regions programmable for location and size.
- Programmable enable/disable per region.
- Automatic FMT field translation between three DWORDS and four DWORDS for 64-bit addresses.
- Invert Address Matching mode to translate accesses outside of a successful address match.
- Response code which defines the completion status to return for accesses matching a region.
- The minimum region size is 64 kB.
- DMA bypass mode to allow TLPs which are initiated by the embedded DMA engine, to pass through the iATU un-translated.

Configuration and Programming

The ***iATU Configuration*** table provides the basic values for configuring the iATU.

Table 29-10: iATU Configuration Summary

Parameter	Value	Notes
Maximum Number of Outbound Address Translation Regions	4	Enables the SCB to PCIe address mapping. The PCIe space is defined as 256 Mbytes.
Maximum Number of Inbound Address Translation Regions	5	Enables PCIe to SCB address mapping. Assuming one region dedicated to System L2, one to L1s in multiprocessing space, one to MMRs, one to ASYNC memory and one DDR memory.
Minimum size of an Address Translation Region	64 Kbytes	

The following table shows the iATU software register map. The table shows the address's offset. The iATU registers Base Address is 0x310B8700.

Table 29-11: iATU Software Registers

Byte Offset	Name	Description	Reset	Access
0x310B8900	`PCIE_IATU_VWPRT_[n]`	IATU ViewPort	0x00000000	R/W
0x310B8904	`PCIE_IATU_CTL1_INB_[n]`	iATU Region Control 1 Register Inbound	0x0XXX0XXX	R/W
0x310B8904	`PCIE_IATU_CTL1_OUTB_[n]`	iATU Region Control 1 Register Outbound	0x0XXX0XXX	R/W
0x310B8908	`PCIE_IATU_CTL2_OUTB_[n]`	iATU Region Control 2 Register Outbound	0x00000000	R/W
0x310B8908	`PCIE_IATU_CTL2_INB_[n]`	iATU Region Control 2 Register Inbound	0x00000000	R/W
0x310B890C	`PCIE_IATU_LBADDR_OUTB_[n]`	iATU Lower Base Address Outbound	0x00000000	R/W
0x310B890C	`PCIE_IATU_LBADDR_INB_[n]`	iATU Lower Base address Inbound	0x00000000	R/W
0x310B8910	`PCIE_IATU_UBADDR_OUTB_[n]`	iATU Upper Base Address Outbound	0x00000000	R/W
0x310B8910	`PCIE_IATU_UBADDR_INB_[n]`	iATU Upper Base address Inbound	0x00000000	R/W

Table 29-11: iATU Software Registers (Continued)

Byte Offset	Name	Description	Reset	Access
0x310B8914	PCIE_IATU_LMTADDR_OUTB_[n]	iATU Outbound Limit Address	0xFFFFFFFF	R
0x310B8914	PCIE_IATU_LMTADDR_INB_[n]	iATU Inbound Limit address	0xFFFFFFFF	R
0x310B8918	PCIE_IATU_LTADDR_INB_[n]	iATU Lower Target Address Inbound	0x00000000	R/W
0x310B8918	PCIE_IATU_LTADDR_OUTB_[n]	iATU Lower Target Address Outbound	0x00000000	R/W
0x310B891C	PCIE_IATU_UTADDR_INB_[n]	iATU Upper Target Address Inbound	0x00000000	R/W
0x310B891C	PCIE_IATU_UTADDR_OUTB_[n]	iATU Upper Target Address Outbound.	0x00000000	R/W
0x310B8920	PCIE_IATU_CTL3_INB_[n]	iATU Region Control 3 Register.	0x00000000	R/W
0x310B8920	PCIE_IATU_CTL3_OUTB_[n]	iATU Region Control 3 Register.	0x00000000	R/W

Embedded DMA Controller (eDMA)

The Embedded DMA Controller implements single DMA write and DMA read channels. It can simultaneously perform the following types of memory transactions.

- DMA Write. Copy a block of data in local memory to remote memory over the PCIe link.
- DMA Read. Copy a block of data in remote memory to local memory over the SCB fabric.

Read and write transfers are processed at the same time, and in parallel with normal (non-DMA) traffic. When a DMA transfer completes or an error occurs, the DMA optionally interrupts the local CPU or sends an interrupt MWr (IMWr) to the remote CPU. The DMA is highly configurable and can be programmed using the local DBI or over the PCIe wire. The channels are named *read channel 0* and *write channel 0*. There are a set of registers associated with each DMA channel that are used for programming the DMA channel to create a correct set of header fields for any TLP that it generates. To access some of these registers (DMA channel context registers), indirect addressing through an index register needs to be used.

The following tables show the DMA software register map. The tables show the address's offset. The eDMA registers base address is 0x310B8700.

Table 29-12: DMA Software Registers

Byte Offset	Description
+0x278	Number of DMA Channels
+0x27C	DMA Write Engine Enable
+0x280	DMA Write Doorbell
+0x288	DMA Write Engine Channel Arbitration Weight Low
+0x28C	DMA Write Engine Channel Arbitration Weight High
+0x29C	DMA Read Engine Enable
+0x2A0	DMA Read Doorbell

Table 29-12: DMA Software Registers (Continued)

Byte Offset	Description
+0x2A8	DMA Read Engine Channel Arbitration Weight Low
+0x2AC	DMA Read Engine Channel Arbitration Weight High

The DMA Interrupt registers are shown in the table below:

Table 29-13: DMA Interrupt Registers

Byte Offset	Description
+0x2BC	DMA Write Interrupt Status
+0x2C0	Not used (RsvdP)
+0x2C4	DMA Write Interrupt Mask
+0x2C8	DMA Write Interrupt Clear
+0x2CC	DMA Write Error Status
+0x2D0	DMA Write Done IMWr Address Low
+0x2D4	DMA Write Done IMWr Address High
+0x2D8	DMA Write Abort IMWr Address Low
+0x2DC	DMA Write Abort IMWr Address High
+0x2E0	DMA Write Channels 1 and 0 IMWr Data
+0x2E4	DMA Write Channels 3 and 2 IMWr Data
+0x2E8	DMA Write Channels 5 and 4 IMWr Data
+0x2EC	DMA Write Channels 7 and 6 IMWr Data
+0x300	DMA Write Linked List Error Enable
up to +0x30C	Not used (RsvdP)
+0x310	DMA Read Interrupt Status
+0x314	Not used (RsvdP).
+0x318	DMA Read Interrupt Mask
+0x31C	DMA Read Interrupt Clear
+0x320	Not used (RsvdP)
+0x324	DMA Read Error Status Low
+0x328	DMA Read Error Status High
up to +0x330	Not used (RsvdP)
+0x334	DMA Read Linked List Error Enable
+0x338	Not used (RsvdP)

Table 29-13: DMA Interrupt Registers (Continued)

Byte Offset	Description
+0x33C	DMA Read Done IMWr Address Low
+0x340	DMA Read Done IMWr Address High
+0x344	DMA Read Abort IMWr Address Low
+0x348	DMA Read Abort IMWr Address High
+0x34C	DMA Read Channels 1 and 0 IMWr Data
+0x350	DMA Read Channels 3 and 2 IMWr Data
+0x354	DMA Read Channels 5 and 4 IMWr Data
+0x358	DMA Read Channels 7 and 6 IMWr Data

Operations

The following sections provide information about PCIe operations including initialization, reading and writing memory and DMA.

Initialization

Initialization involves two main tasks:

- Configuring the PCIe port. This is accomplished by the application software immediately after taking the PCIe port out of reset.
- Initializing the PCIe. This is accomplished by the PCIe core as a result of the previous step or as a result of configuration writes received across the PCIe link.

Port Configuration

Clearing the `PCIE_RSCK_CTL.PCIERES` bit starts the reset exit sequence of the PCIe port. The `PCIE_RSCK_CTL.DEVTYP` bit determines if the PCIe core is configured as an End Point or a Root Complex port. The `PCIE_RSCK_CTL.DEVTYP` bit has to be stable before taking the PCIe core out of reset. The application software should check that all reset bits in the `PCIE_RSCK_STAT` register are cleared before accessing the local PCIe core and the APP registers. Initialize all relevant PCIe registers along with any address translation mapping before setting the `PCIE_APP_CTL.LTSSMEN` bit. Once asserted, the APP block and the LTSSM begin link training.

PCI configuration header space, PCIe standard and extended capability structures and port logic and application registers are accessed by a SOC fabric master over the DBI SCB port.

The PCIe specification requires components to enter the LTSSM detect state within 20 ms of the end of reset. The specification does not ask components on the same link to exit reset at the same time. The PCIe specification also notes that the only transition out of detect is polling. The LTSSM is kept in the detect state as long as the

`PCIE_APP_CTL.LTSSMEN` bit is low and transitions to polling when the `PCIE_APP_CTL.LTSSMEN` bit is set. Polling makes the PCIe link exit electrical idle and send TS1 ordered set.

If the PCIe core is configured as a RC, software must wait a minimum of 100 ms before sending a configuration request to the device immediately below that port. The PCIe specification allows downstream devices that receive a configuration request to respond with a configuration request retry status (CRS) completion status to terminate the request. The request stalls until the local initialization is completed and the local device is ready to communicate with the host.

BARs Disable Example

Five BAR registers are enabled by default. As a result of that the amount of memory space advertised by the EP may be too large. The following example shows how to disable all BARs except BAR0.

1. Verify that the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set (enables DBI write accesses to read only registers).
2. Clear the `PCIE_EP_BAR0_MASK[n].ENABLED`/`PCIE_EP_BAR1_MASK[n].ENABLED` bits (1 ≥ n ≥ 4) registers.

   The `PCIE_EP_BAR0_MASK[n].ENABLED`/`PCIE_EP_BAR1_MASK[n].ENABLED` bits are sticky and they do not need to be modified after a warm or hot reset event.

Using the iATU's address match mode, it is also possible to map multiple SCB slaves (for example L1, L2, SMMR, DMC0, DMC1) to a single BAR address range.

For example, to allocate various SCB slaves (L1, L2, SMMR, DMC0, DMC1) to BAR0 in the following way.

Table 29-14: iATU SCB Slave Address Mapping

iATU Region	SCB Slave	SCB Slave Start Address	Size Required	BAR0 address range
0	L1	0x380000	64K bytes	0x10000000-0x1000FFFF
1	L2	0x20080000	256K bytes	0x10010000-0x1004FFFF
2	SMMR	0x31024180	64K bytes	0x10050000-0x1005FFFF
3	DMC0	0x80000000	63M bytes	0x10200000-0x140FFFFF
4	DMC1	0xC0000000	63M bytes	0x14100000-0x17FFFFFF

The following example code can be used to configure the iATU to achieve the address mapping shown in the table.

```
//L1 iATU map
*pREG_PCIE0_IATU_VWPRT_0          =    0x80000000;
*pREG_PCIE0_IATU_LBADDR_INB_0     = 0x10000000;
*pREG_PCIE0_IATU_UBADDR_INB_0     = 0x0;
*pREG_PCIE0_IATU_LMTADDR_OUTB_0   = 0x1000FFFF;
*pREG_PCIE0_IATU_LTADDR_INB_0     =    0x380000|0x28800000;
*pREG_PCIE0_IATU_UTADDR_INB_0     =    0x0;
*pREG_PCIE0_IATU_CTL1_INB_0       =    0x0;
```

```
*pREG_PCIE0_IATU_CTL2_INB_0        =    0x80000000;

//L2 iATU map
*pREG_PCIE0_IATU_VWPRT_0           =    0x80000001;
*pREG_PCIE0_IATU_LBADDR_INB_0      = 0x10010000;
*pREG_PCIE0_IATU_UBADDR_INB_0      = 0x0;
*pREG_PCIE0_IATU_LMTADDR_OUTB_0    = 0x1004FFFF;
*pREG_PCIE0_IATU_LTADDR_INB_0      =    0x20080000;
*pREG_PCIE0_IATU_UTADDR_INB_0      =    0x0;
*pREG_PCIE0_IATU_CTL1_INB_0        =    0x0;
*pREG_PCIE0_IATU_CTL2_INB_0        =    0x80000000;

//SMMR iATU map
*pREG_PCIE0_IATU_VWPRT_0           =    0x80000002;
*pREG_PCIE0_IATU_LBADDR_INB_0      = 0x10050000;
*pREG_PCIE0_IATU_UBADDR_INB_0      = 0x0;
*pREG_PCIE0_IATU_LMTADDR_OUTB_0    = 0x1005FFFF;
*pREG_PCIE0_IATU_LTADDR_INB_0      =    0x31024180;
*pREG_PCIE0_IATU_UTADDR_INB_0      =    0x0;
*pREG_PCIE0_IATU_CTL1_INB_0        =    0x0;
*pREG_PCIE0_IATU_CTL2_INB_0        =    0x80000000;

//DMC0 iATU map
*pREG_PCIE0_IATU_VWPRT_0           =    0x80000003;
*pREG_PCIE0_IATU_LBADDR_INB_0      = 0x10200000;
*pREG_PCIE0_IATU_UBADDR_INB_0      = 0x0;
*pREG_PCIE0_IATU_LMTADDR_OUTB_0    = 0x140FFFFF;
*pREG_PCIE0_IATU_LTADDR_INB_0      =    0x80000000;
*pREG_PCIE0_IATU_UTADDR_INB_0      =    0x0;
*pREG_PCIE0_IATU_CTL1_INB_0        =    0x0;
*pREG_PCIE0_IATU_CTL2_INB_0        =    0x80000000;

//DMC1 iATU map
*pREG_PCIE0_IATU_VWPRT_0           =    0x80000004;
*pREG_PCIE0_IATU_LBADDR_INB_0      = 0x14100000;
*pREG_PCIE0_IATU_UBADDR_INB_0      = 0x0;
*pREG_PCIE0_IATU_LMTADDR_OUTB_0    = 0x17FFFFFF;
*pREG_PCIE0_IATU_LTADDR_INB_0      =    0xC0000000;
*pREG_PCIE0_IATU_UTADDR_INB_0      =    0x0;
*pREG_PCIE0_IATU_CTL1_INB_0        =    0x0;
*pREG_PCIE0_IATU_CTL2_INB_0        =    0x80000000;
```

Link Training and Establishment

The PCIe core implements the LTSSM function according to the *PCI Express Base 3.0 Specification*, revision 1.0.

Downstream Device Enumeration by Root Complex

Before issuing any request to the device across the PCIe link, the application software must initialize the iATU register to correctly translate the SCB read or write transactions into PCIe TLPs. All PCIe configuration registers may be accessed by an SOC fabric master over the DBI SCB.

When the PHY link is up and running the `PCIE_APP_STAT.LNKUP` bit is set.

The PCIe software:

- Scans the PCIe fabric to discover its topology and learn how the fabric is populated
- Programs the base and limit registers of switch ports to reflect the BAR range of the devices enumerated downstream
- Programs the BARs of endpoints

Host Software Writes to Bus Master Enable (BME), Memory Space Enable (MSE), and I/O Space Enable (ISE) Bits in PCI-Compatible Command Register

End Point

The EP application logic must not generate any MEM or I/O requests until the host software has set the `PCIE_EP_STATCMD_[n].BME` bit in the PCI-compatible command register. This bit is not checked before transmitting requests. Therefore the application software must monitor the status of the `PCIE_EP_STATCMD_[n].BME` bit by reading the register over the DBI SCB interface.

Root Complex

The RC application logic must not generate any MEM or I/O requests until the host has enabled the Memory Space Enable bit (`PCIE_RC_STATCMD_[n].MSE`) and the I/O Space Enable bits (`PCIE_RC_STATCMD_[n].IOE`) in the PCI-compatible command register. The RC port core does not check these bits before transmitting requests. Therefore the application can monitor the status of these bits by reading the register over the DBI SCB.

Configuration for Memory Read and Write Transactions

The following sections provides procedures for configuring memory reads and writes.

NOTE: Do not configure the IATU when inbound or outbound transactions are in progress. The ***Inbound Transactions*** and ***Outbound Transactions*** figures show the respective data transfers.

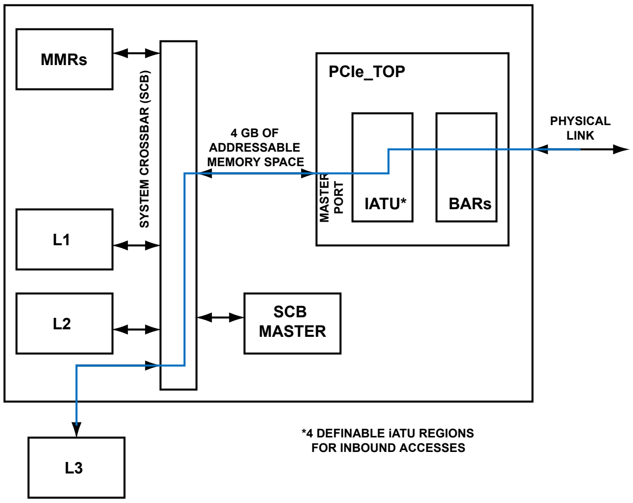

MMRs
SYSTEM CROSSBAR (SCB)
4 GB OF ADDRESSABLE MEMORY SPACE
PCIe_TOP
MASTER PORT
IATU*
BARs
PHYSICAL LINK
L1
L2
SCB MASTER
L3
*4 DEFINABLE iATU REGIONS FOR INBOUND ACCESSES

Figure 29-4: Inbound Transactions (PHYSICAL LINK to L3)

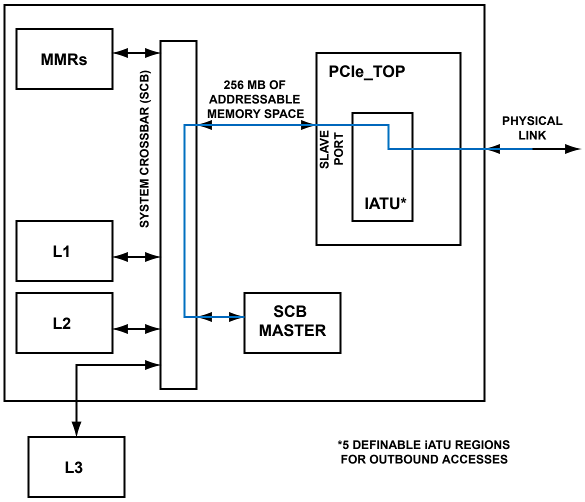

MMRs
SYSTEM CROSSBAR (SCB)
256 MB OF ADDRESSABLE MEMORY SPACE
PCIe_TOP
SLAVE PORT
IATU*
PHYSICAL LINK
L1
L2
SCB MASTER
L3
*5 DEFINABLE iATU REGIONS FOR OUTBOUND ACCESSES

Figure 29-5: Outbound Transactions (SCB MASTER to PHYSICAL LINK)

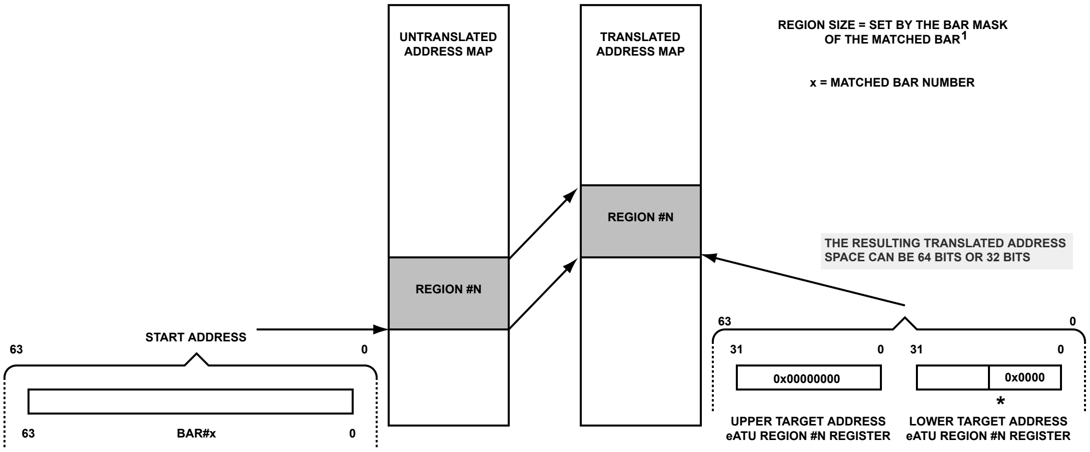
Inbound Programming Example (BAR Match Mode)

Define Inbound Region n (4 ≥ n ≥ 0) as the memory region matching BAR4 mapping to 0x8000 0000 in the processor's address space using the following equation. In BAR Match mode the region size is set by the BAR mask of the matched BAR.

Refer to IATU Register section and the ***iATU Address Region Mapping: Inbound 64-bit (BAR Match Mode)*** figure below. Set up the `PCIE_IATU_VWPRT_[n]` register to region n to access the IATU configuration registers of this region.

1. Configure the Index register to the set inbound region n as the current region.
2. Write n to the `PCIE_IATU_VWPRT_[n].REGNINDX` bit field and 1 to the `PCIE_IATU_VWPRT_[n].REGNDIR` bit.

3. Configure the target address registers.
4. Write 0x8000 0000 to the `PCIE_IATU_LTADDR_INB_[n]` register to set the lower target address.
5. Write 0x0000 0000 to the `PCIE_IATU_UTADDR_INB_[n]` register to set the upper target address.
6. Configure the region through the region control 1 inbound register.
7. Write 0x0000 0000 to the `PCIE_IATU_CTL1_INB_[n]` register to define the type of the region to be MEM.
8. Enable the region for BAR match mode.
9. Write 0xC000 0400 to the `PCIE_IATU_CTL2_INB_[n]` register to enable the region for BAR match for BAR#4.


UNTRANSLATED ADDRESS MAP
TRANSLATED ADDRESS MAP
REGION SIZE = SET BY THE BAR MASK OF THE MATCHED BAR¹
x = MATCHED BAR NUMBER
REGION #N
REGION #N
THE RESULTING TRANSLATED ADDRESS SPACE CAN BE 64 BITS OR 32 BITS
START ADDRESS
63
0
63
BAR#x
0
63
0
31
0
31
0
0x00000000
0x0000
*
UPPER TARGET ADDRESS eATU REGION #N REGISTER
LOWER TARGET ADDRESS eATU REGION #N REGISTER

* IS log2(CX_ATU_MIN_REGION_SIZE)

NOTE:
1. Setting the destination address for the BAR match is tied closely to the BAR mask. For example BAR0 has a BAR mask of 0xF0000000 meaning it can be assigned any address with bits [31:28] between 0x1 and 0xF and bits [27:0] always 0x0000000. The BAR match address translation mode also uses this mask. PCIe's target address should provide the correct [27:0] bits

Figure 29-6: iATU Address Region Mapping: Inbound 64-bit (BAR Match Mode)

Inbound Programming Example (Address Match Mode)

Define Inbound Region n (4 ≥ n ≥ 0) as MEM region matching TLPs with address in the range (see also the ***iATU Address Region Mapping: 64-bit Address Outbound and Inbound (Address Match Mode*** figure):

- From {PCIE_MSTARTH, PCIE_MSTARTHL} to {PCIE_MSTARTH, PCIE_MSTARTHL} + 0xFF_FFFF
- SOC MEM Region
- From 0x8000_0000 to 0x80FF_FFFF

Configure the `PCIE_IATU_VWPRT_[n]` register to region n to access the IATU configuration registers of this region.

1. Configure the `PCIE_IATU_VWPRT_[n].REGNINDX`Index Register to set inbound region n as the current region.
2. Write n to the `PCIE_IATU_VWPRT_[n].REGNINDX` bit field and write 1 to the `PCIE_IATU_VWPRT_[n].REGNDIR` bit.
3. Setup the region base and limit address registers.
4. Write PCIE_MSTARTL to the `PCIE_IATU_LBADDR_INB_[n]` register to set the lower base address.
5. Write PCIE_MSTARTH to the `PCIE_IATU_UBADDR_INB_[n]` register to set the upper base address.
6. Write 0xFF_FFFF to the `PCIE_IATU_LMTADDR_INB_[n]` register to set the limit address.
7. Setup the target address registers.
8. Write 0x8000_0000 to the `PCIE_IATU_LTADDR_INB_[n]` register to set the lower target address.
9. Write 0x0000_0000 `PCIE_IATU_LTADDR_INB_[n]` register to set the upper target address
10. Configure the region through the region control 1 register.
11. Write 0x0000_0000 to the `PCIE_IATU_CTL1_INB_[n]` register to define the type of the region to be MEM.
12. Enable the region.
13. Write 0x8000_0000 to the `PCIE_IATU_CTL1_INB_[n]` register to enable the region.

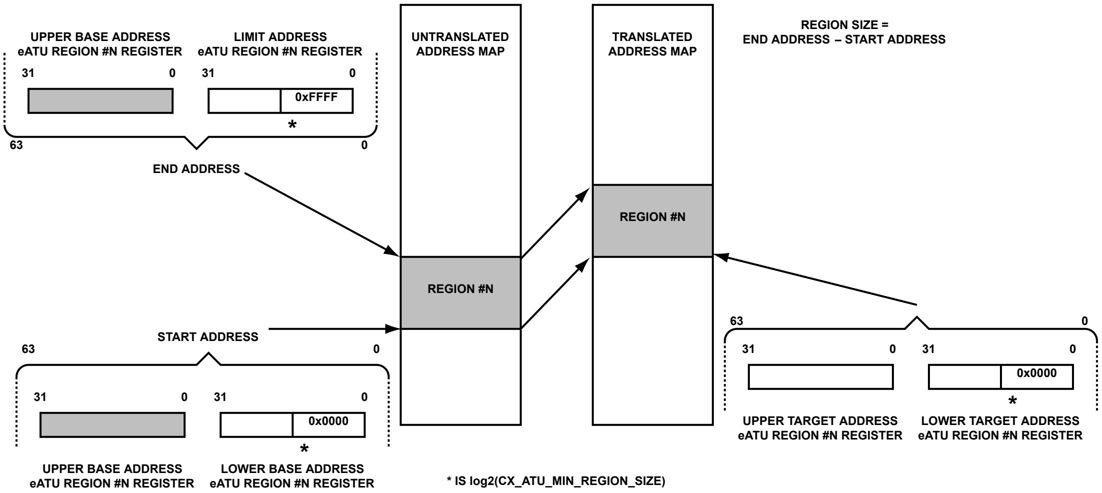

UPPER BASE ADDRESS
eATU REGION #N REGISTER
LIMIT ADDRESS
eATU REGION #N REGISTER
31
0
31
0
0xFFFF
*
63
0
END ADDRESS
UNTRANSLATED
ADDRESS MAP
TRANSLATED
ADDRESS MAP
REGION SIZE =
END ADDRESS – START ADDRESS
REGION #N
REGION #N
START ADDRESS
63
0
31
0
31
0
0x0000
*
UPPER BASE ADDRESS
eATU REGION #N REGISTER
LOWER BASE ADDRESS
eATU REGION #N REGISTER
63
0
31
0
31
0
0x0000
*
UPPER TARGET ADDRESS
eATU REGION #N REGISTER
LOWER TARGET ADDRESS
eATU REGION #N REGISTER
* IS log2(CX_ATU_MIN_REGION_SIZE)

Figure 29-7: iATU Address Region Mapping: 64-bit Address Outbound and Inbound (Address Match Mode)

Outbound Programming Example

Define Outbound Region n (3 ≥ n ≥ 0) as:

- SOC MEM Region

  - From MSTART (0x5000_0000 ≥ MSTART ≥ 0x5FFF_0000)
  - To MEND (0x5000_FFFF ≥ MSTART ≥ 0x5FFF_FFFF)
- Mapped to PCIe IO Space
  - From PCIE_MSTART
  - To PCIE_MEND

Refer to iATU Register section. Set up the port logic registers (`PCIE_IATU_VWPRT_[n]`) to region n to access the iATU configuration registers of this region.

1. Setup the Index Register to set outbound region n as the current region.
2. Write n to the `PCIE_IATU_VWPRT_[n].REGNINDX` bit field and write 1 to the `PCIE_IATU_VWPRT_[n].REGNDIR` bit.
3. Setup the region base and limit address registers.
4. Write MSTART to the `PCIE_IATU_LBADDR_OUTB_[n]` register to set the lower base address.
5. Write 0x0000 0000 to the `PCIE_IATU_LBADDR_OUTB_[n]` register to set the upper base address.
6. Write MEND to the `PCIE_IATU_LMTADDR_OUTB_[n]` register to set the limit address.
7. Setup the target address registers.
8. Write PCIE_MSTART to the `PCIE_IATU_LTADDR_OUTB_[n]` register to set the lower target address.
9. Write 0x0000 0000 to the `PCIE_IATU_UTADDR_OUTB_[n]` register to set the upper target address.
10. Configure the region through the region control 1 register.
11. Write 0x0000 0002 to the `PCIE_IATU_CTL1_OUTB_[n]` register to define the region type as I/O.
12. Enable the region.
13. Write 0x8000 0000 to the `PCIE_IATU_CTL2_OUTB_[n]` register to enable the region.

Non DMA Transfers

Non DMA transfers are transfers triggered by a local System Cross Bar fabric master (transmit) or by a remote partner (receive).

Transmit Transaction Layer Packets (TLP)

Data is transmitted to a PCIe link partner by writing to the slave system cross bar port. Data is read from a PCIe partner by reading from the slave SCB port. The slave system cross bar buses are directly connected to PCIe core's corresponding inputs and outputs. The iATU is used to map outbound system cross bar transactions from different system cross bar address regions to PCIe address regions and to determine their TLP types.

The address field of each request is checked to see if it falls into any of the enabled regions. When an address match is found, the TLP address field, TYPE, TLP digest field present (TD) and attributes (ATTR) header fields are replaced with the corresponding fields in the `PCIE_IATU_CTL1_OUTB_[n]` register.

Receive Transaction Layer Packets (TLP)

The writes addresses of the PCIe partners are translated into master SCB write transactions to the SOC address space. PCIe partners' reads from the processor's assigned address are translated into master system cross bar read transactions from SOC address space. The master system cross bar buses are directly connected to PCIe core corresponding inputs and outputs.

Inbound address translation operates in one of two modes as determined by the `PCIE_IATU_CTL2_INB_[n].MAT` bit field.

- Address Match Mode. Normally used by a RC, but may be useful in mapping sections of an EP memory region into noncontiguous sections. The address field of each request TLP is checked to see if it falls into any of the enabled address regions. When an address match is found the TLP address is modified.
- BAR Match Mode. Normally used by an EP. Looking for an address match is a two step process.
    1. The standard internal PCI Express BAR matching mechanism checks if the address field of any MEM and IO request TLP falls into any address region defined by the enabled BAR address and masks.
    2. When a matched BAR is found, the iATU compares the BAR ID to the `PCIE_IATU_CTL2_INB_[n].BARNUM` bit field for all enabled regions.

The `PCIE_APP_STAT.RXQNEMPTY` bit signals that the receive queues are not empty.

DMA Transfers

These are transfers triggered by the DMA write channel (local to remote memory) or by the DMA read channel (remote to local memory).

The presence of the DMA controller does not affect:

- Normal filtering rules for inbound TLPs (for example, BAR checking in end points)
- The operation of any internal or external address translation. To prevent the iATU from translating inbound or outbound requests that are generated by the internal DMA module one of the following approaches can be implemented:
    - Ensure that the combination of DMA channel programming and iATU control register programming, causes no translation of DMA traffic to occur in the iATU.
    - Activate the DMA bypass mode to allow request TLPs which are initiated by the DMA controller to pass through the iATU untranslated.

Initialize all registers except `PCIE_DMA_VWPRT_SEL_[n]`. Initialize reserved bits to 0x0.

Read Transfer

The DMA issues a read request to the PCIe link (1 and 2 in the ***DMA Read Transfers*** figure). When the read data is received (3 in the figure), the DMA issues a write request to the local memory (4 and 5 in the figure). Once the write completes, the local memory issues a write response to the PCIe core (6 in the figure).

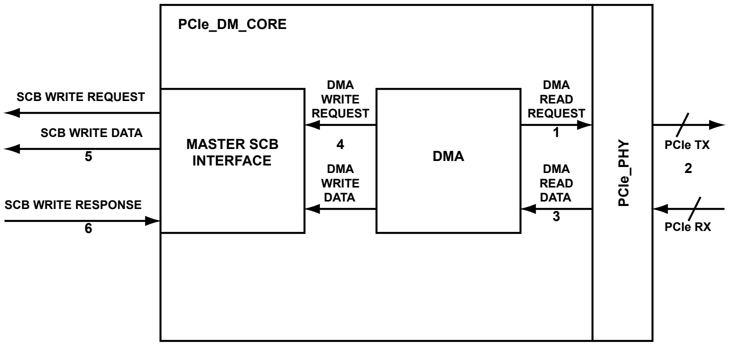


Figure 29-8: DMA Read Transfers

To access the read channel registers, set the `PCIE_DMA_VWPRT_SEL_[n].DIR` bit and clear the `PCIE_DMA_VWPRT_SEL_[n].NUM` bit field.

The address of the remote buffer is programmed using the `PCIE_DMARD_SAR_LO_[n]` register and the address of the local buffer is programmed using the `PCIE_DMARD_DAR_LO_[n]` register.

The DMA transfer size is programmed using the `PCIE_DMARD_XFRSZ_[n]` register. The DMA decrements the value in this register as the DMA transfer progresses. This register can be read at any time to determine the number of bytes remaining to be transferred. When all bytes are successfully transferred, the result is zero. The DMA transfer size can be of any value from one byte, up to a maximum of 4 GB.

Program bits [31:12] in the `PCIE_DMARD_CTL1_[n]` register to set the TC, TD, RO, NS, AT and FN TLP header fields for any TLP it generates. Once all the DMA controller registers are programmed, a DMA read transfer is started by writing 0x0 to the `PCIE_DMARD_DRBL_[n].NUM` bit field.

The normal end of a DMA read transfer is detected by any of the following methods:

- The assertion of the RSCKPHY_EDMA_INT interrupt and setting the `PCIE_DMARD_IMSK_[n].DONE` bit.
- The reception of a Done remote interrupt
- When the CS (Channel Status) field in the `PCIE_DMARD_CTL1_[n].CS` is stopped, and the `PCIE_DMARD_XFRSZ_[n]` register = 0x0.
- Polling the `PCIE_DMARD_ISTAT_[n]` register.

The abnormal end of a DMA read transfer is detected by any of the following methods:

- The assertion of a RSCKPHY_EDMA_INT interrupt and setting the `PCIE_DMARD_ISTAT_[n].ABRT` bit.

- The reception of an abort remote interrupt
- When the `PCIE_DMARD_CTL1_[n].CS` (channel status) field halts. The `PCIE_DMARD_XFRSZ_[n]` register indicates the remaining number bytes to be transferred except when there is a system cross bar write error.
- Polling the `PCIE_DMARD_ISTAT_[n]` register.

The application software can stop DMA read transfers by writing to the `PCIE_DMARD_DRBL_[n].STOP` bit. This action also sets the `PCIE_DMARD_ISTAT_[n].ABRT` bit.

Write Transfer

The DMA issues a read request to the local memory (1 and 2 in the ***DMA Write Transfers*** figure). When the read data is received (3 and 4 in the figure), the DMA sends the data to the PCIe link (5 and 6 in the figure).

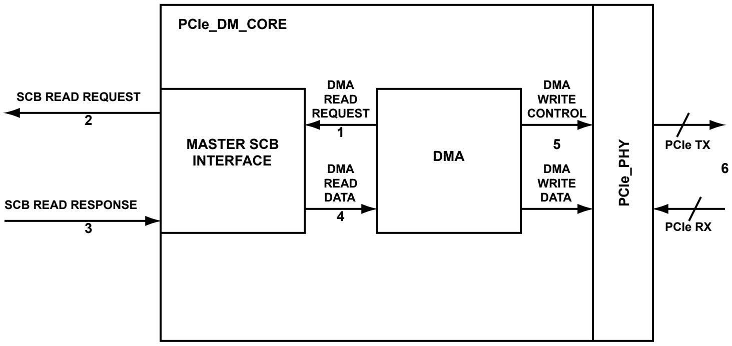


Figure 29-9: DMA Write Transfers

To access the write channel registers, clear the `PCIE_DMA_VWPRT_SEL_[n].DIR` bit and the `PCIE_DMA_VWPRT_SEL_[n].NUM` bit field.

The address of the local buffer is programmed using the DMASAR_WR_LOn register and the address of the remote buffer is programmed using the `PCIE_DMAWR_DAR_LO_[n]` register.

The DMA transfer size is programmed using the `PCIE_DMAWR_XFRSZ_0[n]` register. The DMA decrements the value in this register as the DMA transfer progresses. This register can be read at any time to determine the number of bytes remaining to be transferred. When all bytes are successfully transferred, the result is zero. The DMA transfer size can be of any value from one byte, up to a maximum of 4 GB. Program bits [31:12] in the `PCIE_DMAWR_CTL1_[n]` register to set the TC, TD, RO, NS, AT and FN TLP header fields for any TLP it generates.

Once all the DMA controller registers are programmed, a DMA write transfer is started by writing 0x0 to the `PCIE_DMAWR_DRBL_[n].NUM` bit field.

The normal end of a DMA write transfer is detected by any of the following methods:

- The assertion of RSCKPHY_EDMA_INT interrupt and setting the `PCIE_DMAWR_IMSK_[n].DONE` bit.
- The reception of a DONE remote interrupt.

- When the `PCIE_DMAWR_CTL1_[n].CS` bit field (Channel Status) is stopped and the `PCIE_DMAWR_XFRSZ_0[n]` register =0x0
- Polling the `PCIE_DMAWR_ISTAT_[n]` register.

The abnormal end of a DMA write transfer is detected by any of the following methods:

- The RSCKPHY_EDMA_INT interrupt is asserted and the`PCIE_DMAWR_IMSK_[n].ABRT` bit is set.
- The reception of a ABORT remote interrupt.
- When the `PCIE_DMAWR_CTL1_[n].CS` bit field halts. The `PCIE_DMAWR_XFRSZ_0[n]` register indicates the remaining number of bytes to be transferred.
- Polling the `PCIE_DMAWR_ISTAT_[n]` register.

The application software can stop DMA write transfers by writing to the `PCIE_DMARD_DRBL_[n].STOP` bit. This action also sets the `PCIE_DMAWR_ISTAT_[n].ABRT` bit.

Flow Control

The flow control mechanism is divided into two phases: initialization and update. The PCIe core automatically performs both of these phases with minimal support required from the application.

EP to EP Transactions

An EP does not know the actual address of other EPs in the PCIe system. To enable transfers between EPs or between an EP and other memory-mapped components, the iATU needs to be programmed. To do this the PCIe upstream device can access the iATU directly over the PCIe link or can “ask” an EP local master to do so.

Gen2 5.0 GT/s Operation

The PCIe core supports all non-optional Gen2 5.0 GT/s features defined in the PCI Express Base 3.0 Specification, revision 1.0. The interface between the core and the PHY is compliant with the PIPE Specification for PCI Express, Version 4.0. The number of active symbols on the PIPE interface is constant and the frequency of the core doubles each time as the core transitions from the Gen1 to the Gen2 rate.

Power Management

The power management (PM) state of a PCIe link is determined by the D-state of the upstream port (USP, PCIe core configured as an EP). Power is managed in two ways, by software running on the host or locally by the Active State Power Management (ASPM) hardware.

Power Management Software

The Power Management Control and Status registers should be set to the desired default values, the NO_SOFT_RST should be set as software Reset is not supported. The PM software running on the upstream device can direct the PCIe core to enter any of the D1 or D3 low-power states. It does this by writing to the Power Management Control and Status Register (PMCSR) in the PCI-PM capability structure. Link states L0 and L1 are not visible to the PM software and are derived from PM D-state connected to the link. The action of changing the D-state in the PMCSR indirectly causes the USP to change the link power state. The L1 state is entered when the USP on a link is programmed to a D-state other than D0.

The following table shows the relationship between Device State (Dx), Link State (Sx) and PHY state (Px) and summarizes the state of clock and power sources for the supported Power States.

Table 29-15: Device State (Dx), Link State (Sx) and PHY state (Px)

D-State	Link State Change Trigger[*1]	Permissible Quiescent[*2] Link States	Active Power Supplies[*3]	Active Clock Supplies[*4]	PIPE_CLK CORE_CLK	PHY State
D0$_{\text{unitialized}}$	n/a	L0	VDDINT	PCIeCLK SYSCLK	ON	P0, P0S
D0$_{\text{active}}$	ASPM[*5]	L0				
		L0S				
		L1				P1
D1, D3$_{\text{hot}}$	PCI-PM[*6]	L1				

*1 The ***Port Type Link States*** table shows what link state each port type can trigger.

*2 V_{DDINT} is connected to the PCIe core V_{AUX} and V_{MAIN}. V_{AUX} is the power supply for the Power Management Controller (PMC) and Configuration Dependent Module (CDM) modules. V_{MAIN} is the power supply for the rest of the core.

*3 V_{DDINT} is connected to the PCIe core V_{AUX} and V_{MAIN}. V_{AUX} is the power supply for the Power Management Controller (PMC) and Configuration Dependent Module (CDM) modules. V_{MAIN} is the power supply for the rest of the core.

*4 The PCIeCLK or REFCLK is connected to the PCIe PHY which is the platform reference clock (REFCLK) for the PHY TX PLL. The platform low-power clock (CLK_RST's AUXCLK input) is derived from *SYSCLK*.

*5 ASPM is the Active State Power Management (Autonomous Link Power Management) hardware.

*6 By executing a PMCSR write. PMCSR is the power management control and status register (`PCIE_PMT_CTLSTAT_[n]`) in the PCI Compatible Power Management structure (PCI-PM).

Table 29-16: Port Type Link States

Port	Triggers L0/L0s Entry	Triggers L1 Entry
USP	Yes	Yes, on any of these triggers: Software programs non-D0 state (PCI-PM) USP's idle timer expires (ASPM)
DSP	No	Indirectly by writing non-D0 value to PMCSR of the USP link partner.

If the USP device needs to transition back to D0 it can send a PME message to the host. The PM software can put the USP into D0 by writing to the Power Management Control and Status Register (`PCIE_PMT_CTLSTAT_[n]`) in the PCI-PM capability structure.

Active State Power Management (ASPM)

ASPM is enabled through the ASPM control field in the link control register (`PCIE_LNK_CAPB_[n].ASPM`). When the USP is in L0 and detects idle on the link for a specific amount of time (determined by the `PCIE_ACK_ASPMCTL_[n].FREQ` and `PCIE_LNK_CAPB_[n].L0SELAT` L0s entrance latency field2 in the (Ack Frequency and L0-L1 ASPM Control Register), it automatically transitions the link to the L0 or L1 power state. Each link direction (transmit and receive) is processed separately. The core exits L0s when there is traffic waiting to be sent.

Completion Timeout Ranges

Timeout ranges are supported as defined in the PCI Express Base 3.0 Specification, revision 1.0.

PHY Registers Access

PHY registers can be accessed in the following ways:

- As memory-mapped registers across the ADI_RCCKPHY and PHY parallel interface
- Across the JTAG port

RCCKPHY

See Reset Clock and PHY Control Module (RSCKPHY).

Event Control

The following sections provide information about PCIe errors, messages, and interrupts.

Error Handling and Debug

The following sections provide information on errors and debugging the PCIe module.

System Cross Bar Master Interface to PCIe Link Error Mapping

PCIe masters can access the ADSP-SC58x address space across the master system cross bar interface. The system cross bar slave responds to posted or non-posted requests issuing write or read response transactions.

Memory read transactions are non-posted and require the PCIe port to create a completion packet, including error information if needed.

Error Mapping:

- System cross bar SLVERR Slave Error, means that the slave wishes to return an error condition to the master is mapped to a CA (Completer Abort) error.
- System cross bar DECERR Decode Error, means that there is no slave at the transaction address is mapped to an UR (Unsupported Request) error.

Memory writes are posted transactions and do not require a completion packet to be generated. The application can send a message to the root to inform the PCIe software about the error, but the error is not seen by the requester. The system cross bar master interface does not forward information about the requester to the target of the write transaction, the write ID is always 0. The write response is dropped by the system cross bar bridge and not forwarded to the PCIe link.

PCIe Link Error Mapping to SCB Slave Interface

PCIe slaves can access PCIe's address space across the slave system cross bar interface. All accesses are treated as memory ones. System cross bar slave accesses can be used to create different type of packets if the iATU is enabled and configured as needed. The PCIe slave responds to non-posted requests (read requests) sending completion packets. Completion packets including error information, if needed.

Error mapping:

- PCIe completion time-out is mapped to system cross bar DECERR (decode error).
- ECRC error is mapped to system cross bar SLVERR (slave error).
- Poisoned error (data in TLP is known to be corrupted) is mapped to system cross bar SLVERR
- CA (completer abort) error is mapped to system cross bar SLVERR
- CRS (configuration request retry status) error is mapped to system cross bar SLVERR
- UR (unsupported request) is mapped to system cross bar DECERR

Completion packets are not generated for PCIe posted transactions, therefore responses to slave memory write transactions are generated locally.

Messages

Messages are generated from memory write transactions using the Internal Address Translation Unit. To ensure that the iATU does not translate MSI or power management messages, the third and fourth double words of these messages do not match any programmed iATU address region.

When there is a successful address match on an outbound MEM TLP, and the translated TLP type field is MSG, then the message code field of the TLP is set to the value in the *Message Code* field of the `PCIE_IATU_CTL2_OUTB_[n]` register. A MWr (memory write transaction) with an effective length of 0" (taking into account SLV_AWSTRB) is converted to Msg and all other MWr TLPs are converted to MsgD.

PCIe Interrupts

The following sections provide information on interrupts associated with the PCIe module.

PCIe Interrupt Request

The following interrupts can trigger a PCIe interrupt request:

- Message signaled interrupt

- Legacy interrupts
- Incoming message interrupts
- Link up
- Receive overflow

These interrupts can be individually enabled by the application software, as needed. To determine the source of the interrupt, read the `PCIE_APP_INTSTAT` register and write a 1 to the interrupt status bit to clear it.

Configuration

To enable end points to generate MSIs, MSI capabilities register `PCIE_MSI_CAPB_NPTR_[n]`,`PCIE_MSI_CAPB_OFF4_[n]`, `PCIE_MSI_CAPB_OFF8_[n]` and `PCIE_MSI_CAPB_OFFC_[n]` (See section 10) need to be programmed.

The local core or the PCIe host configures the MSI interrupt controller.

- The local core or the PCIe host programs the common MSI address used for the EPs into the `PCIE_MSI_CTL_LADDR_[n]` and `PCIE_MSI_CTL_UADDR_[n]` registers.
- The PCIe host reads the MSI capability of each EP to determine the number of vectors enabled in each EP and use this information to program the interrupt enable registers in the MSI interrupt controller.
- The local core or the PCIe host programs all interrupt enable registers (for example `PCIE_MSI_IEN0_[n]`) and all interrupt mask registers (for example `PCIE_MSI_IMSK0_[n]`).

MSI Generation by the EP Local Core

The local EP core provides the following.

- Verify that the `PCIE_APP_STAT.OUTSTMSI` bit is cleared.
- Write 1 to the `PCIE_APP_CTL.INTREQ` bit.
- If the PCIe core is multiple message capable, set the `PCIE_APP_CTL.MSIVECT` bit field to identify the MSI source.
- Poll the `PCIE_APP_STAT.MSIGRANT` bit to check if the MSI request has been granted.
- Clear the `PCIE_APP_STAT.MSIGRANT` bit before issuing a new MSI request.

MSI Reception by the RC port

MSI packets are captured and terminated in the system cross bar bridge, the packet's payload is decoded, the corresponding interrupt status register (for example `PCIE_MSI_ISTAT0_[n]`) is updated and an interrupt is signaled asserting the PCIe interrupt request. The interrupt routine running on the local RC core should:

- Read the interrupt status registers if MSIs from EPn are enabled.
- Write to the interrupt status registers to clear the bit that triggered the interrupt.

Configuration

Only INTA can be generated. To enable end points to generate INTA, disable MSIs `PCIE_MSI_CAPB_NPTR_[n]` bit 16 =0). The local core or any other system cross bar master in the RC enables each specific interrupt.

- The local core or any other system cross bar master sets the `PCIE_APP_CTL.AIEN`, `PCIE_APP_CTL.BIEN`, `PCIE_APP_CTL.CIEN` or `PCIE_APP_CTL.DIEN` bits.

Interrupt Assert Generation by the EP Local Core (INTA only)

The local EP core should:

- Verify that the `PCIE_APP_STAT.OUTSTMSI` bit is cleared
- Write 1 to the `PCIE_APP_CTL.INTREQ` bit

Interrupt De-assert Generation by the EP Local Core (INTA only)

The application software or any system cross bar master can trigger an INTA deassert message and clear the `PCIE_APP_STAT.OUTSTINTA` bit. It should:

- Write 1 to the `PCIE_APP_CTL.INTACLR` bit. Two cases can be considered:
  - The device that receives the interrupt understands a sequence of assert INTA message and deassert INTA message as an interrupt pulse that has to be serviced if enabled.
  - The device that receives the interrupt needs to acknowledge its reception to the interrupt source to clear it. In this case the interrupt routine on the PCIe Host can either write to the EP's `PCIE_APP_CTL` register or send a message (write) to the EP to acknowledge the reception of the interrupt.

Writing to the `PCIE_APP_CTL.INTACLR` bit when the `PCIE_APP_STAT.MSIEN` bit is set has no effect.

Interrupt Reception by the RC Port (INTA, INTB, INTC or INTD)

The reception of an enabled interrupt sets the corresponding bit in the `PCIE_APP_INTSTAT` register. These interrupts are treated as level. The interrupt service routine should:

- Test the `PCIE_APP_INTSTAT.INTA`, `PCIE_APP_INTSTAT.INTB`, `PCIE_APP_INTSTAT.INTC` and `PCIE_APP_INTSTAT.INTD` bits to determine which interrupt to service.
- If necessary, write to the EP to acknowledge that the interrupt is being serviced. The interrupt service routine can only clear the above interrupt bits by writing to EP's APP_CTL register.
- Wait for the INTiASSTD to clear to exit.

Incoming Messages Interrupts

The following Inbound messages trigger a PCIe interrupt:

- Vendor Defined
- Unlock Transaction (RC Mode)
- Error Signaling (EP Mode)
    - Correctable
    - Non-Fatal
    - Fatal
- Power Management
    - Event (RC Mode)
    - Turn Off Acknowledge (RC Mode)
    - Turn Off (EP Mode)

Configuration

Enabling interrupts that cannot be received in EP or RC modes is meaningless.

The local core or any other system cross bar master in the RC enables each specific interrupt.

- Write 1 to the appropriate interrupt enable bit in the `PCIE_APP_CTL` register.

Interrupt Reception

Applies to the reception of an enabled interrupt. The corresponding bit in the `PCIE_APP_INTSTAT` register is set. The interrupt routine should:

- Test the `PCIE_APP_INTSTAT.CORERR`, `PCIE_APP_INTSTAT.FTLERR`, `PCIE_APP_INTSTAT.NFTLERR`, `PCIE_APP_INTSTAT.PME` `PCIE_APP_INTSTAT.PMTOACK`, `PCIE_APP_INTSTAT.PMTOFF`, `PCIE_APP_INTSTAT.UNLK`, and `PCIE_APP_INTSTAT.VENDMSG` bits to determine which interrupt to serve.
- Read the corresponding APP_<message name>_MSG registers (xxxMSGRQID) to discover the source of the xxx message.
- If a Vendor message is received, reading the `PCIE_APP_VNDR_MSG_HDR1` and `PCIE_APP_VNDR_MSG_HDR0` registers provide
- Once the interrupt is serviced, clear the status bit by writing 1 to the corresponding bit in the `PCIE_APP_INTSTAT` register.

Link Up- and Receive-Overflow Interrupts

These interrupts can be enabled in both EP as well as RC modes of operation.

Configuration

The Local core or any other system cross bar master in the RC has to enable the Receive Overflow interrupt. The Link Up interrupt is enabled by default but can be disabled by software.

- Write 1 to the `PCIE_APP_CTL.RXQOVFIEN` bits to enable the Receive Overflow interrupt.
- Write 1 to the `PCIE_APP_CTL.LNKUPINTDIS` bits to disable the Link Up interrupt.

The `PCIE_APP_STAT.LNKUP` bit reflects the status of the Links Data and MAC/PHY layers.

Interrupt Reception

Applies to the reception of an enabled interrupt. The corresponding bit in the `PCIE_APP_INTSTAT` register is set. The interrupt routine should:

- Test the `PCIE_APP_INTSTAT.LNKUP` and `PCIE_APP_INTSTAT.RXQOVF` bits to determine which interrupt to serve.
- Once the interrupt is serviced, clear the status bit by writing 1 to the corresponding bit in the `PCIE_APP_INTSTAT` register.

End Point

If the PCIe core is configured as an EP, the application software can send the upstream device either MSI (Message Signaled Interrupts) or virtual legacy INTA messages.

MSI Generation

See section 4.5.3.2.1 Interrupts in EP mode.

INTA Assert and INTA Deassert Generation

See section 4.5.3.2.1 Interrupts in EP mode.

The application software needs to deassert the virtual interrupt when the system software has disabled interrupts or if the PCI Express link has been placed in a low power state. The PCIe core does automatically send a deassert interrupt message when software disables interrupts. The application software must deassert the virtual interrupt before it can send a new interrupt because the PCIe core requires a rising edge on the virtual interrupt signal to generate a new Assert interrupt message. The core does not automatically send a deassert interrupt message when the power state changes.

Root Complex

If the PCIe core is configured as an RC, it can receive either MSI or Legacy Interrupt messages but cannot generate them. These messages make the RSCKPHY_PCIEINT_REQ interrupt signal to become active. If the MSI controller is disabled, MSIs do not make RSCKPHY_PCIEINT_REQ interrupt signal assert, instead MSI are forwarded as Write Transactions to the SCB master port. See section 4.5.3.2.2. for more information.

MSI Reception in the SCB Bridge

When an inbound (received) MWr request is passed to the peripheral or system cross bar bridge and matches the specified MSI address, as well as the conditions specified for an MSI memory write request, then an MSI interrupt is detected and RSCKPHY_PCIEINT_REQ interrupt signal asserted. The MSI write request never appears on the SCB master bus.

Programming and Using the MSI Controller in the SCB Bridge

The host CPU configures the MSI capabilities of all endpoints through CFG requests across the link or accessing the DBI address space.

- Program the "MSI Data Register" `PCIE_MSI_CAPB_OFF8_[n]` (which is part of the PCI Express MSI capability register structure of each EP) as follows to allow the MSI interrupt controller to decode the interrupt source.
- Program the "MSI Lower 32 Bits Address Register" `PCIE_MSI_CAPB_OFF4_[n]` (which is part of the PCI Express MSI capability register structure) of every EP with one common MSI address.

Table 29-17: MSI Data Register Programming

15:8	7:5	4:0
Not Used	EP Number Allows each EP to be identified within the system, For example, EP#5 is programmed with 3'b101	Interrupt Vector Number • Identifies the interrupt source within each EP • Programmed to 5'b00000. • Set by MSI generation logic to identify each interrupt source in real time • Supports up to 32 Vectors

The host CPU configures the MSI interrupt controller through CFG requests across the link or accessing the DBI address space.

- Program the common MSI address that you used for the EPs into the interrupt controller's MSI Controller Address register (`PCIE_MSI_CTL_LADDR_[n]`)
- Read the MSI capability of each EP to determine the number of vectors enabled in each EP and uses this information to program the interrupt enable registers in the MSI interrupt controller. The interrupt enable register allows up to 32 MSI interrupt vectors to be enabled within the MSI interrupt controller for a given EP. It is the responsibility of the host CPU to read the contents of the "Multiple Message Enable" field in an EP's MSI capability structure and program that EP's Interrupt Enable register in the interrupt controller appropriately. For example when the "Multiple Message Enable" is 0b100 for endpoint N (which corresponds to 16 enabled interrupt vectors), then the host CPU should program interrupt enable register #N in the interrupt controller with 0x0000FFFF.

Legacy Interrupts Reception

The PCIe core can receive INTA, INTB, INTC and INTD interrupt messages. Each interrupt can be individually enabled by setting the corresponding bit in the `PCIE_APP_CTL` register. If any of the enabled interrupts is detected, the corresponding bit in the `PCIE_APP_INTSTAT` register is set and RSCKPHY_PCIEINT_REQ asserted.

Local Interrupts

In addition to the PCIe Interrupts that are always generated by EP ports and received by RC ports, messages, and local events can also make RSCKPHY_PCIEINT_REQ assert. These interrupts may be generated in any of the PCIe core configurations.

Inbound Messages

The following Inbound Messages are registered:

- Vendor Defined
- Unlock Transaction (RC Mode)
- Error Signaling (EP Mode)
  - Correctable
  - Non-Fatal
  - Fatal
- Power Management
  - Event (RC Mode)
  - Turn Off Acknowledge (RC Mode)
  - Turn Off (EP Mode)

The information related to vendor messages is registered in a set of three registers:

- (If applicable) `PCIE_APP_VNDR_MSG_HDR0` message request header bytes 8 to 11
- (If applicable) `PCIE_APP_VNDR_MSG_HDR1` message request header bytes 12 to 15
- `PCIE_APP_VNDR_MSG` message request header requester ID field (bytes 4 and 5)

All other messages have only an APP_*Message Name*_MSG register. Each register holds the coresponding message request header ID field The message header information and requester ID are registered if the corresponding interrupt bit in the `PCIE_APP_INTSTAT` is not set.

Link Up

This Interrupt is triggered when the `PCIE_APP_STAT.LNKUP` bit transitions from 0 to 1. It tells the system that the link is up and ready to transfer data or messages. The `PCIE_APP_STAT.LNKUP` bit is read only. When the Link Up interrupt is asserted the `PCIE_APP_INTSTAT.LNKUP` bit is set.

Receive Overflow Interrupt

If an overflow occurs in any of the receive queues, the `PCIE_APP_INTSTAT.RXQOVF` bit is set and RSCKPHY_PCIEINT_REQ is asserted. This bit stays set and the interrupt asserted until the application software writes 1 to it.

Reset Interrupt Request

The Reset Interrupt Request is triggered only by the PCIe core warm or hot link reset request. This interrupt has to be enabled if the PCIe port is enabled. A warm or hot reset event sets the `PCIE_RSCK_STAT.WRMHTRST` bit.

In response to the RSCKPHY_PCIERST_REQ the interrupt service routine performs the following tasks.

- Allows the PCIe master accesses to complete normally.
- Stops all system cross bar bus masters from accessing PCIe core slave system cross bar port.
- Allows the slave system cross bar transactions in progress complete. The response is bus error.
- Stops the system cross bar bus masters from accessing the PCIe core DBI SCB port.
- Allows the slave DBI SCB transactions in progress to complete. The response is bus error.
- Two actions may follow.
  - Write 1 to the `PCIE_RSCK_CTL.DBISLVDIS` bit to let CLK_RST reset the PCIe core, non sticky registers and PCIe PHY.
  - Set the `PCIE_RSCK_CTL.PCIERES` bit to reset the whole PCIe port (similar to cold reset).
- If set, clear the `PCIE_RSCK_CTL.PCIERES` bit.
- Write 1 to the `PCIE_RSCK_STAT.WRMHTRST` bit to clear the interrupt request.

The core reset, PHY reset system cross bar master interface, system cross bar slave interface and system cross bar DBI interface reset status can be monitored by software by checking the `PCIE_RSCK_STAT.CORERST`, `PCIE_RSCK_STAT.PHYRST`, `PCIE_RSCK_STAT.MSTRST`, `PCIE_RSCK_STAT.SLVRST` and `PCIE_RSCK_STAT.DBIRST` bits. These bits are implemented for debug purposes and should be treated as asynchronous.

DMA Interrupts

RSCKPHY_EDMAINT_REQ is an interrupt request signal dedicated to the Embedded DMA controller.

Programming Model

The following sections provide general programming information for the PCIe module.

Enumeration

Enumeration is the first step in the system discovery and configuration process run by the PCIe CPU.

Enumeration Driver Example

The processor's PCIe port does not check configuration requests (at the application transmit interface) for valid target Bus-Device-Function (BDF) numbers before it transmits them. It does not check that the *target bus number* > *secondary bus number* and that *target bus number* ≤ *subordinate bus number*. Software running on the local core provides only legal requests to the PCIe core. Only one PCIe port is implemented, therefore bus 0 has only a single device on it. This device may be an EP or a switch. If the device is an EP the number of functions (F) is determined and the only valid BDF numbers are 00F. If the device is a switch the search on the secondary bus and down continues normally until the last EP is discovered.

The following ***CfgRd0 or CfgRd1 Example*** example requires the use and configuration of the iATU (See section 4.6.4. Internal Address Translation). Enumeration uses configuration requests, therefore the iATU has to be configured to translate slave system cross bar reads into CfgRd0 or CfgRd1 and slave system cross bar writes into CfgWr0 or CfgWr1 outbound transactions. To determine what Device 0 on bus 0 is, the RC can issue a CfgRd1 request from the device's 0 Vendor ID register. If the response is 0xFFFF (unsupported request), device 0 is an End Point and the PCIe software continues issuing CfgRd0 to determine the number of functions. If the response is different than 0xFFFF, device 0 is a bridge and the Enumeration process continues using configuration requests as required by the PCIe system topology.

Define Outbound Region n (3 ≥ n ≥ 0) as:

- MSIZE = 64 Kbytes
- SOC MEM Region
  - From MSTART (0x5000_0000 ≥ MSTART ≥ 0x5FFF_0000)
  - To MEND (0x5000_FFFF ≥ MSTART ≥ 0x5FFF_FFFF)
- Mapped to PCIe Configuration Space
  - CFG Space[31:16] = BDF
  - CFG Space[15:12] = 0x0
  - CFG Space[11:8] = Extended Register Number
  - CFG Space[7:2] = Configuration Register Number
  - CFG Space[1:0] = 0x0

Define Outbound Transaction as Cfg

- TYPE = CfgRd0 or CfgRd1
- TC = 0x0
- TD = 0
- ATTR = 0x0
- AT = 0x0

Port Logic Registers. Configure the `PCIE_IATU_VWPRT_[n]` register to access the region n IATU configuration registers.

1. Setup the Index Register to set outbound region n as the current region.
   a. Write n to `PCIE_IATU_VWPRT_[n].REGNINDX` and 0 to `PCIE_IATU_VWPRT_[n].REGNDIR` register address.
2. Setup the Region Base and Limit Address registers.
   a. Write “MSTART” to the `PCIE_IATU_LBADDR_OUTB_[n]` register to set the lower base address.
   b. Write 0x0000_0000 to the `PCIE_IATU_UBADDR_OUTB_[n]` register to set the upper base address.
   c. Write “MEND” to `PCIE_IATU_LMTADDR_OUTB_[n]` register to set the limit address.
3. Setup the Target Address Registers.
   a. Write “CFG Space” to the `PCIE_IATU_LTADDR_OUTB_[n]` register to set the lower target address.
   b. Write 0x0000_0000 to the `PCIE_IATU_UTADDR_OUTB_[n]` register to set the upper target address.
4. Configure the region through the Region Control 1 register.
   a. Write to the `PCIE_IATU_CTL1_OUTB_[n].TYPE`, `PCIE_IATU_CTL1_OUTB_[n].TC`, `PCIE_IATU_CTL1_OUTB_[n].TD`, `PCIE_IATU_CTL1_OUTB_[n].ATTR` and `PCIE_IATU_CTL1_OUTB_[n].AT` bits and 0 to all the other bits.
5. Enable the region.
   a. Write 0x8000_0000 to the `PCIE_IATU_CTL2_OUTB_[n]` register to enable the region.

Configuration write transactions are implemented in the same way as configuration read transactions. Configure the `PCIE_IATU_CTL1_OUTB_[n].TYPE` bit field to CfgWr0 or CfgWr1.

DMA Transfers

The order in which the DMA Controller’s registers are programmed is not important except for the `PCIE_DMAWR_DRBL_[n]` or `PCIE_DMARD_DRBL_[n]` registers, which initiates the DMA transfer.

DMA Write Transfer Example

Transfer 1MB from local L2 (0x2400 0000) to PCIe memory space (0x5000 0000)

1. Write 0x0 to the `PCIE_DMA_VWPRT_SEL_[n]` register to select channel 0 write registers.
2. Write 0x1 to the `PCIE_DMAWR_EN_[n].EN` bit.
3. Write 0x0 to the `PCIE_DMAWR_IMSK_[n]` register.

4. Clear the PCIE_DMAWR_CTL1_[n].AT, PCIE_DMAWR_CTL1_[n].RO, PCIE_DMAWR_CTL1_[n].NS, PCIE_DMAWR_CTL1_[n].TC, PCIE_DMAWR_CTL1_[n].FNUM and PCIE_DMAWR_CTL1_[n].RIE bit fields.
5. Set the PCIE_DMAWR_CTL1_[n].LIE and the PCIE_DMAWR_CTL1_[n].TD bits.
6. Set the PCIE_DMAWR_XFRSZ_0[n] register to 0x0010 0000 (1 MB).
7. Set the PCIE_DMAWR_SAR_LO_[n] register to 0x2400 0000.
8. Set the PCIE_DMAWR_SAR_HI_[n] register to 0x0000 0000.
9. Set the PCIE_DMAWR_DAR_LO_[n] register to 0x5000 0000.
10. Set the PCIE_DMAWR_DAR_HI_[n] register to 0x0000 0000.
11. Write 0x0 to the PCIE_DMAWR_DRBL_[n] register.

DMA Read Transfer Example

Transfer 1MB from remote PCIe memory space (0x5000 0000) to local L2 (0x2400 0000)

1. Write 0x8000_0000 to the PCIE_DMA_VWPRT_SEL_[n] register to select channel 0 read registers.
2. Write 0x1 to the PCIE_DMARD_EN_[n].EN bit.
3. Write 0x0 to the PCIE_DMARD_IMSK_[n] register.
4. Clear the PCIE_DMARD_CTL1_[n].AT, PCIE_DMARD_CTL1_[n].RO, PCIE_DMARD_CTL1_[n].NS, PCIE_DMARD_CTL1_[n].TC, PCIE_DMARD_CTL1_[n].FNUM and PCIE_DMARD_CTL1_[n].RIE bit fields.
5. Set the PCIE_DMARD_CTL1_[n].LIE and the PCIE_DMARD_CTL1_[n].TD bits.
6. Set the PCIE_DMARD_XFRSZ_[n] register to 0x0010 0000 (1 MB).
7. Set the PCIE_DMARD_SAR_LO_[n] register to 0x2400 0000.
8. Set the PCIE_DMARD_SAR_LO_[n] register to 0x0000 0000.
9. Set the PCIE_DMARD_DAR_LO_[n] register to 0x5000 0000.
10. Set the PCIE_DMARD_DAR_HI_[n] register to 0x0000 0000.
11. Write 0x0 to the PCIE_DMARD_DRBL_[n] register.

Device Reset

There are two reset types for the ADI PCIe core. The first is initial power-up and is only reached from chip initialization. This has been referred to as Cold reset in prior descriptions. The key to this step is making sure the Physical layer is correctly out of reset and setting the device into the proper operating mode, either Root Complex or End point. The second type of reset is one initiated from an upstream port, referred to as a hot reset request. The hot reset does not require resetting of the physical layer but does require re-training the link and setup of all configuration registers, address translation and DMA engines. The cold reset sequence always proceeds to a hot reset.

Cold Reset

The cold reset is entered via power up or a system reset. This is also referred to as a fundamental reset. The PERST# signal is not connected to the system reset by default. The following steps prepare the hardware to enter the Hot reset sequence.

1. Take the core out of reset and configure as Root complex or Endpoint.

```
if (*pREG_RSCK_CTL & 0x01)  // check if it is reset.
         {
*pREG_RSCK_CTL = 0x00000002; //take core out of reset. set to end point
 delay(256);
          }
```

2. Wait for the core to come out of rest.

```
While (*pREG_RSCK_STAT & 0x1);
```

Hot Reset

The hot reset is entered via the upstream port PCIe TSI ordered sets. This is typically a software driven event. The ADI PCIe core intercepts this bit and when detected sets a bit in the `PCIE_RSCK_STAT` register. Additionally, this bit can trigger an interrupt via the APP_STAT interrupt. The reception of the Hot Reset command indicates the link must re-enter detect state. All configuration registers and address mapping must be re-initialized.

1. Either detect the hot reset request (via the APP_STAT interrupt or polling the `PCIE_RSCK_STAT` register) or enter from the fundamental reset.
2. Clear the reset events and verify

```
if (*pREG_RSCK_STAT & 0x20000)  // lock write error bit is set
{
*pREG_RSCK_STAT = 0x20000;    // clear lock write error bit
 }
 if (*pREG_RSCK_STAT & 0x20)  // reset bit is set
 {
 *pREG_RSCK_STAT = 0x20;  // clear  Hot reset request
 }
 // wait for device to come out of reset
 while ( (*pREG_RSCK_STAT & 0x1F) !=0 );
```

3. Setup the configuration registers, including BAR address size, MSIs and custom device id. Finally, enable the IO and command register space. The program must enable the DBI registers in order to access the configuration space.

```
*pREG_PCIE0_MISC_CONTROL_1_REG0 = 0x1; //Write enable for DBI Read register

*pREG_PCIE0_BAR0_REG0    = 0x08;        // 32 bit address
*pREG_PCIE0_BAR1_CS    = 0x00000000;    //  disable bar1-4
*pREG_PCIE0_BAR2_CS    = 0x00000000;
*pREG_PCIE0_BAR3_CS    = 0x00000000;
*pREG_PCIE0_BAR4_CS    = 0x00000000;

*pREG_PCIE0_TYPE1_DEV_ID_VEND_ID_REG0 =   ((DEV_ID << 16) |VEND_ID);;
*pREG_PCIE0_TYPE1_SUBSYS_ID_VEND_REG0 =   ( (EP_SS_ID << 16) | SSVID_ID );
*pREG_PCIE0_TYPE1_STATUS_COMMAND_REG0   |= 3;
```

4. Setup the address translation. This example matches the incoming BAR0 and translates to L2RAM at 0x2xxxxxxx.

```
*pREG_PCIE0_IATU_VIEWPORT_REG0 =   0x80000000;        // inbound region 0
*pREG_PCIE0_PL_IATU_LWR_TARGET_ADDR_REG_OUTBOUND_00   =   0x20000000;
*pREG_PCIE0_PL_IATU_UPPER_TARGET_ADDR_REG_OUTBOUND_00 =   0x0;
*pREG_PCIE0_PL_IATU_REGION_CTRL_REG_1_OUTBOUND_00     =    0x0;    // MEMr/w
*pREG_PCIE0_PL_IATU_REGION_CTRL_REG_2_OUTBOUND_00   =   0xC0000000;  // BAR0
```

5. Enter link training and wait for link establishment. Link establishment can be detected by polling the `PCIE_APP_STAT` register or via interrupt.

```
// bits[3:0]link speed 1 = 2.5G, 2=5G
*pREG_PCIE_USP0_LINK_CAPABILITIES = 0x0040AC11;
*pREG_APP_CTL = 0x1;                 // start training process
While  (!(*pREG_APP_STAT & 0x01) );       // wait for link to train
```

6. If the device is configured as an endpoint, wait for the upstream port to configure the bus enable bit. If the device is a Root complex begin configuration of endpoint.

```
while ( !(*pREG_PCIE0_TYPE1_STATUS_COMMAND_REG0 & 0x04 ) ); //endpoint
```

7. Begin normal processing while monitoring link status.

ADSP-SC58x Product Specific Information

The following information is specific to PCIe operation on the ADSP-SC58x processors.

PHY Parameter Control

Several signals set static or programmable values that are used by the PHY. Some of these parameters are package-dependent and their programmability is advantageous. Systems drive these from a register with a default value set.

The values for transmit de-emphasis and amplitude are derived from the following equations.

$$\text{TX de-emphasis (dB)} = 20 \times \log_{10}\left[\frac{128 - 2 \times \text{pcs\_tx\_deemph}}{128}\right]$$

$$\text{TX amplitude (V)} = \text{vptx} \times \left[\frac{\text{pcs\_tx\_swing} + 1}{128}\right]$$

Using these relationships, the default settings for 1.0-V launch amplitude and transmit equalization are:

Transmit de-emphasis for PCI Express:

- pcs_tx_deemph: PHY_TXDEEMPH.GEN1 = 24 (0x18) (Typical setting for 2.5 Gb/s operation)

  PHY_TXDEEMPH.GEN2_35 = 24 (0x18) (Typical setting for 5.0 Gb/s operation, 3.5 dB)

  PHY_TXDEEMPH.GEN2_6 = 33 (0x21) (Typical setting for 5.0 Gb/s operation, 6 dB)

Transmit launch amplitude:

- pcs_tx_swing: PHY_TXSWING.FULL = 115 (0x73) (For the default 1.0 V launch amplitude)

  PHY_TXSWING.LOW = 115 (0x73) (To support Mobile mode)

ADSP-SC58x PCIE Register Descriptions

(PCIE) contains the following registers.

Table 29-18: ADSP-SC58x PCIE Register List

Name	Description
PCIE_ACK_ASPMCTL_[n]	Acknowledge Frequency and L0-L1 ASPM Control Register
PCIE_ACK_TMR_[n]	Acknowledge Latency Timer and Replay Timer Register
PCIE_ADVERR_CAPBCTL_[n]	Advanced Error Capabilities and Control Register
PCIE_ADVERR_EXTCAPB_[n]	Advanced Error Reporting Extended Capability Header Register
PCIE_APP_CORERR_MSG	Correctable Error Message Requester ID Register
PCIE_APP_CTL	Application Control Register
PCIE_APP_DIAGCTL	Diagnostic Control Register
PCIE_APP_DIAGSTAT	Diagnostic Status Register
PCIE_APP_FERR_MSG	Fatal Error Message Requester ID Register
PCIE_APP_INTSTAT	Application Interrupts Status Register
PCIE_APP_NFERR_MSG	Non Fatal Error Message Requester ID Register
PCIE_APP_PMACK_MSG	Power Management Acknowledge Message Requester ID Register
PCIE_APP_PMEVT_MSG	Power Management Event Message Requester ID Register
PCIE_APP_PMTOFF_MSG	Power Management Turn Off Message Requester ID Register
PCIE_APP_STAT	Application Status Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_APP_UNLK_MSG	Unlock Message Requester ID Register
PCIE_APP_VNDR_MSG	Vendor Message Requester ID Register
PCIE_APP_VNDR_MSG_HDR0	Vendor Message Header Bytes 8 to 11 Register
PCIE_APP_VNDR_MSG_HDR1	Vendor Message Header Bytes 12 to 15 Register
PCIE_AUX_CLKFREQ_[n]	Auxiliary Clock Frequency Control Register
PCIE_BERR_RESP_[n]	Peripheral and SCB Bridge Slave Error Response Register
PCIE_BUSMGR_WDOG_[n]	Bus Order Manager Watchdog Off Register
PCIE_BUS_MULOB_DECOMP_[n]	Bus Multiple Outbound Decomposition SubReq Control Register
PCIE_CACHE_CTL1_[n]	ACE Cache Coherency Control Register 1
PCIE_CACHE_CTL3_[n]	ACE Cache Coherency Control Register 3
PCIE_CAPB_NPTR_[n]	PCIe Capabilities, ID, Next Pointer Register
PCIE_CORERR_MSK_[n]	Correctable Error Mask Register
PCIE_CORERR_STAT_[n]	Correctable Error Status Register
PCIE_DEV_CAPB2_[n]	Device Capabilities 2 Register
PCIE_DEV_CTLSTAT2_[n]	Device Control 2 and Status 2 Register
PCIE_DEV_CTLSTAT_[n]	Device Control and Status Register
PCIE_DMARD_ARBWGT_LO_[n]	DMA Read Arbitration Weight Low Off Register
PCIE_DMARD_CH01_IMWR_[n]	DMA Read Channel 1 and 0 IMWr Data Register
PCIE_DMARD_CTL1_[n]	DMA Control 1 Read Channel Register
PCIE_DMARD_DAR_HI_[n]	DMA Destination Address High Read Channel Register
PCIE_DMARD_DAR_LO_[n]	DMA Destination Address Low Read Channel Register
PCIE_DMARD_DONE_IMWR_HI_[n]	DMA Read Done IMWr Address High Register
PCIE_DMARD_DONE_IMWR_LO_[n]	DMA Read Done IMWr Address Low Register
PCIE_DMARD_DRBL_[n]	DMA Read Doorbell Register
PCIE_DMARD_EN_[n]	DMA Read Engine Enable Register
PCIE_DMARD_ERRSTAT_HI_[n]	DMA Read Error Status High Register
PCIE_DMARD_ERRSTAT_LO_[n]	DMA Read Error Status Low Register
PCIE_DMARD_ICLR_[n]	DMA Read Interrupt Clear Register
PCIE_DMARD_IMSK_[n]	DMA Read Interrupt Mask Register
PCIE_DMARD_IMWRABRT_HI_[n]	DMA Read Abort IMWr Address High Register
PCIE_DMARD_IMWRABRT_LO_[n]	DMA Read Abort IMWr Address Low Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_DMARD_ISTAT_[n]	DMA Read Interrupt Status Register
PCIE_DMARD_LLP_HI_[n]	DMA LLP High Read Channel Register
PCIE_DMARD_LLP_LO_[n]	DMA LLP Low Read Channel Register
PCIE_DMARD_LLSTERR_EN_[n]	DMA Read Linked List Error Enable Register
PCIE_DMARD_SAR_HI_[n]	DMA SAR High Read Channel Register
PCIE_DMARD_SAR_LO_[n]	DMA SAR Low Read Channel Register
PCIE_DMARD_XFRSZ_[n]	DMA Transfer Size Read Channel Register
PCIE_DMAWR_ABRT_IMWR_HI[n]	DMA Write Abort IMWr Address High Register
PCIE_DMAWR_ABRT_IMWR_LO[n]	DMA Write Abort IMWr Address Low Register
PCIE_DMAWR_ARBWGT_LO_[n]	DMA Write Channel Arbitration Weight Low Off Register
PCIE_DMAWR_CH01_IMWR_[n]	DMA Write Channel 1 and 0 IMWr Data Register
PCIE_DMAWR_CTL1_[n]	DMA Control 1 Write Channel Register
PCIE_DMAWR_DAR_HI_[n]	DMA Dar High Write Channel Register
PCIE_DMAWR_DAR_LO_[n]	DMA DAR Low Write Channel Register
PCIE_DMAWR_DONE_IMWR_HI_[n]	DMA Write Done IMWr Interrupt Address High Register
PCIE_DMAWR_DONE_IMWR_LO_[n]	DMA Write Done IMWr Address Low Register
PCIE_DMAWR_DRBL_[n]	DMA Write Doorbell Register
PCIE_DMAWR_EN_[n]	DMA Write Engine Enable Register
PCIE_DMAWR_ERRSTAT_[n]	DMA Write Error Status Register
PCIE_DMAWR_ICLR_[n]	DMA Write Interrupt Clear Register
PCIE_DMAWR_IMSK_[n]	DMA Write Interrupt Mask Register
PCIE_DMAWR_ISTAT_[n]	DMA Write Interrupt Status Register
PCIE_DMAWR_LLP_HI_[n]	DMA LLP High Write Channel Register
PCIE_DMAWR_LLP_LO_[n]	DMA LLP Low Write Channel Register
PCIE_DMAWR_LLSTERR_EN_[n]	DMA Write Linked List Error Enable Register
PCIE_DMAWR_PREQ_TMR_[n]	DMA Write Posted Request Deadlock Timer Register
PCIE_DMAWR_SAR_HI_[n]	DMA SAR High Write Channel Register
PCIE_DMAWR_SAR_LO_[n]	DMA SAR Low Write Channel Register
PCIE_DMAWR_XFRSZ_0[n]	DMA Transfer Size Write Channel Register
PCIE_DMA_CTL_[n]	DMA Number of Channels Register
PCIE_DMA_VWPRT_SEL_[n]	DMA Channel Context Index Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_EP_BAR0_MASK[n]	Endpoint Base Address Mask Register 0
PCIE_EP_BAR0_[n]	Endpoint Base Address Register 0
PCIE_EP_BAR1_MASK[n]	Endpoint Base Address Mask Register 1
PCIE_EP_BAR1_[n]	Endpoint Base Address Register 1
PCIE_EP_BAR2_MASK[n]	Endpoint Base Address Mask Register 2
PCIE_EP_BAR2_[n]	Endpoint Base Address Register 2
PCIE_EP_BAR3_MASK[n]	Endpoint Base Address Mask Register 3
PCIE_EP_BAR3_[n]	Endpoint Base Address Register 3
PCIE_EP_BAR4_MASK[n]	Endpoint Base Address Mask Register 4
PCIE_EP_BAR4_[n]	Endpoint Base Address Register 4
PCIE_EP_BAR5_MASK[n]	Endpoint Base Address Mask Register 5
PCIE_EP_BAR5_[n]	End Point Base Address Register 5
PCIE_EP_CAPBPTR_[n]	Capability Pointer Register
PCIE_EP_CCRID_[n]	Class Code and Revision ID Register
PCIE_EP_CFG_[n]	End Point Configuration Register
PCIE_EP_CRDBPTR_[n]	CardBus CIS Pointer Register
PCIE_EP_DEVCAPB_[n]	Device Capabilities Register
PCIE_EP_ID_[n]	Device ID and Vendor ID Register
PCIE_EP_PINLN_INT_[n]	Interrupt Line and Pin Register
PCIE_EP_ROMCFG_[n]	End Point Expansion ROM Base Address Register
PCIE_EP_SSVID_[n]	Subsystem ID and Subsystem Vendor ID Register
PCIE_EP_STATCMD_[n]	Command and Status Register
PCIE_ERRSRC_ID_[n]	Error Source Identification Register
PCIE_FILTMSK2_[n]	Filter Mask 2 Register
PCIE_GEN2_CTL_[n]	Link Width and Speed Change Control Register
PCIE_HDRLOG0_[n]	Header Log Register 0
PCIE_HDRLOG1_[n]	Header Log Register 1
PCIE_HDRLOG2_[n]	Header Log Register 2
PCIE_HDRLOG3_[n]	Header Log Register 3
PCIE_IATU_CTL1_INB_[n]	iATU Region Control 1 Inbound Register
PCIE_IATU_CTL1_OUTB_[n]	iATU Region Control 1 Outbound Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_IATU_CTL2_INB_[n]	iATU Region Control 2 Inbound Register
PCIE_IATU_CTL2_OUTB_[n]	iATU Region Control 2 Register Outbound
PCIE_IATU_CTL3_INB_[n]	iATU Region Control 3 Register
PCIE_IATU_CTL3_OUTB_[n]	iATU Region Control 3 Register
PCIE_IATU_LBADDR_INB_[n]	IATU Lower Base Inbound Address Register
PCIE_IATU_LBADDR_OUTB_[n]	IATU Lower Base Outbound Address Register
PCIE_IATU_LMTADDR_INB_[n]	IATU Inbound Limit Address Register
PCIE_IATU_LMTADDR_OUTB_[n]	IATU Outbound Limit Address Register
PCIE_IATU_LTADDR_INB_[n]	IATU Lower Target Address Inbound Register
PCIE_IATU_LTADDR_OUTB_[n]	IATU Lower Target Address Outbound Register
PCIE_IATU_UBADDR_INB_[n]	IATU Upper Base Address Inbound Register
PCIE_IATU_UBADDR_OUTB_[n]	IATU Upper Base Address Outbound Register
PCIE_IATU_UTADDR_INB_[n]	iATU Upper Target Address Inbound Register
PCIE_IATU_UTADDR_OUTB_[n]	iATU Upper Target Address Outbound Register
PCIE_IATU_VWPRT_[n]	IATU View Port Register
PCIE_LANE_SKEW_[n]	Lane Skew Register
PCIE_LNK_CAPB2_[n]	Link Capabilities 2 Register
PCIE_LNK_CAPB_[n]	Link Capabilities Register
PCIE_LNK_CTLSTAT2_[n]	Link Control 2 and Status 2 Register
PCIE_LNK_CTLSTAT_[n]	Link Control and Status Register
PCIE_MISC_CTL1_[n]	DBI Read-Only Write Enable Register
PCIE_MSI_CAPB_NPTR_[n]	MSI Capability ID, Next Pointer and Control Register
PCIE_MSI_CAPB_OFF4_[n]	MSI Capability Offset Register
PCIE_MSI_CAPB_OFF8_[n]	MSI Capability Offset Register
PCIE_MSI_CAPB_OFFC_[n]	MSI Capability Offset Register
PCIE_MSI_CTL_LADDR_[n]	MSI Controller Lower Address Register
PCIE_MSI_CTL_UADDR_[n]	MSI Controller Upper Address Register
PCIE_MSI_GPIO_IO_[n]	MSI Controller General-Purpose IO Register
PCIE_MSI_IEN0_[n]	MSI Controller Interrupt 0 Enable Register
PCIE_MSI_IEN1_[n]	MSI Controller Interrupt 1 Enable Register
PCIE_MSI_IEN2_[n]	MSI Controller Interrupt 2 Enable Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
`PCIE_MSI_IEN3_[n]`	MSI Controller Interrupt 3 Enable Register
`PCIE_MSI_IEN4_[n]`	MSI Controller Interrupt 4 Enable Register
`PCIE_MSI_IEN5_[n]`	MSI Controller Interrupt 5 Enable Register
`PCIE_MSI_IEN6_[n]`	MSI Controller Interrupt 6 Enable Register
`PCIE_MSI_IEN7_[n]`	MSI Controller Interrupt 7 Enable Register
`PCIE_MSI_IMSK0_[n]`	MSI Controller Interrupt 0 Mask Register
`PCIE_MSI_IMSK1_[n]`	MSI Controller Interrupt 1 Mask Register
`PCIE_MSI_IMSK2_[n]`	MSI Controller Interrupt 2 Mask Register
`PCIE_MSI_IMSK3_[n]`	MSI Controller Interrupt 3 Mask Register
`PCIE_MSI_IMSK4_[n]`	MSI Controller Interrupt 4 Mask Register
`PCIE_MSI_IMSK5_[n]`	MSI Controller Interrupt 5 Mask Register
`PCIE_MSI_IMSK6_[n]`	MSI Controller Interrupt 6 Mask Register
`PCIE_MSI_IMSK7_[n]`	MSI Controller Interrupt 7 Mask Register
`PCIE_MSI_ISTAT0_[n]`	MSI Controller Interrupt 0 Status Register
`PCIE_MSI_ISTAT1_[n]`	MSI Controller Interrupt 1 Status Register
`PCIE_MSI_ISTAT2_[n]`	MSI Controller Interrupt 2 Status Register
`PCIE_MSI_ISTAT3_[n]`	MSI Controller Interrupt 3 Status Register
`PCIE_MSI_ISTAT4_[n]`	MSI Controller Interrupt 4 Status Register
`PCIE_MSI_ISTAT5_[n]`	MSI Controller Interrupt 5 Status Register
`PCIE_MSI_ISTAT6_[n]`	MSI Controller Interrupt 6 Status Register
`PCIE_MSI_ISTAT7_[n]`	MSI Controller Interrupt 7 Status Register
`PCIE_PHY_TXDEEMPH`	TX De-emphasis Parameters Register
`PCIE_PHY_TXSWING`	TX Launch Amplitude Register
`PCIE_PLCTL_[n]`	Port Logic Link Control Register
`PCIE_PLDBG0_[n]`	Port Logic Debug0 Register
`PCIE_PLDBG1_[n]`	Port Logic Debug1 Register
`PCIE_PL_FRC_[n]`	Port Force Link Register
`PCIE_PL_PHYCTL_[n]`	Port Control PHY Control Register
`PCIE_PL_PHYSTAT_[n]`	Port Control PHY Status Register
`PCIE_PMT_CAPB_[n]`	Power Management Capabilities Register
`PCIE_PMT_CTLSTAT_[n]`	Power Management Control and Status Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_QSTAT_[n]	Queue Status Register
PCIE_RC_BRDG_ILPCTL_[n]	Interrupt Bridge Line and Pin Control Register
PCIE_RC_CAPBPTR_[n]	Capability Pointer Register.
PCIE_RC_CCRID_[n]	Class Code and Revision ID Register
PCIE_RC_CFG_[n]	Root Complex Configuration Register
PCIE_RC_CTLCAPB_[n]	Root Control and Capabilities Register
PCIE_RC_DEVCAPB_[n]	Device Capabilities Register
PCIE_RC_ERRCMD_[n]	Root Complex Error Command Register
PCIE_RC_ERRSTAT_[n]	Root Error Status Register
PCIE_RC_ID_[n]	Device ID and Vendor ID Register
PCIE_RC_IOBL_[n]	Root Complex I/O Base and Limit Upper 16 bits Register
PCIE_RC_MBL_[n]	Memory Base and Memory Limit Register
PCIE_RC_PREFMBL_[n]	Prefetchable Memory Base and Limit Register
PCIE_RC_PREF_BUPP_[n]	Prefetchable Base Upper 32 Bits Register
PCIE_RC_PREF_LMT_UP_[n]	Prefetchable Limit Upper 32 Bits Register
PCIE_RC_ROMCFG_[n]	Root Complex Expansion ROM Base Address Register
PCIE_RC_SECSTAT_[n]	Secondary Status and I/O Base and Limit Register
PCIE_RC_STATCMD_[n]	Command and Status Register
PCIE_RC_STAT_[n]	Root Status Register
PCIE_RC_TMRLAT_[n]	Latency Timer Register
PCIE_RSCK_CTL	Reset and Clock Control Register
PCIE_RSCK_STAT	Reset and Clock Status
PCIE_SYM_TMRFILT1_[n]	Symbol Timer Filter 1 Off
PCIE_TMRCTL_MFN_[n]	Timer Control and Max Function Number Register
PCIE_TXCOMP_FCCSTAT_[n]	Transmit Completion FC Credit Status Register
PCIE_TX_NPST_FCCSTAT_[n]	Transmit Non-Posted FC Credit Status Register
PCIE_TX_PST_FCCSTAT_[n]	Transmit Posted FC Credit Status Register
PCIE_UNCOR_ERRMSK_[n]	Uncorrectable Error Mask Register
PCIE_UNCOR_ERRSEV_[n]	Uncorrectable Error Severity Register
PCIE_UNCOR_ERRSTAT_[n]	Uncorrectable Error Status Register
PCIE_VC0_COMP_RXQCTL_[n]	Segmented-Buffer VC0 Completion Receive Queue Control Register

Table 29-18: ADSP-SC58x PCIE Register List (Continued)

Name	Description
PCIE_VC0_NPST_RXQCTL_[n]	Segmented-Buffer VC0 Non-Posted Receive Queue Control Register
PCIE_VC0_PST_RXQCTL_[n]	Segmented-Buffer VC0 Posted Receive Queue Control Register
PCIE_VC_TXARB1_[n]	Virtual Channel Transmit Arbitration Register 1
PCIE_VC_TXARB2_[n]	Virtual Channel Transmit Arbitration Register 2
PCIE_VEND_DLLP_[n]	Vendor Specific DLLP Register

Acknowledge Frequency and L0-L1 ASPM Control Register

The `PCIE_ACK_ASPMCTL_[n]` register controls several aspects of Active State Power Management acknowledge.

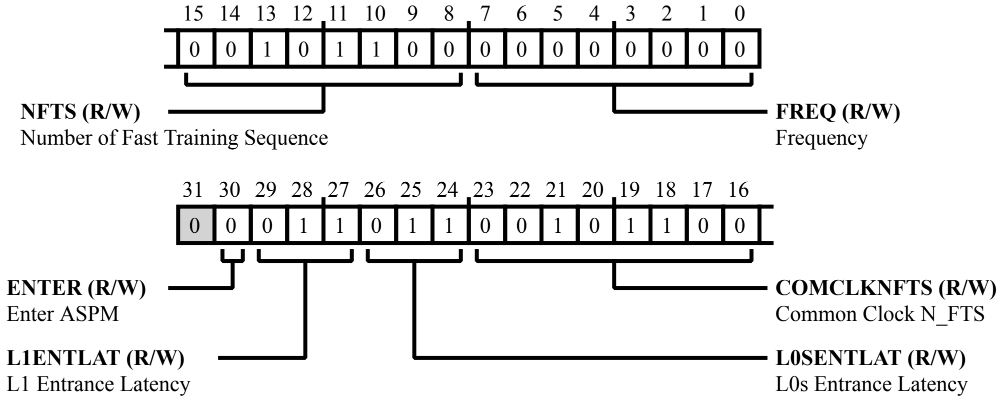

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0
NFTS (R/W)
Number of Fast Training Sequence
FREQ (R/W)
Frequency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0
ENTER (R/W)
Enter ASPM
L1ENTLAT (R/W)
L1 Entrance Latency
COMCLKNFTS (R/W)
Common Clock N_FTS
L0SENTLAT (R/W)
L0s Entrance Latency

Figure 29-10: PCIE_ACK_ASPMCTL_[n] Register Diagram

Table 29-19: PCIE_ACK_ASPMCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	ENTER	Enter ASPM. The `PCIE_ACK_ASPMCTL_[n].ENTER` bit allows the core to enter ASPM L1 even when link partner did not go to L0s (receive is not in L0s). When cleared (=0), the core goes to ASPM L1 only after an idle period during which both receive and transmit are in L0s. This register field is sticky.	
29:27 (R/W)	L1ENTLAT	L1 Entrance Latency. The `PCIE_ACK_ASPMCTL_[n].L1ENTLAT` bit field specifies the L1 Entrance Latency. Programming this timer with a value greater that 32 us has no effect unless extended sync is used, or all of the credits are infinite. This register field is sticky.	
		0	1 us
		1	2 us
		2	4 us
		3	8 us
		4	16 us
		5	32 us
		6	64 us
		7	64 us

Table 29-19: PCIE_ACK_ASPMCTL_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>26:24
(R/W)</td><td>L0SENTLAT</td><td colspan="2">L0s Entrance Latency.
The <code>PCIE_ACK_ASPMCTL_[n].L0SENTLAT</code> bit field specifies the L0s entrance latency. This register field is sticky.</td></tr>
<tr><td></td><td></td><td>0</td><td>1 us</td></tr>
<tr><td></td><td></td><td>1</td><td>2 us</td></tr>
<tr><td></td><td></td><td>2</td><td>3 us</td></tr>
<tr><td></td><td></td><td>3</td><td>4 us</td></tr>
<tr><td></td><td></td><td>4</td><td>5 us</td></tr>
<tr><td></td><td></td><td>5</td><td>6 us</td></tr>
<tr><td></td><td></td><td>6</td><td>7 us</td></tr>
<tr><td></td><td></td><td>7</td><td>7 us</td></tr>
<tr><td>23:16
(R/W)</td><td>COMCLKNFTS</td><td colspan="2">Common Clock N_FTS.
The <code>PCIE_ACK_ASPMCTL_[n].COMCLKNFTS</code> bit field specifies the number of Fast Training Sequence ordered sets transmitted when transitioning from L0s to L0 when a common clock is used. The maximum number of FTS ordered-sets that a component can request is 255. This field is only writable (sticky) when all of the following configuration parameter equations are true:
• CX_NFTS !=CX_COMM_NFTS
• DEFAULT_L0S_EXIT_LATENCY !=DEFAULT_COMM_L0S_EXIT_LATENCY
• DEFAULT_L1_EXIT_LATENCY !=DEFAULT_COMM_L1_EXIT_LATENCY
The core does not support a value of zero; a value of zero can cause the LTSSM to go into the recovery state when exiting from L0s. The access attributes of this field are as follows:
• Wire: R
• Dbi: R
• Dbi2: R</td></tr>
<tr><td>15:8
(R/W)</td><td>NFTS</td><td colspan="2">Number of Fast Training Sequence.
The <code>PCIE_ACK_ASPMCTL_[n].NFTS</code> bit field specifies the number of Fast Training Sequence ordered sets transmitted when transitioning from L0s to L0. The maximum number of FTS ordered-sets that a component can request is 255. The core does not support a value of zero; a value of zero can cause the LTSSM to go into the recovery state when exiting from L0s.</td></tr>
</table>

Table 29-19: PCIE_ACK_ASPMCTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FREQ	Frequency. The `PCIE_ACK_ASPMCTL_[n].FREQ` bit field specifies the number of pending Acks the core accumulates (up to 255) before sending an Ack DLLP. Zero indicates that this Ack frequency control feature is turned off and that the core accumulates 255 received TLPs before sending an Ack DLLP. Other values indicate that the core Acks every received TLP.

Acknowledge Latency Timer and Replay Timer Register

The `PCIE_ACK_TMR_[n]` register controls Ack latency timer limits and replay timer limits.

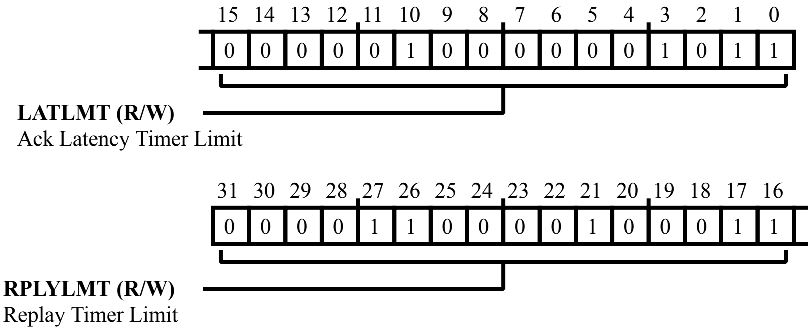

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
LATLMT (R/W)
Ack Latency Timer Limit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1
RPLYLMT (R/W)
Replay Timer Limit

Figure 29-11: PCIE_ACK_TMR_[n] Register Diagram

Table 29-20: PCIE_ACK_TMR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	RPLYLMT	Replay Timer Limit. The `PCIE_ACK_TMR_[n].RPLYLMT` bit field specifies the replay timer expiration limit. The core initiates a replay on reception of a Nak or when the replay timer expires. For more details, see "Transmit Retry". Programs can modify the effective timer limit with the `PCIE_TMRCTL_MFN_[n].RLMT` bit field. After reset, the core updates the default according to the Negotiated Link Width and Max_Payload_Size. When operating at 5 Gb/s, then an additional 153/CX_NB is added for additional internal processing for received TLPs and transmitted DLLPs.
15:0 (R/W)	LATLMT	Ack Latency Timer Limit. The `PCIE_ACK_TMR_[n].LATLMT` bit field specifies the Ack latency timer expire limit. Programs can modify the effective timer limit with the `PCIE_TMRCTL_MFN_[n].RLMT` bit field. After reset, the core updates the default according to the Negotiated Link Width and Max_Payload_Size. When operating at 5 Gb/s, an additional 51/CX_NB is added for additional internal processing for received TLPs and transmitted DLLPs.

Advanced Error Capabilities and Control Register

The PCIE_ADVERR_CAPBCTL_[n] register controls some ECRC capabilities and their function.

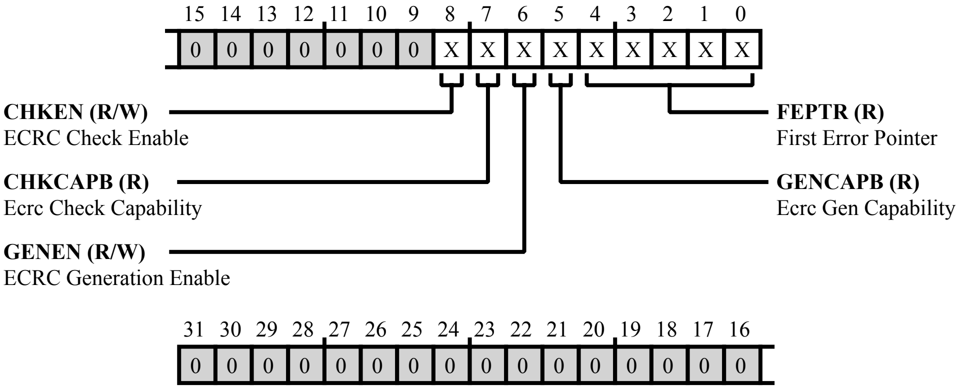

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 X X X X X X X X X
CHKEN (R/W)
ECRC Check Enable
CHKCAPB (R)
Ecrc Check Capability
GENEN (R/W)
ECRC Generation Enable
FEPTR (R)
First Error Pointer
GENCAPB (R)
Ecrc Gen Capability
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-12: PCIE_ADVERR_CAPBCTL_[n] Register Diagram

Table 29-21: PCIE_ADVERR_CAPBCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	CHKEN	ECRC Check Enable. The PCIE_ADVERR_CAPBCTL_[n].CHKEN bit enables ECRC checking. Functions that do not implement the associated mechanism are permitted to set this bit to 0. The default value of this bit is 0 (disabled).
7 (R/NW)	CHKCAPB	Ecrc Check Capability. The PCIE_ADVERR_CAPBCTL_[n].CHKCAPB bit indicates that the Function is capable of checking ECRC.
6 (R/W)	GENEN	ECRC Generation Enable. The PCIE_ADVERR_CAPBCTL_[n].GENEN bit enables ECRC. Functions that do not implement the associated mechanism are permitted to hardwire this bit to 0. The default value of this bit is 0 (disabled).
5 (R/NW)	GENCAPB	Ecrc Gen Capability. If set, the PCIE_ADVERR_CAPBCTL_[n].GENCAPB bit indicates that the Function is capable of generating an ECRC.
4:0 (R/NW)	FEPTR	First Error Pointer. The PCIE_ADVERR_CAPBCTL_[n].FEPTR bit field identifies the bit position of the first error reported in the PCIE_UNCOR_ERRSTAT_[n] register.

Advanced Error Reporting Extended Capability Header Register

The `PCIE_ADVERR_EXTCAPB_[n]` register provides optional extended error reporting capability. It implemented by PCI Express device Functions that support advanced error control and reporting.

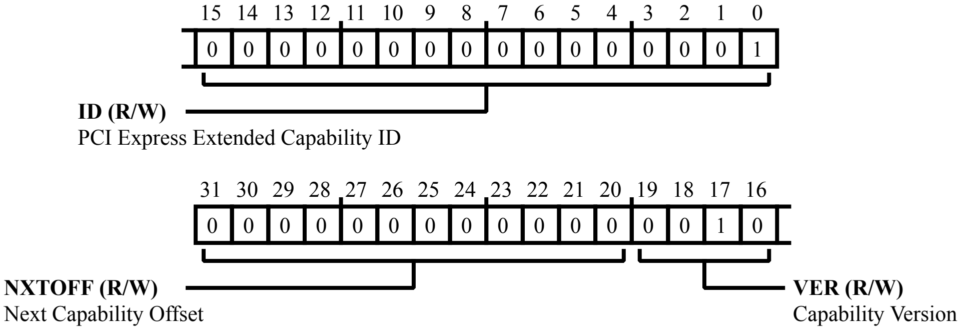

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ID (R/W)
PCI Express Extended Capability ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
NXTOFF (R/W)
Next Capability Offset
VER (R/W)
Capability Version

Figure 29-13: PCIE_ADVERR_EXTCAPB_[n] Register Diagram

Table 29-22: PCIE_ADVERR_EXTCAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:20 (R/W)	NXTOFF	Next Capability Offset. The `PCIE_ADVERR_EXTCAPB_[n].NXTOFF` bit field contains the offset to the next PCI Express capability structure or 0x0 if no other items exist in the linked list of capabilities. For extended capabilities implemented in configuration space, this offset is relative to the beginning of PCI compatible configuration space and thus must always be either 0x0 (for terminating list of capabilities) or greater than 0xFF.
19:16 (R/W)	VER	Capability Version. The `PCIE_ADVERR_EXTCAPB_[n].VER` bit field is a PCI-SIG defined version number that indicates the version of the Capability structure present. This field must be 0x2 if the End-End TLP Prefix Supported bit is set and must be 0x1 or 0x2 otherwise.
15:0 (R/W)	ID	PCI Express Extended Capability ID. The `PCIE_ADVERR_EXTCAPB_[n].ID` bit field is read-only and reports the PCI-SIG defined ID number that indicates the nature and format of the Extended Capability. The Extended Capability ID for the Advanced Error Reporting Capability is 0x001.

Correctable Error Message Requester ID Register

The `PCIE_APP_CORERR_MSG` register contains the requester ID of the latest received Correctable Error Message. It is advised to read this register when the `PCIE_APP_INTSTAT.CORERR` bit is set.

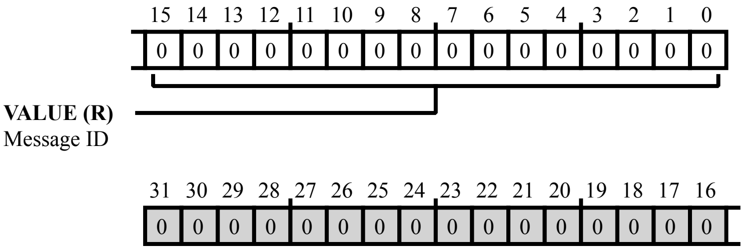

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-14: PCIE_APP_CORERR_MSG Register Diagram

Table 29-23: PCIE_APP_CORERR_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID. The `PCIE_APP_CORERR_MSG.VALUE` bit field contains the requester ID of the latest received Correctable Error Message.

Application Control Register

The `PCIE_APP_CTL` register contains bits that control various PCIE functions. This register also reports whether interrupts are enabled.

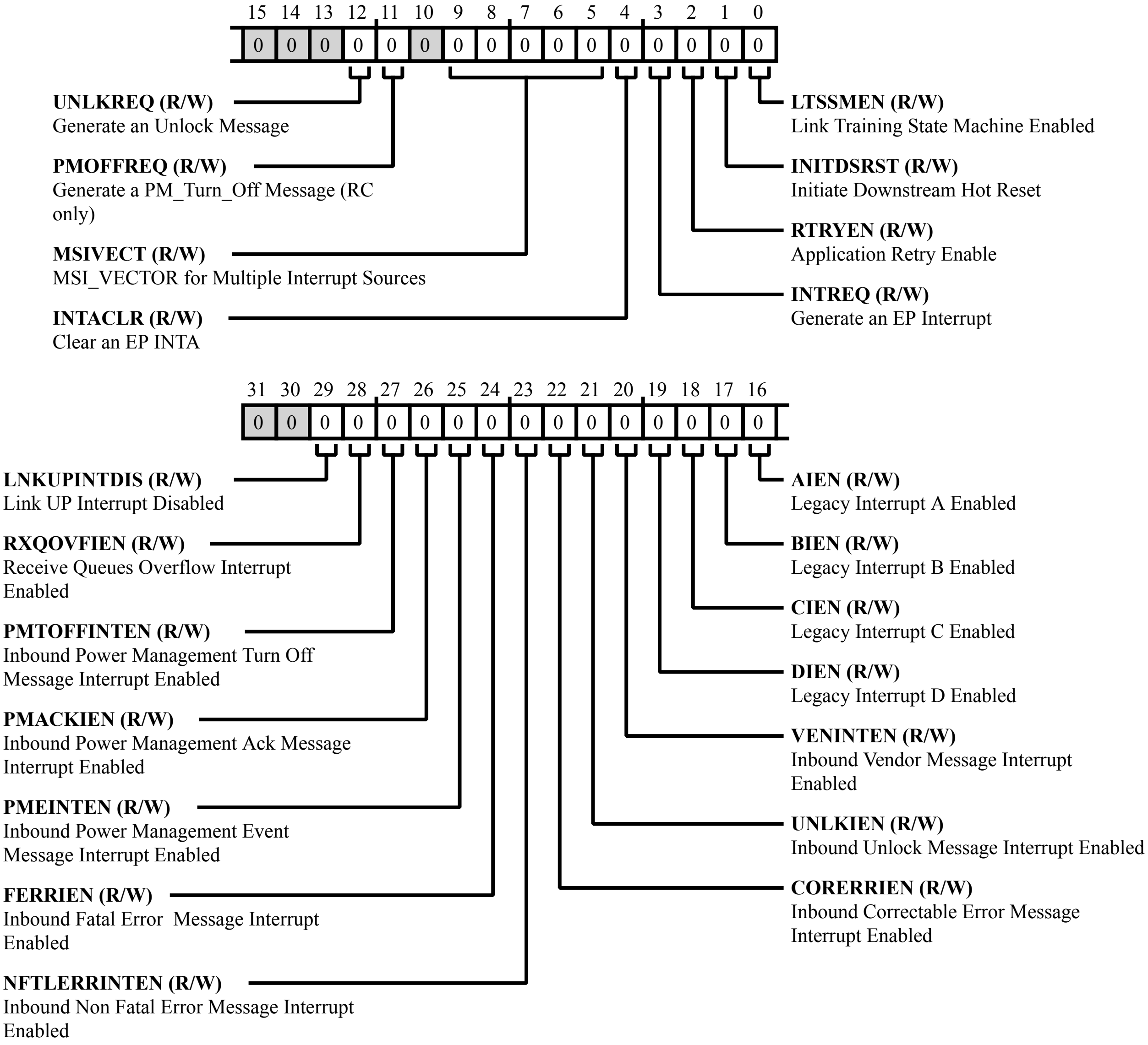

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
UNLKREQ (R/W)
Generate an Unlock Message
PMOFFREQ (R/W)
Generate a PM_Turn_Off Message (RC only)
MSIVECT (R/W)
MSI_VECTOR for Multiple Interrupt Sources
INTACLR (R/W)
Clear an EP INTA
LTSSMEN (R/W)
Link Training State Machine Enabled
INITDSRST (R/W)
Initiate Downstream Hot Reset
RTRYEN (R/W)
Application Retry Enable
INTREQ (R/W)
Generate an EP Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LNKUPINTDIS (R/W)
Link UP Interrupt Disabled
RXQOVFIEN (R/W)
Receive Queues Overflow Interrupt Enabled
PMTOFFINTEN (R/W)
Inbound Power Management Turn Off Message Interrupt Enabled
PMACKIEN (R/W)
Inbound Power Management Ack Message Interrupt Enabled
PMEINTEN (R/W)
Inbound Power Management Event Message Interrupt Enabled
FERRIEN (R/W)
Inbound Fatal Error Message Interrupt Enabled
NFTLERRINTEN (R/W)
Inbound Non Fatal Error Message Interrupt Enabled
AIEN (R/W)
Legacy Interrupt A Enabled
BIEN (R/W)
Legacy Interrupt B Enabled
CIEN (R/W)
Legacy Interrupt C Enabled
DIEN (R/W)
Legacy Interrupt D Enabled
VENINTEN (R/W)
Inbound Vendor Message Interrupt Enabled
UNLKIEN (R/W)
Inbound Unlock Message Interrupt Enabled
CORERRIEN (R/W)
Inbound Correctable Error Message Interrupt Enabled

Figure 29-15: PCIE_APP_CTL Register Diagram

Table 29-24: PCIE_APP_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	LNKUPINTDIS	Link UP Interrupt Disabled.	
		0	Interrupt Enabled
		1	Interrupt Disabled
28 (R/W)	RXQOVFIEN	Receive Queues Overflow Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
27 (R/W)	PMTOFFINTEN	Inbound Power Management Turn Off Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
26 (R/W)	PMACKIEN	Inbound Power Management Ack Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
25 (R/W)	PMEINTEN	Inbound Power Management Event Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
24 (R/W)	FERRIEN	Inbound Fatal Error Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
23 (R/W)	NFTLERRINTEN	Inbound Non Fatal Error Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
22 (R/W)	CORERRIEN	Inbound Correctable Error Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
21 (R/W)	UNLKIEN	Inbound Unlock Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
20 (R/W)	VENINTEN	Inbound Vendor Message Interrupt Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled

Table 29-24: PCIE_APP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	DIEN	Legacy Interrupt D Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
18 (R/W)	CIEN	Legacy Interrupt C Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
17 (R/W)	BIEN	Legacy Interrupt B Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
16 (R/W)	AIEN	Legacy Interrupt A Enabled.	
		0	Interrupt Disabled
		1	Interrupt Enabled
12 (R/W)	UNLKREQ	Generate an Unlock Message.	
11 (R/W)	PMOFFREQ	Generate a PM_Turn_Off Message (RC only).	
9:5 (R/W)	MSIVECT	MSI_VECTOR for Multiple Interrupt Sources.	
4 (R/W)	INTACLR	Clear an EP INTA.	
3 (R/W)	INTREQ	Generate an EP Interrupt.	
2 (R/W)	RTRYEN	Application Retry Enable. The `PCIE_APP_CTL.RTRYEN` bit determines whether the core completes incoming configuration requests with or without a configuration request retry status.	
		0	The core completes incoming configuration requests without a configuration request retry status.
		1	The core completes incoming configuration requests with a configuration request retry status.
1 (R/W)	INITDSRST	Initiate Downstream Hot Reset.	

Table 29-24: PCIE_APP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	LTSSMEN	Link Training State Machine Enabled. The `PCIE_APP_CTL.LTSSMEN` bit reports the state of the link training state machine.	
		0	The LTSSM is held in the Detect state
		1	Core Configuration Registers are initialized

Diagnostic Control Register

The `PCIE_APP_DIAGCTL` register provides CRC diagnostic message control.

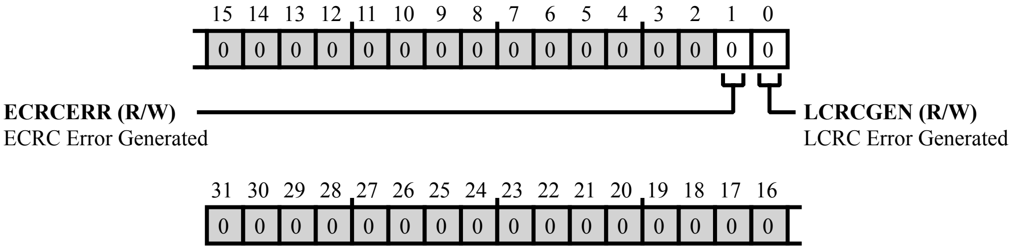

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ECRCERR (R/W)
ECRC Error Generated
LCRCGEN (R/W)
LCRC Error Generated
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-16: PCIE_APP_DIAGCTL Register Diagram

Table 29-25: PCIE_APP_DIAGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	ECRCERR	ECRC Error Generated. The `PCIE_APP_DIAGCTL.ECRCERR` bit enables ECRC error generation.	
		0	Interrupt not enabled
		1	Interrupt enabled
0 (R/W)	LCRCGEN	LCRC Error Generated. The `PCIE_APP_DIAGCTL.LCRCGEN` bit enables the LCRC error.	
		0	Interrupt not enabled
		1	Interrupt enabled

Diagnostic Status Register

The `PCIE_APP_DIAGSTAT` register reports LCRC and ECRC status.

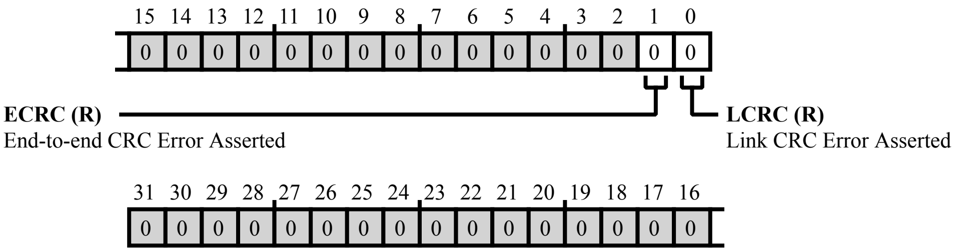

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ECRC (R)
End-to-end CRC Error Asserted
LCRC (R)
Link CRC Error Asserted
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-17: PCIE_APP_DIAGSTAT Register Diagram

Table 29-26: PCIE_APP_DIAGSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	ECRC	End-to-end CRC Error Asserted. The `PCIE_APP_DIAGSTAT.ECRC` bit reports the status of the ECRC.	
		0	End-to-end CRC is not corrupted
		1	End-to-end CRC Error injected by writing 1 to DBGCTL.LCRCERR
0 (R/NW)	LCRC	Link CRC Error Asserted. The `PCIE_APP_DIAGSTAT.LCRC` bit reports the status of the LCRC.	
		0	Link CRC is not corrupted
		1	Link CRC Error injected by writing 1 to DBGCTL.LCRCERR

Fatal Error Message Requester ID Register

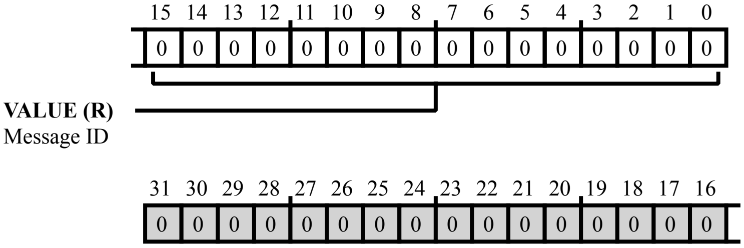

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-18: PCIE_APP_FERR_MSG Register Diagram

Table 29-27: PCIE_APP_FERR_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID.

Application Interrupts Status Register

The `PCIE_APP_INTSTAT` register reports the status of various interrupts and messages.

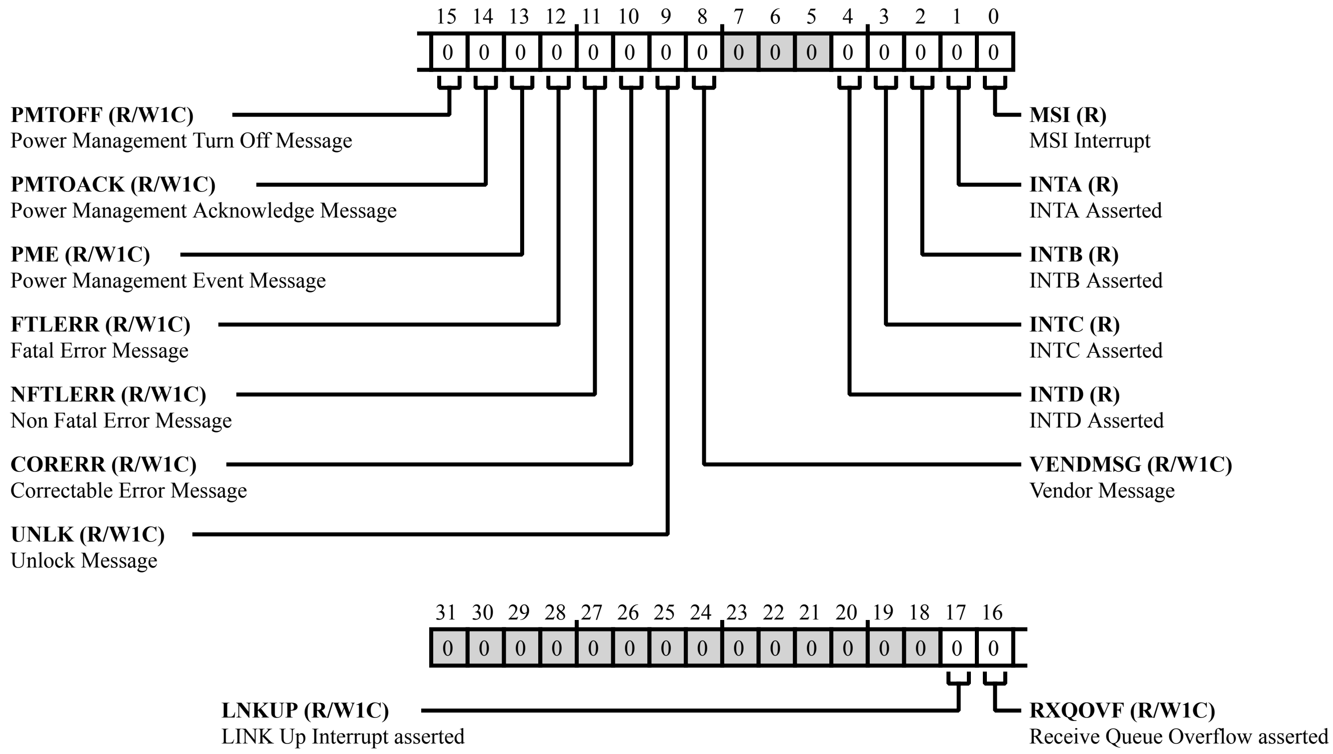

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PMTOFF (R/W1C)
Power Management Turn Off Message
PMTOACK (R/W1C)
Power Management Acknowledge Message
PME (R/W1C)
Power Management Event Message
FTLERR (R/W1C)
Fatal Error Message
NFTLERR (R/W1C)
Non Fatal Error Message
CORERR (R/W1C)
Correctable Error Message
UNLK (R/W1C)
Unlock Message
MSI (R)
MSI Interrupt
INTA (R)
INTA Asserted
INTB (R)
INTB Asserted
INTC (R)
INTC Asserted
INTD (R)
INTD Asserted
VENDMSG (R/W1C)
Vendor Message
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LNKUP (R/W1C)
LINK Up Interrupt asserted
RXQOVF (R/W1C)
Receive Queue Overflow asserted

Figure 29-19: PCIE_APP_INTSTAT Register Diagram

Table 29-28: PCIE_APP_INTSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	LNKUP	LINK Up Interrupt asserted. The `PCIE_APP_INTSTAT.LNKUP` bit reports the status of the link up interrupt.	
		0	Link Up Interrupt was not asserted since the last time this bit was cleared
		1	Link Up Interrupt was asserted since the last time this bit cleared

Table 29-28: PCIE_APP_INTSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	RXQOVF	Receive Queue Overflow asserted. The `PCIE_APP_INTSTAT.RXQOVF` bit reports the status of the receive queue overflow interrupt.	
		0	Receive Queue Overflow Interrupt was not asserted since the last time this bit was cleared
		1	Receive Queue Overflow Interrupt was not asserted since the last time this bit was cleared
15 (R/W1C)	PMTOFF	Power Management Turn Off Message. The `PCIE_APP_INTSTAT.PMTOFF` bit reports the status of the power management turn off message.	
		0	No Message
		1	Message Received
14 (R/W1C)	PMTOACK	Power Management Acknowledge Message. The `PCIE_APP_INTSTAT.PMTOACK` bit reports the status of the power management acknowledge message.	
		0	No Message
		1	Message Received
13 (R/W1C)	PME	Power Management Event Message. The `PCIE_APP_INTSTAT.PME` bit reports the status of the power management event message.	
		0	No Message
		1	Message Received
12 (R/W1C)	FTLERR	Fatal Error Message. The `PCIE_APP_INTSTAT.FTLERR` bit reports the status of the fatal error message.	
		0	No Message
		1	Message Received
11 (R/W1C)	NFTLERR	Non Fatal Error Message. The `PCIE_APP_INTSTAT.NFTLERR` bit reports the status of the non-fatal error message.	
		0	No Message
		1	Message Received

Table 29-28: PCIE_APP_INTSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	CORERR	Correctable Error Message. The `PCIE_APP_INTSTAT.CORERR` bit reports the status of the correctable error message.	
		0	No Message
		1	Message Received
9 (R/W1C)	UNLK	Unlock Message. The `PCIE_APP_INTSTAT.UNLK` bit reports the status of the unlock message.	
		0	No Message
		1	Message Received
8 (R/W1C)	VENDMSG	Vendor Message. The `PCIE_APP_INTSTAT.VENDMSG` bit reports the status of the vendor message.	
		0	No Message
		1	Message Received
4 (R/NW)	INTD	INTD Asserted. The `PCIE_APP_INTSTAT.INTD` bit reports the status of the legacy D interrupt.	
		0	INTD De-asserted
		1	INTD Asserted
3 (R/NW)	INTC	INTC Asserted. The `PCIE_APP_INTSTAT.INTC` bit reports the status of the legacy C interrupt.	
		0	INTC De-asserted
		1	INTC Asserted
2 (R/NW)	INTB	INTB Asserted. The `PCIE_APP_INTSTAT.INTB` bit reports the status of the legacy B interrupt.	
		0	INTB De-asserted
		1	INTB Asserted
1 (R/NW)	INTA	INTA Asserted. The `PCIE_APP_INTSTAT.INTA` bit reports the status of the legacy A interrupt.	
		0	INTA De-asserted
		1	INTA Asserted

Table 29-28: PCIE_APP_INTSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	MSI	MSI Interrupt. The `PCIE_APP_INTSTAT.MSI` bit reports the status of the message signaled interrupt.	
		0	MSI Interrupt Not Asserted
		1	MSI Interrupt Asserted

Non Fatal Error Message Requester ID Register

The `PCIE_APP_NFERR_MSG` register contains the requester ID of the latest received Non Fatal Error Message. It is advised to read this register when the `PCIE_APP_INTSTAT.NFTLERR` bit in APP_INTSTAT register is set.

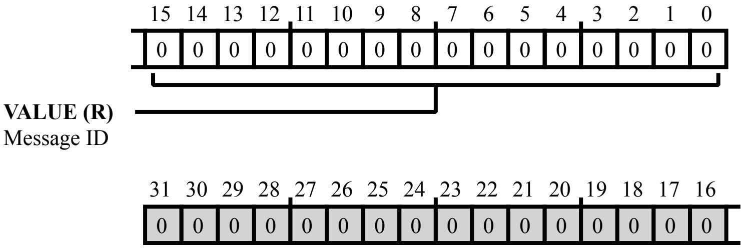

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-20: PCIE_APP_NFERR_MSG Register Diagram

Table 29-29: PCIE_APP_NFERR_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID. The `PCIE_APP_NFERR_MSG.VALUE` bit field contains the requester ID of the latest received Non Fatal Error Message.

Power Management Acknowledge Message Requester ID Register

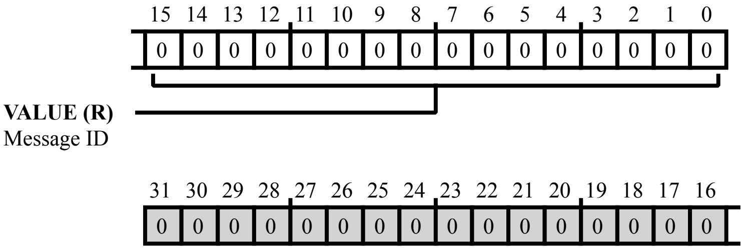

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-21: PCIE_APP_PMACK_MSG Register Diagram

Table 29-30: PCIE_APP_PMACK_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID.

Power Management Event Message Requester ID Register

The PCIE_APP_PMEVT_MSG register contains the requester ID of the latest received Power Management Event Message. It is advised to read this register when the PME bit in PCIE_APP_INTSTAT.PME register is set.

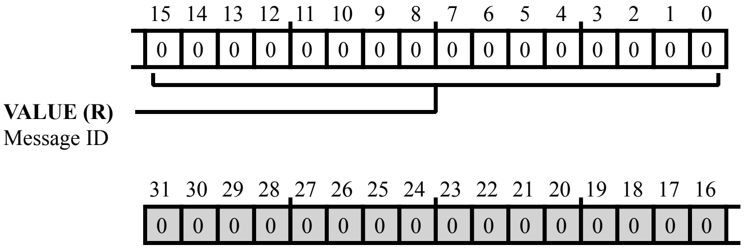

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-22: PCIE_APP_PMEVT_MSG Register Diagram

Table 29-31: PCIE_APP_PMEVT_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID. The PCIE_APP_PMEVT_MSG.VALUE bit field contains the requester ID of the latest received Power Management Event Message.

Power Management Turn Off Message Requester ID Register

The `PCIE_APP_PMTOFF_MSG` register contains the requester ID of the latest received Power Management Turn Off Message. It is advised to read this register when the `PCIE_APP_INTSTAT.PMTOFF` bit is set.

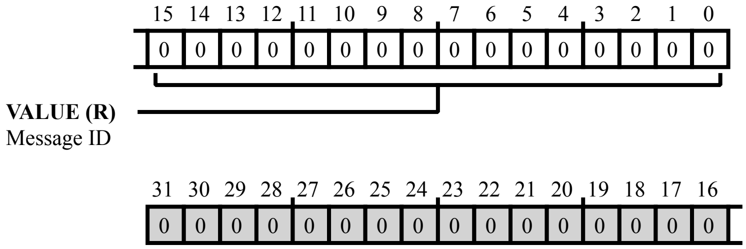

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-23: PCIE_APP_PMTOFF_MSG Register Diagram

Table 29-32: PCIE_APP_PMTOFF_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID. The `PCIE_APP_PMTOFF_MSG.VALUE` bit field contains the requester ID of the latest received Power Management Turn Off Message.

Application Status Register

The `PCIE_APP_STAT` register provides information about various application states.

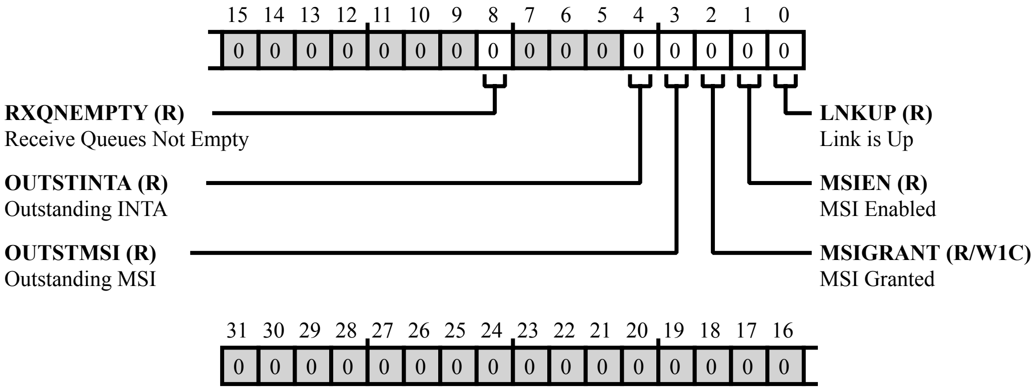

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXQNEMPTY (R)
Receive Queues Not Empty
OUTSTINTA (R)
Outstanding INTA
OUTSTMSI (R)
Outstanding MSI
LNKUP (R)
Link is Up
MSIEN (R)
MSI Enabled
MSIGRANT (R/W1C)
MSI Granted
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-24: PCIE_APP_STAT Register Diagram

Table 29-33: PCIE_APP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RXQNEMPTY	Receive Queues Not Empty.	
		0	Receive Queues Empty
		1	Receive Queues Not Empty
4 (R/NW)	OUTSTINTA	Outstanding INTA.	
		0	No outstanding INTA Request
		1	Outstanding INTA Request
3 (R/NW)	OUTSTMSI	Outstanding MSI.	
		0	No outstanding MSI Request
		1	Outstanding MSI Request
2 (R/W1C)	MSIGRANT	MSI Granted.	
		0	The PCIe_DM_CORE has not accepted the request to send an MSI.
		1	The PCIe_DM_CORE has accepted the request to send an MSI.
1 (R/NW)	MSIEN	MSI Enabled.	
		0	MSI is Disabled / INTx message may be sent
		1	MSI is Enabled / INTx message is not sent

Table 29-33: PCIE_APP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	LNKUP	Link is Up.	
		0	Link is Down
		1	Link is Up

Unlock Message Requester ID Register

The `PCIE_APP_UNLK_MSG` register contains the ID of the Requestor to generate an Unlock message.

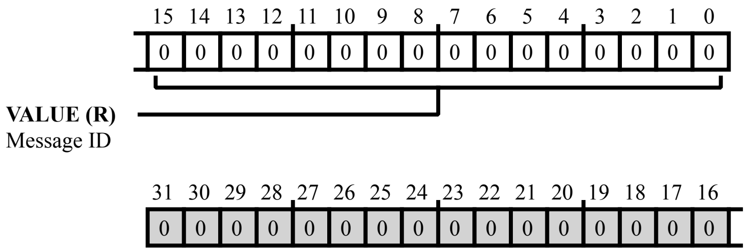

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-25: PCIE_APP_UNLK_MSG Register Diagram

Table 29-34: PCIE_APP_UNLK_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID.

Vendor Message Requester ID Register

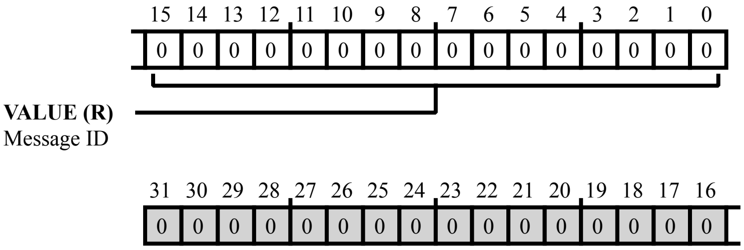

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Message ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-26: PCIE_APP_VNDR_MSG Register Diagram

Table 29-35: PCIE_APP_VNDR_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Message ID.

Vendor Message Header Bytes 8 to 11 Register

The `PCIE_APP_VNDR_MSG_HDR0` register contains the 4th double word of the latest received Vendor Message header. It is advised to read this register when the `PCIE_APP_INTSTAT.VENDMSG` bit is set.

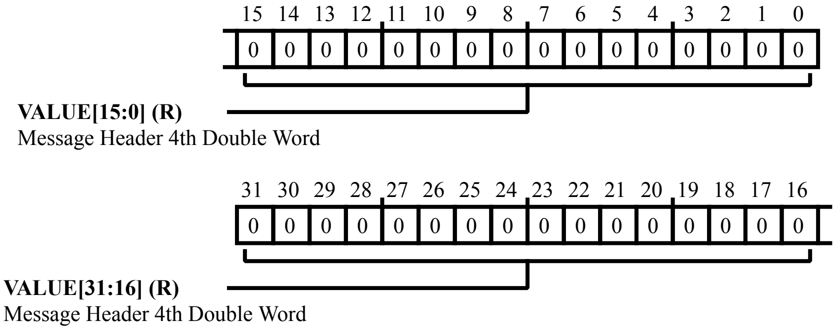

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Message Header 4th Double Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Message Header 4th Double Word

Figure 29-27: PCIE_APP_VNDR_MSG_HDR0 Register Diagram

Table 29-36: PCIE_APP_VNDR_MSG_HDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Message Header 4th Double Word. The `PCIE_APP_VNDR_MSG_HDR0.VALUE` bit field contains the 4th double word of the latest received Vendor Message header.

Vendor Message Header Bytes 12 to 15 Register

The `PCIE_APP_VNDR_MSG_HDR1` register contains the 5th double word of the latest received Vendor Message header. It is advised to read this register when the `PCIE_APP_INTSTAT.VENDMSG` bit is set.

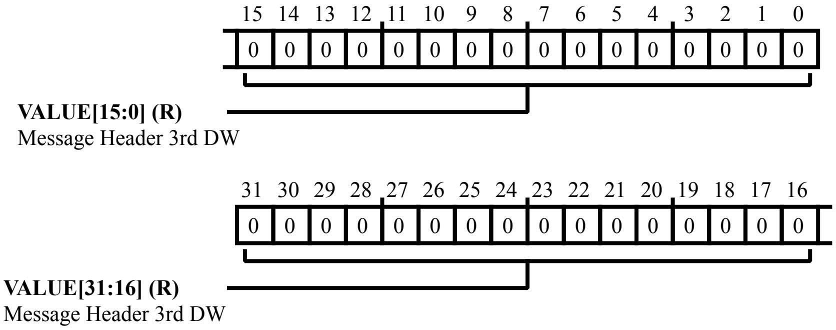

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Message Header 3rd DW
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Message Header 3rd DW

Figure 29-28: PCIE_APP_VNDR_MSG_HDR1 Register Diagram

Table 29-37: PCIE_APP_VNDR_MSG_HDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Message Header 3rd DW.

Auxiliary Clock Frequency Control Register

The PCIE_AUX_CLKFREQ_[n] register is used to provide a 1 us reference for counting time during low-power states with aux_clk when the PHY has removed the pipe_clk.

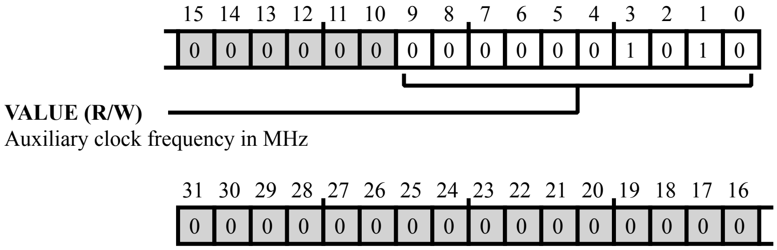

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
VALUE (R/W)
Auxiliary clock frequency in MHz
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-29: PCIE_AUX_CLKFREQ_[n] Register Diagram

Table 29-38: PCIE_AUX_CLKFREQ_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Auxiliary clock frequency in MHz. The `PCIE_AUX_CLKFREQ_[n].VALUE` bit field is used to provide a 1 us reference for counting time during low-power states with aux_clk when the PHY has removed the pipe_clk. Frequencies lower than 1 MHz are possible but with a loss of accuracy in the time counted. If the actual frequency (f) of aux_clk does not exactly match the programmed frequency (f_prog), then an error in the time counted by the core that can be expressed in percentage as: err% = (f_prog/f-1)*100. For example if f=2.5 MHz and f_prog=3 MHz, then err% =(3/2.5-1)*100 =20%, meaning that the time counted by the core on aux_clk will be 20% greater than the time in us programmed in the corresponding time register (for example T_POWER_ON). This bit field is sticky.

Peripheral and SCB Bridge Slave Error Response Register

The `PCIE_BERR_RESP_[n]` register controls the mapping of errors for received inbound completions (of non-posted requests), and the Slave Port response to the SCB master.

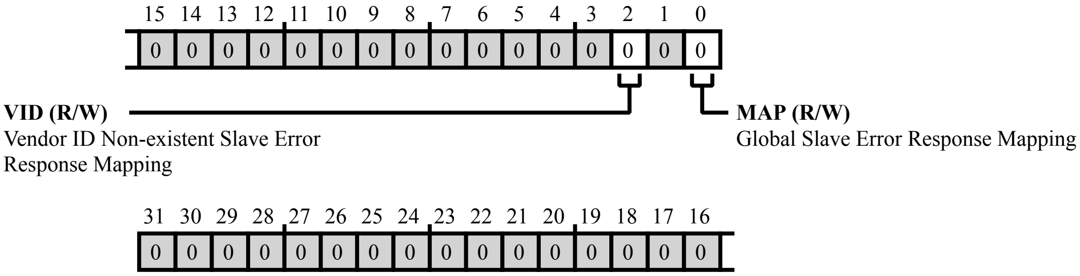

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VID (R/W)
Vendor ID Non-existent Slave Error Response Mapping
MAP (R/W)
Global Slave Error Response Mapping
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-30: PCIE_BERR_RESP_[n] Register Diagram

Table 29-39: PCIE_BERR_RESP_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	VID	Vendor ID Non-existent Slave Error Response Mapping. Vendor ID Non-existent Slave Error Response Mapping. The `PCIE_BERR_RESP_[n].VID` bit determines the peripheral/SCB slave response for errors on reads to non-existent Vendor ID register. The core ignores the setting in the bit when bit 0 of this register is 0. Note: This register field is sticky.	
		0	OKAY (with FFFF data for non-posted requests)
		1	Error
0 (R/W)	MAP	Global Slave Error Response Mapping. Global Slave Error Response Mapping. The `PCIE_BERR_RESP_[n].MAP` bit determines the peripheral/SCB slave response for all error scenarios on non-posted requests. For more details see "PCIe Link Error Mapping to SCB Slave Interface." Note: This register field is sticky.	
		0	OKAY (with FFFF data for non-posted requests) and ignore the settings in bit [3:2] of this register
		1	ERROR for normal link (data) accesses and Error for normal link data accesses and look at bits [3:2] for other scenarios

Bus Order Manager Watchdog Off Register

The `PCIE_BUSMGR_WDOG_[n]` register contains the initial load value for the bus ordering manager watchdog timer. It has a resolution of 2 us.

The bridge dequeue controller checks (per VC) that there are no outstanding posted transactions on the bus master interface before it takes a completion from the native core receive queue; to prevent violation of rule (5). If necessary it will halt the completion queue and start the watchdog timer.

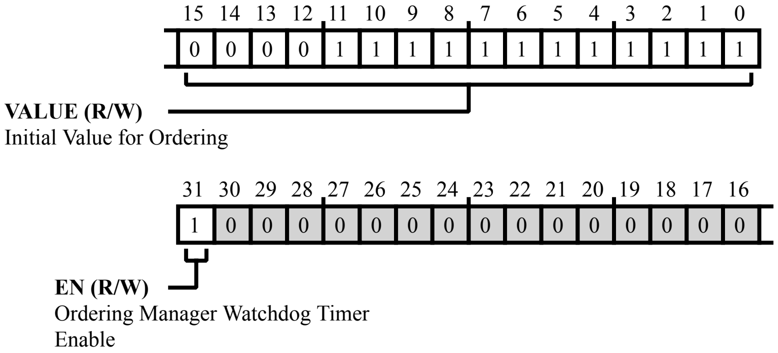

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
VALUE (R/W)
Initial Value for Ordering
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
Ordering Manager Watchdog Timer Enable

Figure 29-31: PCIE_BUSMGR_WDOG_[n] Register Diagram

Table 29-40: PCIE_BUSMGR_WDOG_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	EN	Ordering Manager Watchdog Timer Enable. The `PCIE_BUSMGR_WDOG_[n].EN` bit enables (if set =1) the bus ordering manager watchdog timer.
15:0 (R/W)	VALUE	Initial Value for Ordering. The `PCIE_BUSMGR_WDOG_[n].VALUE` bit field contains the initial load value for the bus ordering manager watchdog timer. It has a resolution of 2 us. Valid write range values are 1 = 2us, 2 = 4us, 3 = 6us ... 65536 = ~131ms.

Bus Multiple Outbound Decomposition SubReq Control Register

The `PCIE_BUS_MULOB_DECOMP_[n]` register controls whether multiple outstanding non-posted requests that were derived from decomposition of an outbound memory bus request can occur.

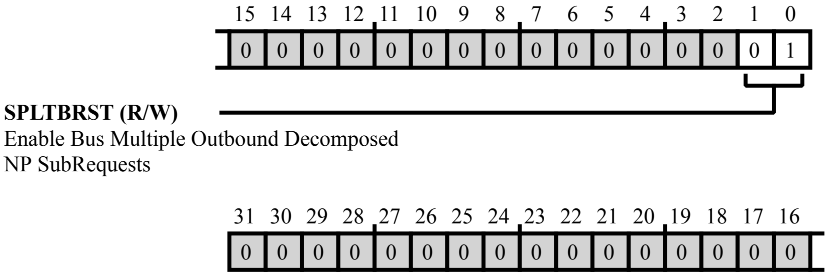

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
SPLTBRST (R/W)
Enable Bus Multiple Outbound Decomposed NP SubRequests
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-32: PCIE_BUS_MULOB_DECOMP_[n] Register Diagram

Table 29-41: PCIE_BUS_MULOB_DECOMP_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	SPLTBRST	Enable Bus Multiple Outbound Decomposed NP SubRequests. When the `PCIE_BUS_MULOB_DECOMP_[n].SPLTBRST` bit is cleared (=0), the possibility of having multiple outstanding non-posted requests that were derived from decomposition of an outbound memory bus request is disabled. You should not clear this register unless your application master is requesting an amount of read data greater than Max_Read_Request_Size, and the remote device (or switch) is reordering completions that have different tags <br><i>Note</i>: This register field is sticky.

ACE Cache Coherency Control Register 1

The `PCIE_CACHE_CTL1_[n]` register sets the memory type for the lower and upper parts of the address space and contains bits [31:0] of dword-aligned address of the boundary for memory type.

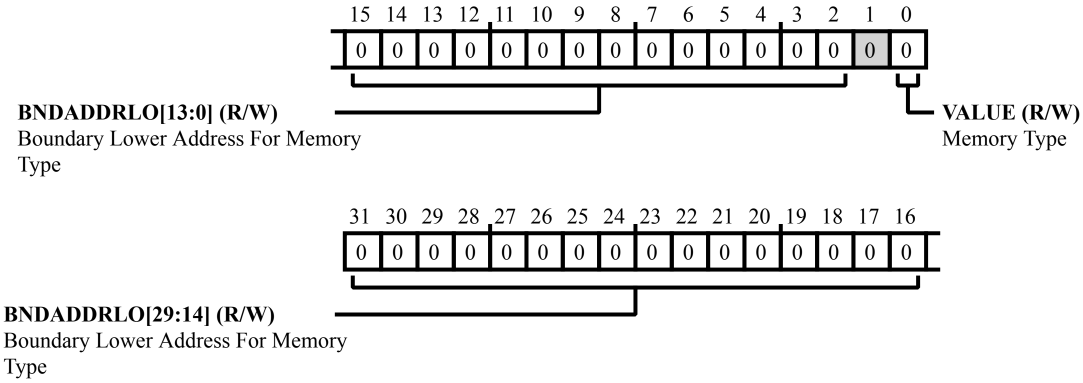

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BNDADDRLO[13:0] (R/W)
Boundary Lower Address For Memory Type
VALUE (R/W)
Memory Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BNDADDRLO[29:14] (R/W)
Boundary Lower Address For Memory Type

Figure 29-33: PCIE_CACHE_CTL1_[n] Register Diagram

Table 29-42: PCIE_CACHE_CTL1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:2 (R/W)	BNDADDRLO	Boundary Lower Address For Memory Type. The `PCIE_CACHE_CTL1_[n].BNDADDRLO` bit field contains bits [31:0] of dword-aligned address of the boundary for Memory type. The two lower address LSBs are 0. Addresses up to but not including this value are in the lower address space region; addresses equal or greater than this value are in the upper address space region. This `PCIE_CACHE_CTL1_[n].BNDADDRLO` bit field is sticky.	
0 (R/W)	VALUE	Memory Type. The `PCIE_CACHE_CTL1_[n].VALUE` bit sets the memory type for the lower and upper parts of the address space. The `PCIE_CACHE_CTL1_[n].VALUE` bit is sticky.	
		0	lower = Peripheral, upper = Memory
		1	lower = Memory type, upper = Peripheral

ACE Cache Coherency Control Register 3

The `PCIE_CACHE_CTL3_[n]` register defines cache signal behavior and identifies cache signal values.

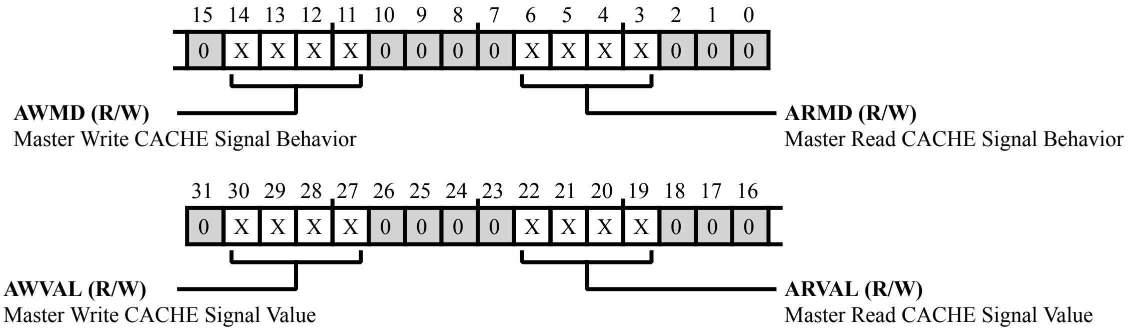

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 X X X X 0 0 0 0 X X X X 0 0 0
AWMD (R/W)
Master Write CACHE Signal Behavior
ARMD (R/W)
Master Read CACHE Signal Behavior
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 X X X X 0 0 0 0 X X X X 0 0 0
AWVAL (R/W)
Master Write CACHE Signal Value
ARVAL (R/W)
Master Read CACHE Signal Value

Figure 29-34: PCIE_CACHE_CTL3_[n] Register Diagram

Table 29-43: PCIE_CACHE_CTL3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30:27 (R/W)	AWVAL	Master Write CACHE Signal Value. The `PCIE_CACHE_CTL3_[n].AWVAL` bit field is the value of the individual bits in mstr_awcache when CFG_MSTR_AWCACHE_MODE is 1. This is not applicable to message requests; for message requests the value of mstr_awcache is always 0000. The `PCIE_CACHE_CTL3_[n].AWVAL` bit field is sticky.	
22:19 (R/W)	ARVAL	Master Read CACHE Signal Value. The `PCIE_CACHE_CTL3_[n].ARVAL` bit field is the value of the individual bits in mstr_arcache when CFG_MSTR_ARCACHE_MODE is 1. The `PCIE_CACHE_CTL3_[n].ARVAL` bit field is sticky.	
14:11 (R/W)	AWMD	Master Write CACHE Signal Behavior. The `PCIE_CACHE_CTL3_[n].AWMD` bit field defines how the individual bits in mstr_awcache are controlled. For message requests the value of mstr_awcache is always 0000 regardless of the value of this bit field. The `PCIE_CACHE_CTL3_[n].AWMD` bit field is sticky.	
		0	Set automatically by the SCB master
		1	Set automatically by the SCB master
6:3 (R/W)	ARMD	Master Read CACHE Signal Behavior. The `PCIE_CACHE_CTL3_[n].ARMD` bit field defines how the individual bits in mstr_arcache are controlled. The `PCIE_CACHE_CTL3_[n].ARMD` bit field is sticky.	
		0	Set automatically by the SCB master
		1	Set by the value of the corresponding bit of the CFG_MSTR_ARCACHE_VALUE field

PCIe Capabilities, ID, Next Pointer Register

The `PCIE_CAPB_NPTR_[n]` register configures PCI Express capabilities, identifies the capability structure version number, identifies which MSI/MSI-X vector is used for the interrupt message, and the offset to the next PCI Capability structure Next Pointer.

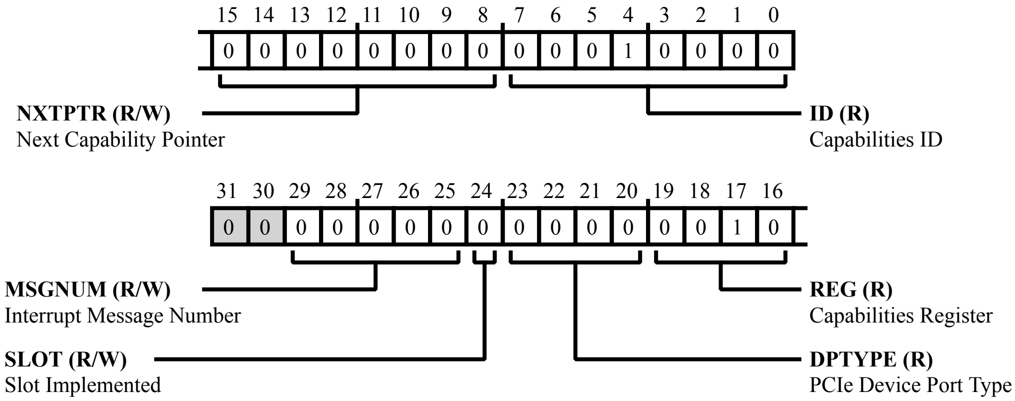

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
NXTPTR (R/W)
Next Capability Pointer
ID (R)
Capabilities ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
MSGNUM (R/W)
Interrupt Message Number
SLOT (R/W)
Slot Implemented
REG (R)
Capabilities Register
DPTYPE (R)
PCIe Device Port Type

Figure 29-35: PCIE_CAPB_NPTR_[n] Register Diagram

Table 29-44: PCIE_CAPB_NPTR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	MSGNUM	Interrupt Message Number. The `PCIE_CAPB_NPTR_[n].MSGNUM` bit field indicates which MSI/MSI-X vector is used for the interrupt message generated in association with any of the status bits of this Capability structure. For MSI, the value in this field indicates the offset between the base Message Data and the interrupt message that is generated. Hardware is required to update this field so that it is correct if the number of MSI Messages assigned to the Function changes when software writes to the Multiple Message Enable field in the MSI Message Control register. For MSI-X, the value in this field indicates which MSI-X Table entry is used to generate the interrupt message. The entry must be one of the first 32 entries even if the Function implements more than 32 entries. For a given MSI-X implementation, the entry must remain constant. If both MSI and MSI-X are implemented, they are permitted to use different vectors, though software is permitted to enable only one mechanism at a time. If MSI-X is enabled, the value in this field must indicate the vector for MSI-X. If MSI is enabled or neither is enabled, the value in this field must indicate the vector for MSI. If software enables both MSI and MSI-X at the same time, the value in this field is undefined.

Table 29-44: PCIE_CAPB_NPTR_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>24
(R/W)</td><td>SLOT</td><td colspan="2">Slot Implemented.
When Set, the PCIE_CAPB_NPTR_[n].SLOT bit indicates that the Link associated with this Port is connected to a slot (as compared to being connected to a system-integrated device or being disabled). This bit is valid for Downstream Ports. This bit is undefined for Upstream Ports.</td></tr>
<tr><td rowspan="10">23:20
(R/NW)</td><td rowspan="10">DPTYPE</td><td colspan="2">PCIe Device Port Type.
The PCIE_CAPB_NPTR_[n].DPTYPE bit field indicates the specific type of this PCI Express Function. Note that different Functions in a multi-Function device can generally be of different types. All encodings not listed are Reserved. *This value is only valid for Functions that implement a Type 0x1 PCI Configuration Space header.</td></tr>
<tr><td>0</td><td>PCI Express Endpoint</td></tr>
<tr><td>1</td><td>Legacy PCI Express Endpoint</td></tr>
<tr><td>2</td><td>Root Port of PCI Express Root Complex*</td></tr>
<tr><td>3</td><td>Upstream Port of PCI Express Switch*</td></tr>
<tr><td>4</td><td>Downstream Port of PCI Express Switch*</td></tr>
<tr><td>5</td><td>PCI Express to PCI/PCI-X Bridge*</td></tr>
<tr><td>6</td><td>PCI/PCI-X to PCI Express Bridge*</td></tr>
<tr><td>7</td><td>Root Complex Integrated Endpoint</td></tr>
<tr><td>8</td><td>Root Complex Event Collector</td></tr>
<tr><td>19:16
(R/NW)</td><td>REG</td><td colspan="2">Capabilities Register.
The PCIE_CAPB_NPTR_[n].REG bit field is read-only field must be set to 0x2.</td></tr>
<tr><td>15:8
(R/W)</td><td>NXTPTR</td><td colspan="2">Next Capability Pointer.
The PCIE_CAPB_NPTR_[n].NXTPTR bit field contains the offset to the next PCI Capability structure or 0x0 if no other items exist in the linked list of Capabilities.</td></tr>
<tr><td>7:0
(R/NW)</td><td>ID</td><td colspan="2">Capabilities ID.
The PCIE_CAPB_NPTR_[n].ID read-only bit indicates PCI-SIG defined PCI Express Capability structure version number. A version of the specification that changes the PCI Express Capability structure in a way that is not otherwise identifiable (e.g., through a new Capability field) is permitted to increment this field. All such changes to the PCI Express Capability structure must be software-compatible.
Software must check for Capability Version numbers that are greater than or equal to the highest number defined when the software is written, as Functions reporting any such Capability Version numbers will contain a PCI Express Capability structure that is compatible with that piece of software. Must be hardwired to 0x2 for Functions compliant to this specification.</td></tr>
</table>

Correctable Error Mask Register

The `PCIE_CORERR_MSK_[n]` register controls reporting of individual correctable errors by this Function to the PCI Express Root Complex via a PCI Express error Message. A masked error (respective bit Set in the mask register) is not reported to the PCI Express Root Complex by this Function. There is a mask bit per error bit in the Correctable Error Status register. Register fields for bits not implemented by the Function are set to 0.

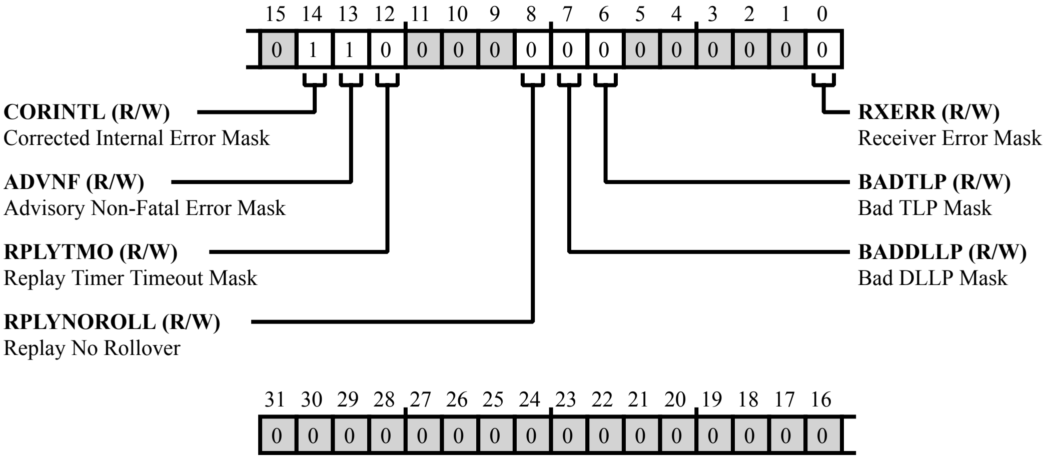

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
CORINTL (R/W)
Corrected Internal Error Mask
ADVNF (R/W)
Advisory Non-Fatal Error Mask
RPLYTMO (R/W)
Replay Timer Timeout Mask
RPLYNOROLL (R/W)
Replay No Rollover
RXERR (R/W)
Receiver Error Mask
BADTLP (R/W)
Bad TLP Mask
BADDLLP (R/W)
Bad DLLP Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-36: PCIE_CORERR_MSK_[n] Register Diagram

Table 29-45: PCIE_CORERR_MSK_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	CORINTL	Corrected Internal Error Mask.
13 (R/W)	ADVNF	Advisory Non-Fatal Error Mask. The `PCIE_CORERR_MSK_[n].ADVNF` bit is set by default to enable compatibility with software that does not comprehend Role-Based Error Reporting
12 (R/W)	RPLYTMO	Replay Timer Timeout Mask.
8 (R/W)	RPLYNOROLL	Replay No Rollover.
7 (R/W)	BADDLLP	Bad DLLP Mask.
6 (R/W)	BADTLP	Bad TLP Mask.
0 (R/W)	RXERR	Receiver Error Mask.

Correctable Error Status Register

The `PCIE_CORERR_STAT_[n]` register reports error status of individual correctable error sources on a PCI Express device Function. When an individual error status bit is Set, it indicates that a particular error occurred; software may clear an error status by writing a 1 to the respective bit. Register bits not implemented by the Function are hardwired to 0.

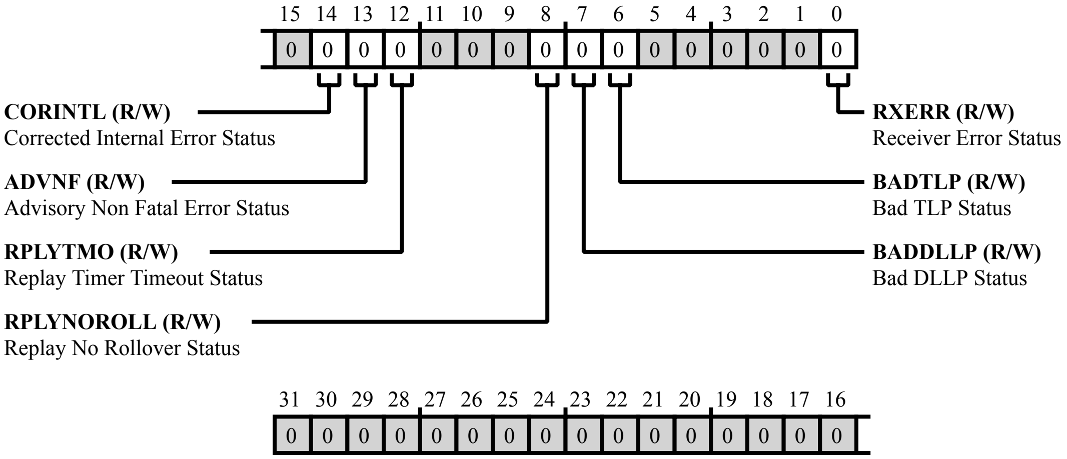

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CORINTL (R/W)
Corrected Internal Error Status
ADVNF (R/W)
Advisory Non Fatal Error Status
RPLYTMO (R/W)
Replay Timer Timeout Status
RPLYNOROLL (R/W)
Replay No Rollover Status
RXERR (R/W)
Receiver Error Status
BADTLP (R/W)
Bad TLP Status
BADDLLP (R/W)
Bad DLLP Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-37: PCIE_CORERR_STAT_[n] Register Diagram

Table 29-46: PCIE_CORERR_STAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	CORINTL	Corrected Internal Error Status.	
13 (R/W)	ADVNF	Advisory Non Fatal Error Status.	
12 (R/W)	RPLYTMO	Replay Timer Timeout Status.	
8 (R/W)	RPLYNOROLL	Replay No Rollover Status.	
7 (R/W)	BADDLLP	Bad DLLP Status.	
		0	No error
		1	DLLP status error occurred
6 (R/W)	BADTLP	Bad TLP Status.	
		0	No error
		1	TLP error occurred

Table 29-46: PCIE_CORERR_STAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	RXERR	Receiver Error Status.	
		0	No error
		1	Receive error occurred

Device Capabilities 2 Register

The `PCIE_DEV_CAPB2_[n]` register configures and reports the status of various Function types.

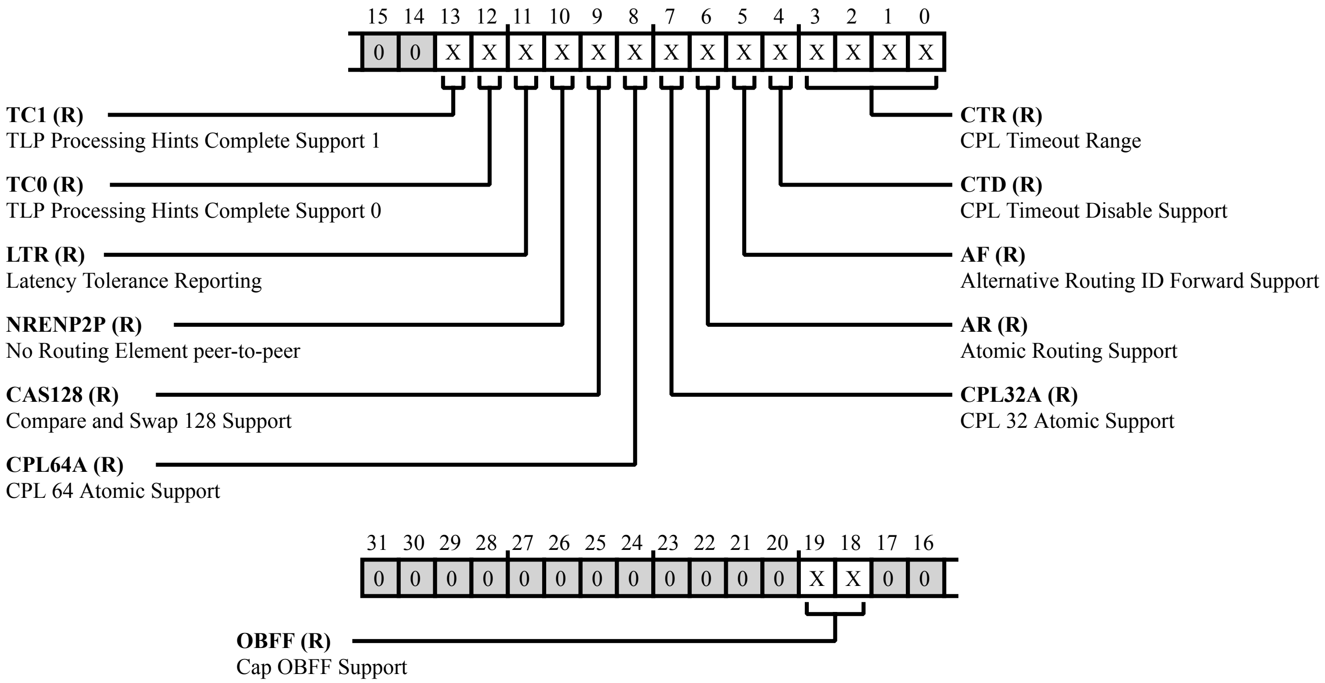

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 X X X X X X X X X X X X X X
TC1 (R)
TLP Processing Hints Complete Support 1
TC0 (R)
TLP Processing Hints Complete Support 0
LTR (R)
Latency Tolerance Reporting
NRENP2P (R)
No Routing Element peer-to-peer
CAS128 (R)
Compare and Swap 128 Support
CPL64A (R)
CPL 64 Atomic Support
CTR (R)
CPL Timeout Range
CTD (R)
CPL Timeout Disable Support
AF (R)
Alternative Routing ID Forward Support
AR (R)
Atomic Routing Support
CPL32A (R)
CPL 32 Atomic Support
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 X X 0 0
OBFF (R)
Cap OBFF Support

Figure 29-38: PCIE_DEV_CAPB2_[n] Register Diagram

Table 29-47: PCIE_DEV_CAPB2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:18 (R/NW)	OBFF	Cap OBFF Support. The `PCIE_DEV_CAPB2_[n].OBFF` bit field indicates if OBFF (Optimized Buffer Flush/Fill) is supported and, if so, what signaling mechanism is used. The value reported in this field must indicate support for WAKE# signaling only if: • For a Downstream Port, driving the WAKE# signal for OBFF is supported and the connector or component connected Downstream is known to receive that same WAKE# signal • For an Upstream Port, receiving the WAKE# signal for OBFF is supported and, if the component is on an add-in-card, that the component is connected to the WAKE# signal on the connector. Root Ports, Switch Ports, and Endpoints are permitted to implement this capability. For Bridges and Ports that do not implement this capability, this field must be set to 00.
		0 Not supported
		1 Supported using Message signaling only
		2 Supported using WAKE# signaling only
		3 Supported using WAKE# and Message signaling
13 (R/NW)	TC1	TLP Processing Hints Complete Support 1. The `PCIE_DEV_CAPB2_[n].TC1` bit indicates Completer support for TPH or Extended TPH. Applicable only to Root Ports and Endpoints. For all other Functions, this field is Reserved.
12 (R/NW)	TC0	TLP Processing Hints Complete Support 0. The `PCIE_DEV_CAPB2_[n].TC0` bit indicates Completer support for TPH or Extended TPH. Applicable only to Root Ports and Endpoints. For all other Functions, this field is Reserved.
11 (R/NW)	LTR	Latency Tolerance Reporting. The `PCIE_DEV_CAPB2_[n].LTR` bit indicates indicates support for the optional Latency Tolerance Reporting (LTR) mechanism. Root Ports, Switches and Endpoints are permitted to implement this capability. For Bridges and other Functions that do not implement this capability, this bit must be set to 0.
10 (R/NW)	NRENP2P	No Routing Element peer-to-peer. If the `PCIE_DEV_CAPB2_[n].NRENP2P` bit is set, the routing element never carries out the passing that is associated with the Relaxed Ordering Attribute field being Set. This bit applies only for Switches and RCs that support peer-to-peer traffic between Root Ports. The `PCIE_DEV_CAPB2_[n].NRENP2P` bit applies only to Posted Requests being forwarded through the Switch or Root-Complex and does not apply to traffic originating or terminating within the Switch or Root-Complex itself. All Ports on a Switch or Root-Complex must report the same value for this bit. For all other functions, this bit must be 0.

Table 29-47: PCIE_DEV_CAPB2_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	CAS128	Compare and Swap 128 Support. Applicable to Functions with Memory Space BARs as well as all Root Ports; must be 0 otherwise. The `PCIE_DEV_CAPB2_[n].CAS128` bit must be set to 1 if the Function supports this optional capability.
8 (R/NW)	CPL64A	CPL 64 Atomic Support. The `PCIE_DEV_CAPB2_[n].CPL64A` bit applies to Functions with Memory Space BARs as well as all Root Ports; must be 0 otherwise. Includes FetchAdd, Swap, and CAS AtomicOps. This bit must be set to 1 if the Function supports this optional capability.
7 (R/NW)	CPL32A	CPL 32 Atomic Support. The `PCIE_DEV_CAPB2_[n].CPL32A` bit applies to Functions with Memory Space BARs as well as all Root Ports; must be 0 otherwise. Includes FetchAdd, Swap, and CAS AtomicOps. This bit must be set to 1 if the Function supports this optional capability.
6 (R/NW)	AR	Atomic Routing Support. The `PCIE_DEV_CAPB2_[n].AR` bit applies only to Switch Upstream Ports, Switch Downstream Ports, and Root Ports; must be 0 for other Function types. This bit must be set to 1 if the Port supports this optional capability.
5 (R/NW)	AF	Alternative Routing ID Forward Support. The `PCIE_DEV_CAPB2_[n].AF` bit applies only to Switch Downstream Ports and Root Ports; must be 0 for other Function types. This bit must be set to 1 if a Switch Downstream Port or Root Port supports this optional capability.
4 (R/NW)	CTD	CPL Timeout Disable Support. A value of 1 in the `PCIE_DEV_CAPB2_[n].CTD` bit indicates support for the Completion Timeout Disable mechanism. The Completion Timeout Disable mechanism is required for Endpoints that issue Requests on their own behalf and PCI Express to PCI/PCI-X Bridges that take ownership of Requests issued on PCI Express. This mechanism is optional for Root Ports. For all other Functions this field is Reserved and must be set to 0.

Table 29-47: PCIE_DEV_CAPB2_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/NW)	CTR	CPL Timeout Range. The `PCIE_DEV_CAPB2_[n].CTR` bit field indicates device Function support for the optional Completion Timeout programmability mechanism. This mechanism allows system software to modify the Completion Timeout value. This field is applicable only to Root Ports, Endpoints that issue Requests on their own behalf, and PCI Express to PCI/PCI-X Bridges that take ownership of Requests issued on PCI Express. For all other Functions this field is Reserved and must be set to 0000. Four time value ranges are defined: • Range A: 50 s to 10 ms • Range B: 10 ms to 250 ms • Range C: 250 ms to 4 s • Range D: 4 s to 64 s All other values are Reserved. It is strongly recommended that the Completion Timeout mechanism not expire in less than 10 ms.	
		0	Completion Timeout programming not supported the Function must implement a timeout value in the range 50 s to 50 ms
		1	Range A
		2	Range B
		3	Ranges A and B
		4	Ranges B and C
		5	Ranges A, B and C
		6	Ranges B, C and D
		7	Ranges A, B, C and D

Device Control 2 and Status 2 Register

The `PCIE_DEV_CTLSTAT2_[n]` register configures the Completion Timeout mechanism and allows the Downstream Port to disable its traditional Device Number field.

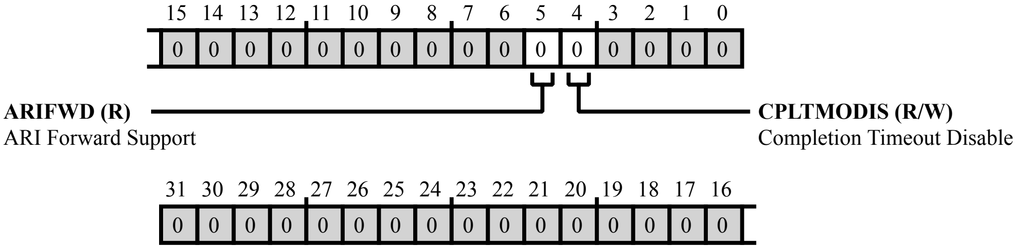

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARIFWD (R)
ARI Forward Support
CPLTMODIS (R/W)
Completion Timeout Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-39: PCIE_DEV_CTLSTAT2_[n] Register Diagram

Table 29-48: PCIE_DEV_CTLSTAT2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	ARIFWD	ARI Forward Support. When the `PCIE_DEV_CTLSTAT2_[n].ARIFWD` bit is set, the Downstream Port disables its traditional Device Number field = 0 enforcement when turning a Type 1 Configuration Request into a Type 0 Configuration Request. This permits access to Extended Functions in an ARI Device immediately below the Port. Must be set to 0 if the ARI Forwarding Supported bit is 0.
4 (R/W)	CPLTMODIS	Completion Timeout Disable. When set, the `PCIE_DEV_CTLSTAT2_[n].CPLTMODIS` bit disables the Completion Timeout mechanism.

Device Control and Status Register

The `PCIE_DEV_CTLSTAT_[n]` register controls PCI Express device specific parameters.

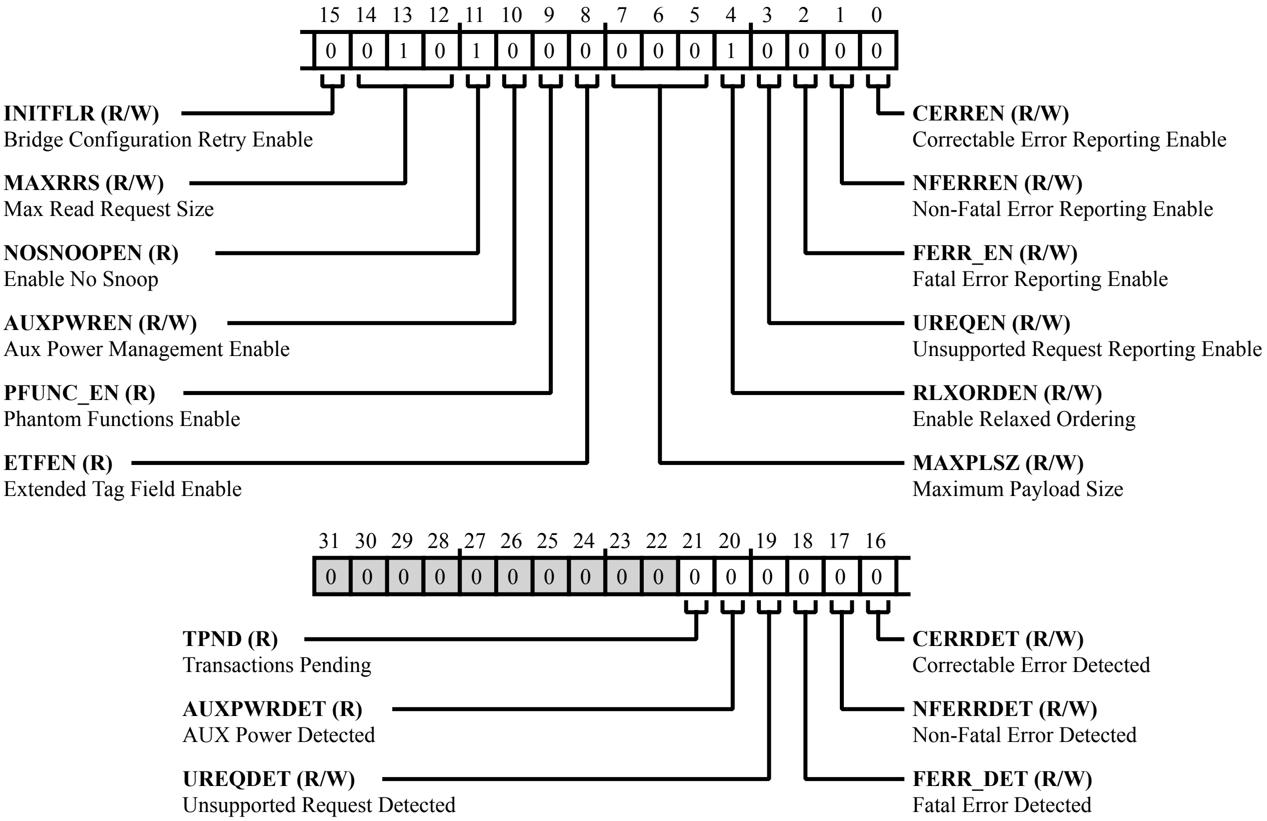

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
INITFLR (R/W)
Bridge Configuration Retry Enable
MAXRRS (R/W)
Max Read Request Size
NOSNOOPEN (R)
Enable No Snoop
AUXPWREN (R/W)
Aux Power Management Enable
PFUNC_EN (R)
Phantom Functions Enable
ETFEN (R)
Extended Tag Field Enable
CERREN (R/W)
Correctable Error Reporting Enable
NFERREN (R/W)
Non-Fatal Error Reporting Enable
FERR_EN (R/W)
Fatal Error Reporting Enable
UREQEN (R/W)
Unsupported Request Reporting Enable
RLXORDEN (R/W)
Enable Relaxed Ordering
MAXPLSZ (R/W)
Maximum Payload Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TPND (R)
Transactions Pending
AUXPWRDET (R)
AUX Power Detected
UREQDET (R/W)
Unsupported Request Detected
CERRDET (R/W)
Correctable Error Detected
NFERRDET (R/W)
Non-Fatal Error Detected
FERR_DET (R/W)
Fatal Error Detected

Figure 29-40: PCIE_DEV_CTLSTAT_[n] Register Diagram

Table 29-49: PCIE_DEV_CTLSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/NW)	TPND	Transactions Pending. When the `PCIE_DEV_CTLSTAT_[n].TPND` bit is set for endpoints, it indicates that the Function has issued Non-Posted Requests that have not been completed. A Function reports this bit cleared only when all outstanding Non-Posted Requests have completed or have been terminated by the Completion Timeout mechanism. This bit must also be cleared upon the completion of an FLR. For Root Complex and Switch Ports, when this bit is set, it indicates that a Port has issued Non-Posted Requests on its own behalf (using the Ports own Requester ID) which have not been completed. The Port reports this bit cleared only when all such outstanding Non- Posted Requests have completed or have been terminated by the Completion Timeout mechanism. Note that Root and Switch Ports implementing only the functionality required by this document do not issue Non-Posted Requests on their own behalf, and therefore are not subject to this case. Root and Switch Ports that do not issue Non-Posted Requests on their own behalf hardwire this bit to 0.
20 (R/NW)	AUXPWRDET	AUX Power Detected. The `PCIE_DEV_CTLSTAT_[n].AUXPWRDET` is read-only. Functions that require Aux power report this bit as set if Aux power is detected by the Function.
19 (R/W)	UREQDET	Unsupported Request Detected. The `PCIE_DEV_CTLSTAT_[n].UREQDET` bit indicates that the Function received an Unsupported Request. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control register. For a multi-Function device, each Function indicates status of errors as perceived by the respective Function. Default value of this bit is 0.
18 (R/W)	FERR_DET	Fatal Error Detected. The `PCIE_DEV_CTLSTAT_[n].FERR_DET` bit indicates status of Fatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control register. For a multi-Function device, each Function indicates status of errors as perceived by the respective Function. For Functions supporting Advanced Error Handling, errors are logged in this register regardless of the settings of the Uncorrectable Error Mask register. Default value of this bit is 0.
17 (R/W)	NFERRDET	Non-Fatal Error Detected. The `PCIE_DEV_CTLSTAT_[n].NFERRDET` bit indicates status of Nonfatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control register. For a multi-Function device, each Function indicates status of errors as perceived by the respective Function. For Functions supporting Advanced Error Handling, errors are logged in this register regardless of the settings of the Uncorrectable Error Mask register. Default value of this bit is 0.

Table 29-49: PCIE_DEV_CTLSTAT_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>16
(R/W)</td><td>CERRDET</td><td colspan="2">Correctable Error Detected.
The PCIE_DEV_CTLSTAT_[n].CERRDET bit indicates status of correctable errors detected. Errors are logged in this register regardless of whether error reporting is enabled or not in the Device Control register. For a multi-Function device, each Function indicates status of errors as perceived by the respective Function.
For Functions supporting Advanced Error Handling, errors are logged in this register regardless of the settings of the Correctable Error Mask register. Default value of this bit is 0.</td></tr>
<tr><td>15
(R/W)</td><td>INITFLR</td><td colspan="2">Bridge Configuration Retry Enable.</td></tr>
<tr><td rowspan="9">14:12
(R/W)</td><td rowspan="9">MAXRRS</td><td colspan="2">Max Read Request Size.
The PCIE_DEV_CTLSTAT_[n].MAXRRS bit field sets the maximum Read Request size for the Function as a Requester. The Function must not generate Read Requests with a size exceeding the set value. Functions that do not generate Read Requests larger than 128 bytes and Functions that do not generate Read Requests on their own behalf are permitted to implement this field as Read Only (RO) with a value of 000b. Default value of this field is 010b.</td></tr>
<tr><td>0</td><td>128 bytes maximum Read Request size</td></tr>
<tr><td>1</td><td>256 bytes maximum Read Request size</td></tr>
<tr><td>2</td><td>512 bytes maximum Read Request size</td></tr>
<tr><td>3</td><td>1024 bytes maximum Read Request size</td></tr>
<tr><td>4</td><td>2048 bytes maximum Read Request size</td></tr>
<tr><td>5</td><td>4096 bytes maximum Read Request size</td></tr>
<tr><td>6</td><td>Reserved</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>11
(R/NW)</td><td>NOSNOOPEN</td><td colspan="2">Enable No Snoop.
The PCIE_DEV_CTLSTAT_[n].NOSNOOPEN bit allows the Function to Set the No Snoop bit in the Requester Attributes of transactions it initiates that do not require hardware enforced cache coherency. Note that setting this bit to 1 should not cause a Function to set the No Snoop attribute on all transactions that it initiates. Even when the PCIE_DEV_CTLSTAT_[n].NOSNOOPEN bit is set, a Function is only permitted to set the No Snoop attribute on a transaction when it can guarantee that the address of the transaction is not stored in any cache in the system. This bit is permitted to be hardwired to 0 if a Function would never set the No Snoop attribute in transactions it initiates.</td></tr>
</table>

Table 29-49: PCIE_DEV_CTLSTAT_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>10
(R/W)</td><td>AUXPWREN</td><td colspan="2">Aux Power Management Enable.
The PCIE_DEV_CTLSTAT_[n].AUXPWREN bit enables a Function to draw Aux power independent of PME Aux power.</td></tr>
<tr><td>9
(R/NW)</td><td>PFUNC_EN</td><td colspan="2">Phantom Functions Enable.
When the PCIE_DEV_CTLSTAT_[n].PFUNC_EN bit is set, a Function can use unclaimed Functions as Phantom Functions to extend the number of outstanding transaction identifiers. If the bit =0, the Function is not allowed to use Phantom Functions.</td></tr>
<tr><td>8
(R/NW)</td><td>ETFEN</td><td colspan="2">Extended Tag Field Enable.
If the PCIE_DEV_CTLSTAT_[n].ETFEN bit is Set, the Function is permitted to set the Relaxed Ordering bit in the Attributes field of transactions it initiates that do not require strong write ordering. A Function is permitted to hardwire this bit to 0 if it never sets the Relaxed Ordering attribute in transactions it initiates as a Requester. Default value of this bit is 1.</td></tr>
<tr><td rowspan="9">7:5
(R/W)</td><td rowspan="9">MAXPLSZ</td><td colspan="2">Maximum Payload Size.
The PCIE_DEV_CTLSTAT_[n].MAXPLSZ bit field sets maximum TLP payload size for the Function. As a Receiver, the Function must handle TLPs as large as the set value. As a Transmitter, the Function must not generate TLPs exceeding the set value. Permissible values that can be programmed are indicated by the Max_Payload_Size Supported field in the Device Capabilities register.
Functions that support only the 128-byte max payload size are permitted to hardwire this field to 000.</td></tr>
<tr><td>0</td><td>128 bytes max payload size</td></tr>
<tr><td>1</td><td>256 bytes max payload size</td></tr>
<tr><td>2</td><td>512 bytes max payload size</td></tr>
<tr><td>3</td><td>1024 bytes max payload size</td></tr>
<tr><td>4</td><td>2048 bytes max payload size</td></tr>
<tr><td>5</td><td>4096 bytes max payload size</td></tr>
<tr><td>6</td><td>Reserved</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>4
(R/W)</td><td>RLXORDEN</td><td colspan="2">Enable Relaxed Ordering.
If the PCIE_DEV_CTLSTAT_[n].RLXORDEN bit is Set, the Function is permitted to set the Relaxed Ordering bit in the Attributes field of transactions it initiates that do not require strong write ordering. A Function is permitted to hardwire this bit to 0b if it never sets the Relaxed Ordering attribute in transactions it initiates as a Requester. Default value of this bit is 1.</td></tr>
</table>

Table 29-49: PCIE_DEV_CTLSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	UREQEN	Unsupported Request Reporting Enable. The `PCIE_DEV_CTLSTAT_[n].UREQEN` bit, in conjunction with other bits, controls the signaling of Unsupported Request Errors by sending error Messages. For a multi-Function device, this bit controls error reporting for each Function from point-of-view of the respective Function. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. Default value of this bit is 0.
2 (R/W)	FERR_EN	Fatal Error Reporting Enable. The `PCIE_DEV_CTLSTAT_[n].FERR_EN` bit, in conjunction with other bits, controls sending fatal error Messages. For a multi-Function device, this bit controls error reporting for each Function from point-of-view of the respective Function. For a Root Port, the reporting of Fatal errors is internal to the root. No external fatal error Message is generated. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0b. Default value of this bit is 0.
1 (R/W)	NFERREN	Non-Fatal Error Reporting Enable. The `PCIE_DEV_CTLSTAT_[n].NFERREN` bit, in conjunction with other bits, controls sending non-fatal error Messages. For a multi-Function device, this bit controls error reporting for each Function from point-of-view of the respective Function. For a Root Port, the reporting of Non-fatal errors is internal to the root. No external non-fatal error Message is generated. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. Default value of this bit is 0.
0 (R/W)	CERREN	Correctable Error Reporting Enable. The `PCIE_DEV_CTLSTAT_[n].CERREN` bit, in conjunction with other bits, controls sending correctable error messages. For a multi-Function device, this bit controls error reporting for each Function from point-of-view of the respective Function. For a Root Port, the reporting of correctable errors is internal to the root. No external ERR_COR Message is generated. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. Default value of this bit is 0.

DMA Read Arbitration Weight Low Off Register

The `PCIE_DMARD_ARBWGT_LO_[n]` register for 5-bit channel weight (for read channels 0-3) specifies the number of TLP requests that the DMA can issue for that channel before it must return to the arbitration routine. When the channel weight count is reached, the WWR arbiter selects the next channel to be processed. Your software must initialize this register before ringing the doorbell. Value range is (0-0x1F) corresponding to (1-32) transactions.

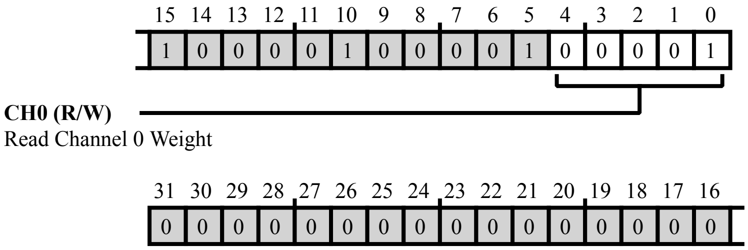

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
CH0 (R/W)
Read Channel 0 Weight
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-41: PCIE_DMARD_ARBWGT_LO_[n] Register Diagram

Table 29-50: PCIE_DMARD_ARBWGT_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	CH0	Read Channel 0 Weight. The `PCIE_DMARD_ARBWGT_LO_[n].CH0` bit field value is used by the channel weighted round robin arbiter to select the next channel read request. The weight is initialized by software before ringing the doorbell.

DMA Read Channel 1 and 0 IMWr Data Register

The `PCIE_DMARD_CH01_IMWR_[n]` register is implemented in RAM whose contents are uninitialized after power on. This register must always have an initialized value because the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

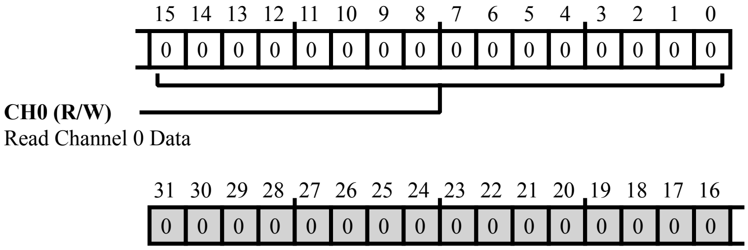

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CH0 (R/W)
Read Channel 0 Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-42: PCIE_DMARD_CH01_IMWR_[n] Register Diagram

Table 29-51: PCIE_DMARD_CH01_IMWR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CH0	Read Channel 0 Data. The DMA uses the `PCIE_DMARD_CH01_IMWR_[n].CH0` bit field to generate the data field for the Done or Abort IMWr TLPs it generates for read channel 0.

DMA Control 1 Read Channel Register

The `PCIE_DMARD_CTL1_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked "Reserved" MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

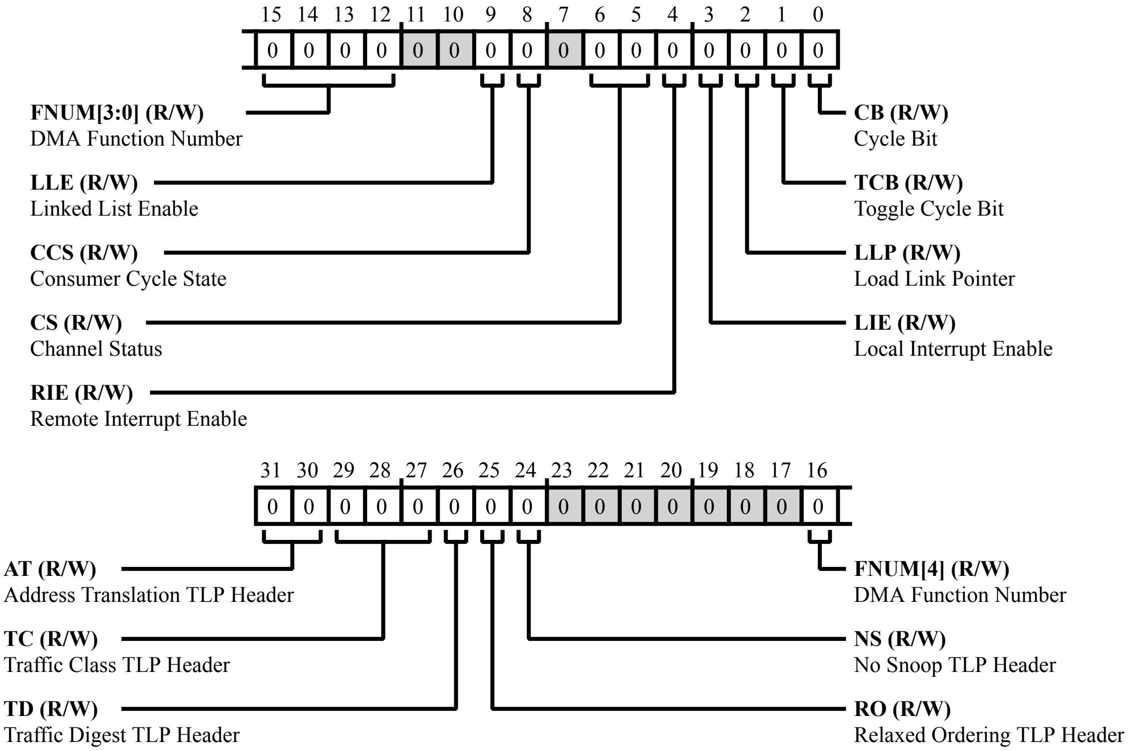

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FNUM[3:0] (R/W)
DMA Function Number
LLE (R/W)
Linked List Enable
CCS (R/W)
Consumer Cycle State
CS (R/W)
Channel Status
RIE (R/W)
Remote Interrupt Enable
CB (R/W)
Cycle Bit
TCB (R/W)
Toggle Cycle Bit
LLP (R/W)
Load Link Pointer
LIE (R/W)
Local Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AT (R/W)
Address Translation TLP Header
TC (R/W)
Traffic Class TLP Header
TD (R/W)
Traffic Digest TLP Header
FNUM[4] (R/W)
DMA Function Number
NS (R/W)
No Snoop TLP Header
RO (R/W)
Relaxed Ordering TLP Header

Figure 29-43: PCIE_DMARD_CTL1_[n] Register Diagram

Table 29-52: PCIE_DMARD_CTL1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	AT	Address Translation TLP Header. The DMA uses the `PCIE_DMARD_CTL1_[n].AT` bit field when generating MRd/MWr (not IMWr) TLPs.
29:27 (R/W)	TC	Traffic Class TLP Header. The DMA uses the `PCIE_DMARD_CTL1_[n].TC` bit field when generating MRd/MWr (not IMWr) TLPs.
26 (R/W)	TD	Traffic Digest TLP Header. The PCIe core adds the ECRC field and sets the `PCIE_DMARD_CTL1_[n].TD` bit in the TLP header. The DMA uses this TLP header field when generating MRd/MWr (not IMWr) TLPs.

Table 29-52: PCIE_DMARD_CTL1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	RO	Relaxed Ordering TLP Header. The DMA uses the `PCIE_DMARD_CTL1_[n].RO` bit field when generating MRd/MWr (not IMWr) TLPs.	
24 (R/W)	NS	No Snoop TLP Header. The `PCIE_DMARD_CTL1_[n].NS` bit is used by DMA when generating MRd/MWr (not IMWr) TLPs.	
16:12 (R/W)	FNUM	DMA Function Number. The core uses the `PCIE_DMARD_CTL1_[n].FNUM` bit field when generating the requester ID for the MRd/MWr DMA TLP. When SR-IOV is enabled, this field is ignored if the VFE field in the DMA Write Channel Control 2 Register is set.	
9 (R/W)	LLE	Linked List Enable. The `PCIE_DMARD_CTL1_[n].LLE` bit enables linked list operation.	
		0	Disable Linked Lists
		1	Enable Linked Lists
8 (R/W)	CCS	Consumer Cycle State. The `PCIE_DMARD_CTL1_[n].CCS` bit is used to synchronize the producer (software) and the consumer (DMA). This bit must be initialized. The DMA updates the `PCIE_DMARD_CTL1_[n].CCS` bit during linked list operation. Used in linked list mode only.	
6:5 (R/W)	CS	Channel Status. The `PCIE_DMARD_CTL1_[n].CS` status bits identify the current operational state of the DMA channel.	
		0	Reserved
		1	Running. This channel is active and transferring data.
		2	Halted. An error condition has been detected, and the DMA has stopped this channel.
		3	Stopped. DMA transferred all data for this channel, or channel prematurely stopped by writing to the Stop field of the DMA Write Doorbell Register or DMA Read Doorbell Register.
4 (R/W)	RIE	Remote Interrupt Enable. Programs must set the `PCIE_DMARD_CTL1_[n].RIE` bit to enable the generation of the Done or Abort Remote interrupts. In LL mode, the DMA overwrites this with the RIE of the LL element. The `PCIE_DMARD_CTL1_[n].RIE` bit in a LL element only enables the Done interrupt. In non-LL mode, this bit enables the Done and Abort interrupts. This field is not defined in a link LL element.	

Table 29-52: PCIE_DMARD_CTL1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	LIE	Local Interrupt Enable. Programs must set the `PCIE_DMARD_CTL1_[n].LIE` bit to enable the generation of the Done or Abort Local interrupts. In LL mode, the DMA overwrites the `PCIE_DMARD_CTL1_[n].LIE` bit with the LIE of the LL element. The LIE bit in a LL element only enables the Done interrupt. In non-LL mode, the LIE bit enables the Done and Abort interrupts. This field is not defined in a link LL element.
2 (R/W)	LLP	Load Link Pointer. The `PCIE_DMARD_CTL1_[n].LLP` bit indicates that this linked list element is a link element, and its LL element pointer dwords point to the next (non-contiguous) element. The DMA loads this field with the LLP of the linked list element. Used in linked list mode only.
1 (R/W)	TCB	Toggle Cycle Bit. The `PCIE_DMARD_CTL1_[n].TCB` bit indicates to the DMA to toggle its interpretation of the CB. It is used to synchronize the producer (software) and the consumer (DMA). The DMA loads this field with the TCB of the linked list element. This field is not defined in a data LL element and is used in linked list mode only.
0 (R/W)	CB	Cycle Bit. The `PCIE_DMARD_CTL1_[n].CB` bit is used to synchronize the producer (software) and the consumer (DMA). The DMA loads this field with the CB of the linked list element. Used in linked list mode only.

DMA Destination Address High Read Channel Register

The `PCIE_DMARD_DAR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked "Reserved" MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

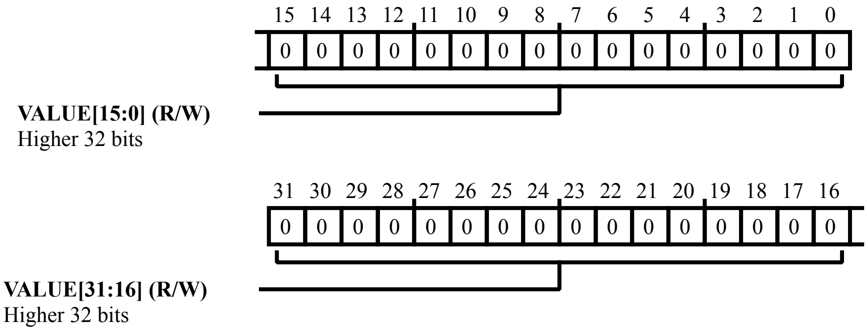

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Higher 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Higher 32 bits

Figure 29-44: PCIE_DMARD_DAR_HI_[n] Register Diagram

Table 29-53: PCIE_DMARD_DAR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Higher 32 bits. In LL mode, the DMA overwrites the `PCIE_DMARD_DAR_HI_[n].VALUE` bit field with the corresponding dword of the LL element.

DMA Destination Address Low Read Channel Register

The `PCIE_DMARD_DAR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked "Reserved" MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

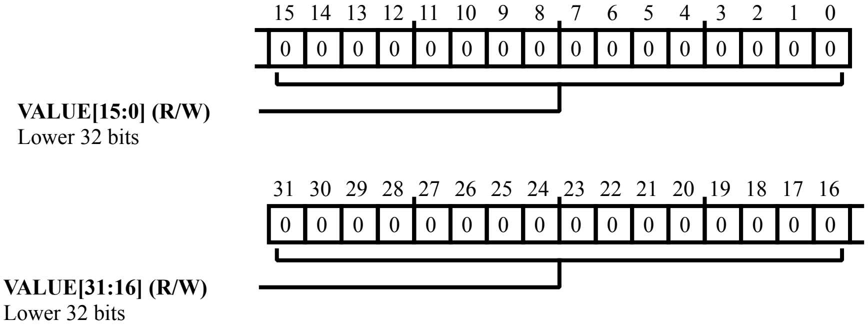

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower 32 bits

Figure 29-45: PCIE_DMARD_DAR_LO_[n] Register Diagram

Table 29-54: PCIE_DMARD_DAR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower 32 bits. The `PCIE_DMARD_DAR_LO_[n].VALUE` bit field indicates the next address to be written to. The DMA increments the DAR as the DMA transfer progresses. The DAR is always dword aligned even if the transfer size is byte aligned. In LL mode, the DMA overwrites this bit field with the corresponding dword of the LL element. he DAR is the address of the local memory.

DMA Read Done IMWr Address High Register

The `PCIE_DMARD_DONE_IMWR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must always initialize this register value because the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

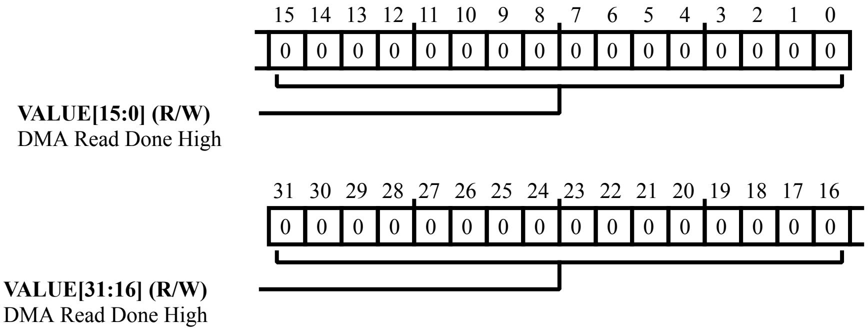

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Read Done High
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Read Done High

Figure 29-46: PCIE_DMARD_DONE_IMWR_HI_[n] Register Diagram

Table 29-55: PCIE_DMARD_DONE_IMWR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Read Done High. The DMA uses the `PCIE_DMARD_DONE_IMWR_HI_[n].VALUE` bit field to generate bits [63:32] of the address field for the Done IMWr TLP.

DMA Read Done IMWr Address Low Register

The `PCIE_DMARD_DONE_IMWR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must always initialize this register value because the default is undefined. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

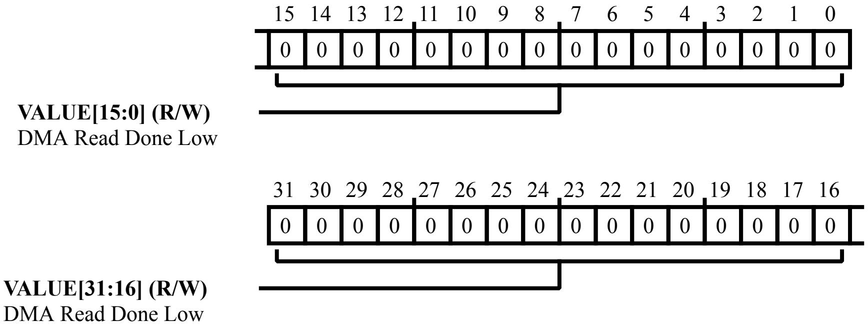

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Read Done Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Read Done Low

Figure 29-47: PCIE_DMARD_DONE_IMWR_LO_[n] Register Diagram

Table 29-56: PCIE_DMARD_DONE_IMWR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Read Done Low. The DMA uses the `PCIE_DMARD_DONE_IMWR_LO_[n].VALUE` bit field to generate bits [31:0] of the address field for the Done IMWr TLP. Bits [1:0] must be 00 as this address must be dword aligned.

DMA Read Doorbell Register

The `PCIE_DMARD_DRBL_[n]` bit register provides the doorbell number and stops issuing requests.

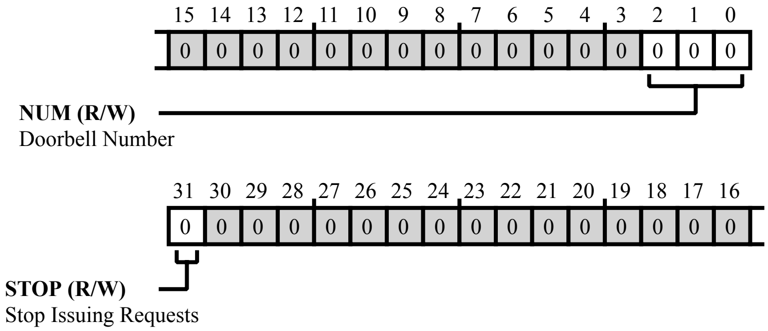

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NUM (R/W)
Doorbell Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STOP (R/W)
Stop Issuing Requests

Figure 29-48: PCIE_DMARD_DRBL_[n] Register Diagram

Table 29-57: PCIE_DMARD_DRBL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	STOP	Stop Issuing Requests. The `PCIE_DMARD_DRBL_[n].STOP` bit is set in conjunction with the doorbell number field. The DMA read channel stops issuing requests, sets the channel status to Stopped, and asserts the Abort interrupt (if enabled). Before setting the `PCIE_DMARD_DRBL_[n].STOP` bit, programs must read the `PCIE_DMARD_CTL1_[n].CS` (channel status) to ensure that the read channel is running (transferring data).
2:0 (R/W)	NUM	Doorbell Number. The `PCIE_DMARD_DRBL_[n].NUM` bit field is the doorbell number. Programs must write 0x0 to this bit field to start the DMA read transfer for that channel. The DMA detects a write to this register field even if the value of this field does not change. The range of this field is 0x0 to 0x7, and 0x0 corresponds to channel 0.

DMA Read Engine Enable Register

The `PCIE_DMARD_EN_[n]` register enables the DMA read engine. Programs must initially enable the the DMA read engine (=1) for normal operation, before any other software setup. The enable bit in this register does not need to be toggled or rewritten during normal operation. To soft reset the DMA controller read logic this bit should be cleared (=0). There are three possible reasons for resetting the DMA controller read logic.

- The `PCIE_DMARD_ISTAT_[n].ABRT` bit is set and any of the bits in the `PCIE_DMARD_ERRSTAT_LO_[n]` register are set. Resetting the DMA controller read logic re-initializes the control logic, ensuring that the next DMA read transfer is executed successfully.
- The procedure outlined in "Stop Bit" has been executed, after which, the `PCIE_DMARD_ISTAT_[n].ABRT` bit is set and the `PCIE_DMARD_CTL1_[n].CS` is set to "Stopped". Resetting the DMA controller read logic re-initializes the control logic ensuring that the next DMA read transfer is executed successfully.
- If the DMA engine is incorrectly programmed during software development.

To Soft Reset the DMA controller read logic:

- De-assert the DMA read engine enable bit.
- Wait for the DMA to complete any in-progress TLP transfer, by waiting until a read on the DMA read engine enable bit returns a 0.
- Assert the DMA read engine enable bit.

This Soft Reset does not clear the DMA configuration registers. The DMA read transfer does not start until you write to the `PCIE_DMARD_DRBL_[n].STOP` bit.

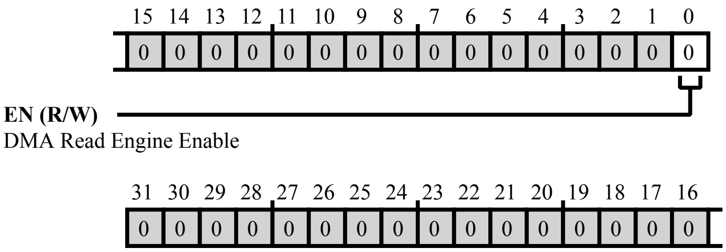


Figure 29-49: PCIE_DMARD_EN_[n] Register Diagram

Table 29-58: PCIE_DMARD_EN_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Read Engine Enable. The `PCIE_DMARD_EN_[n].EN` bit enables or disables the DMA read engine.	
		0	Disable engine (soft reset)
		1	Enable engine

DMA Read Error Status High Register

The PCIE_DMARD_ERRSTAT_HI_[n] register reports errors for read channels 4-7.

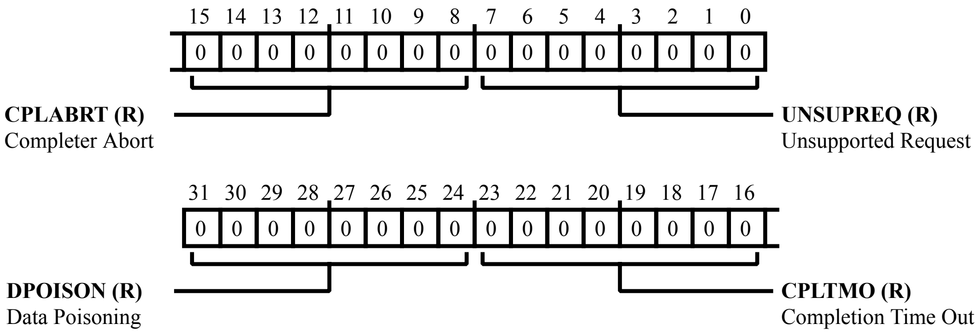

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPLABRT (R)
Completer Abort
UNSUPREQ (R)
Unsupported Request
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DPOISON (R)
Data Poisoning
CPLTMO (R)
Completion Time Out

Figure 29-50: PCIE_DMARD_ERRSTAT_HI_[n] Register Diagram

Table 29-59: PCIE_DMARD_ERRSTAT_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	DPOISON	Data Poisoning. The PCIE_DMARD_ERRSTAT_HI_[n].DPOISON bit field reports that the DMA read channel has detected data poisoning in the completion from the remote device (in response to the MRd request). The DMA read channel drops the completion and then halts. The CX_FLT_MSK_UR_POIS filter rule does not affect this behavior. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enable. For details, see "Interrupts and Error Handling". • Mask. The DMA read interrupt Mask register has no effect on this register. • Clear. Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the (PCIE_DMARD_ICLR_[n]) register to clear this error bit. Note that this action also clears the other error bits for the same channel in this register and in the PCIE_DMARD_ERRSTAT_LO_[n] register.
23:16 (R/NW)	CPLTMO	Completion Time Out. The PCIE_DMARD_ERRSTAT_HI_[n].CPLTMO bit field reports that the DMA read channel has timed-out while waiting for the remote device to respond to the MRd request, or a malformed CplD has been received. For more details, see "Linked List Mode". Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enable. For details, see "Interrupts and Error Handling". • Mask. The DMA read interrupt Mask register has no effect on this register. • Clear. Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the (PCIE_DMARD_ICLR_[n]) register to clear this error bit. Note that this action also clears the other error bits for the same channel in this register and in the PCIE_DMARD_ERRSTAT_LO_[n] register.

Table 29-59: PCIE_DMARD_ERRSTAT_HI_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	CPLABRT	Completer Abort. The `PCIE_DMARD_ERRSTAT_HI_[n].CPLABRT` bit field reports that the DMA read channel has received a PCIe completer abort completion status from the remote device in response to the MRd request. For more details, see "Linked List Mode". Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enable. For details, see "Interrupts and Error Handling". • Mask. The DMA read interrupt Mask register has no effect on this register. • Clear. Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the (`PCIE_DMARD_ICLR_[n]`) register to clear this error bit. Note that this action also clears the other error bits for the same channel in this register and in the `PCIE_DMARD_ERRSTAT_LO_[n]` register.
7:0 (R/NW)	UNSUPREQ	Unsupported Request. The `PCIE_DMARD_ERRSTAT_HI_[n].UNSUPREQ` bit field reports that the DMA read channel has received a PCIe unsupported request completion status from the remote device in response to the MRd request. For more details, see "Linked List Mode". Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. - Enabling: For details, see "Interrupts and Error Handling". - Masking: The DMA read interrupt Mask register has no effect on this register. - Clearing: You must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the "DMA Read Interrupt Clear Register" (DMA_RD_INT_CLEAR_OFF) to clear this error bit. Note, this also clears the other error bits for the same channel in this register and in the DMA Read Error Status Low register.

DMA Read Error Status Low Register

The `PCIE_DMARD_ERRSTAT_LO_[n]` register reports errors for read channels 0-3.

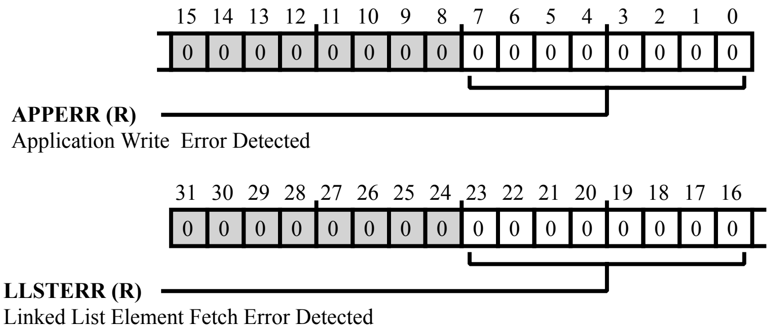


Figure 29-51: PCIE_DMARD_ERRSTAT_LO_[n] Register Diagram

Table 29-60: PCIE_DMARD_ERRSTAT_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	LLSTERR	Linked List Element Fetch Error Detected. The `PCIE_DMARD_ERRSTAT_LO_[n].LLSTERR` bit field reports that the DMA read channel has received an error response from the AHB/AXI bus while reading a linked list element from local memory. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enable. For details, see "Interrupts and Error Handling". • Mask. The DMA read interrupt Mask register has no effect on this register. • Clear. Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the (`PCIE_DMARD_ICLR_[n]`) register to clear this error bit. Note that this action also clears the other error bits for the same channel in this register and in the `PCIE_DMARD_ERRSTAT_HI_[n]` register.
7:0 (R/NW)	APPERR	Application Write Error Detected. The `PCIE_DMARD_ERRSTAT_LO_[n].APPERR` bit field reports that the DMA read channel has received an error response from the peripheral/core bus (or RTRGT1 interface when the AHB/AXI Bridge is not used) while writing data to it. This error is fatal. The data transfer must be restarted from the beginning, as the channel context is corrupted, and the transfer is not rolled back. For more details, see "Linked List Mode". Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enable. For details, see "Interrupts and Error Handling". • Mask. The DMA read interrupt Mask register has no effect on this register. • Clear. Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the (`PCIE_DMARD_ICLR_[n]`) register to clear this error bit. Note that this action also clears the other error bits for the same channel in this register and in the `PCIE_DMARD_ERRSTAT_HI_[n]` register.

DMA Read Interrupt Clear Register

The `PCIE_DMARD_ICLR_[n]` register is used to clear corresponding bits in the `PCIE_DMARD_ISTAT_[n]` register.

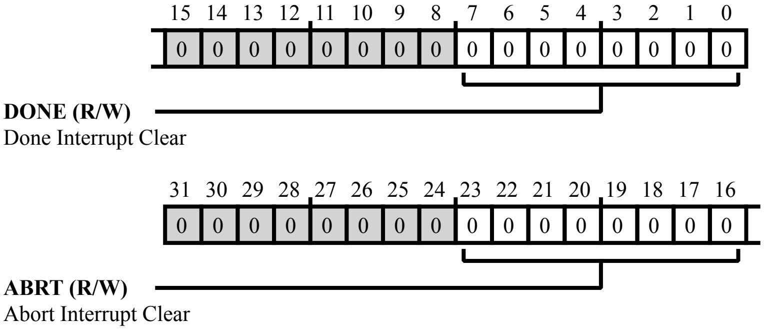

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DONE (R/W)
Done Interrupt Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABRT (R/W)
Abort Interrupt Clear

Figure 29-52: PCIE_DMARD_ICLR_[n] Register Diagram

Table 29-61: PCIE_DMARD_ICLR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Clear. Writing 1'b1 to the `PCIE_DMARD_ICLR_[n].ABRT` bit field clears the corresponding bit in the `PCIE_DMARD_ISTAT_[n].ABRT` bit field. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Reading from this self-clearing register field always returns 0.
7:0 (R/W)	DONE	Done Interrupt Clear. Writing 1'b1 to the `PCIE_DMARD_ICLR_[n].DONE` bit field clears the corresponding bit in the `PCIE_DMARD_ISTAT_[n].DONE` bit field. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Reading from this self-clearing register field always returns 0.

DMA Read Interrupt Mask Register

The `PCIE_DMARD_IMSK_[n]` register is used to mask interrupts in the `PCIE_DMARD_ISTAT_[n]` register.

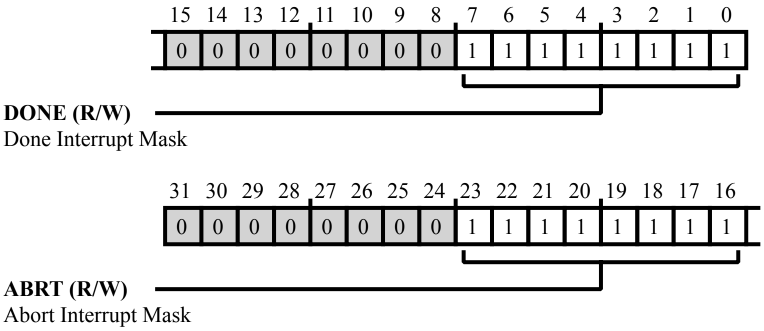

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
DONE (R/W)
Done Interrupt Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
ABRT (R/W)
Abort Interrupt Mask

Figure 29-53: PCIE_DMARD_IMSK_[n] Register Diagram

Table 29-62: PCIE_DMARD_IMSK_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Mask. The `PCIE_DMARD_IMSK_[n].ABRT` bit field prevents the `PCIE_DMARD_ISTAT_[n].ABRT` status field in the DMA from asserting the edma_int output. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0.
7:0 (R/W)	DONE	Done Interrupt Mask. The `PCIE_DMARD_IMSK_[n].DONE` bit field prevents the `PCIE_DMARD_ISTAT_[n].DONE` status field in the DMA from asserting the edma_int output. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0.

DMA Read Abort IMWr Address High Register

The `PCIE_DMARD_IMWRABRT_HI_[n]` register is implemented in RAM whose contents are uninitialized after power on. Programs must always initialize this register value because the default is undefined. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

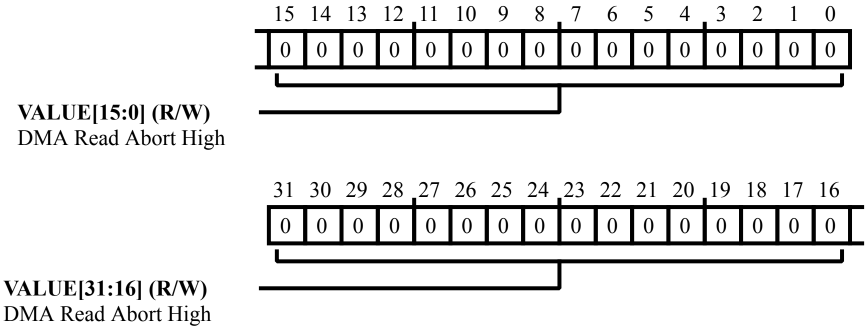

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Read Abort High
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Read Abort High

Figure 29-54: PCIE_DMARD_IMWRABRT_HI_[n] Register Diagram

Table 29-63: PCIE_DMARD_IMWRABRT_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Read Abort High. The `PCIE_DMARD_IMWRABRT_HI_[n].VALUE` bit field is used by the DMA to generate bits [63:32] of the address field for the Abort IMWr TLP.

DMA Read Abort IMWr Address Low Register

The `PCIE_DMARD_IMWRABRT_LO_[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must always initialize this register value because the default is undefined. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

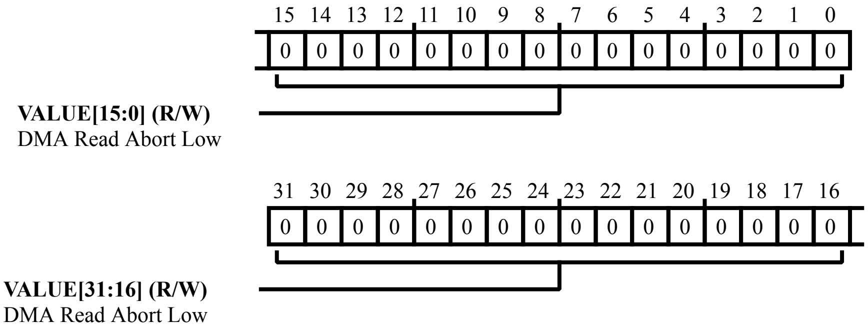

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Read Abort Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Read Abort Low

Figure 29-55: PCIE_DMARD_IMWRABRT_LO_[n] Register Diagram

Table 29-64: PCIE_DMARD_IMWRABRT_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Read Abort Low. The `PCIE_DMARD_IMWRABRT_LO_[n].VALUE` bit field is used by the DMA to generate bits [31:0] of the address field for the Abort IMWr TLP. Bits [1:0] must be 00 as this address must be dword aligned.

DMA Read Interrupt Status Register

The `PCIE_DMARD_ISTAT_[n]` register provides status information on DMA reads.

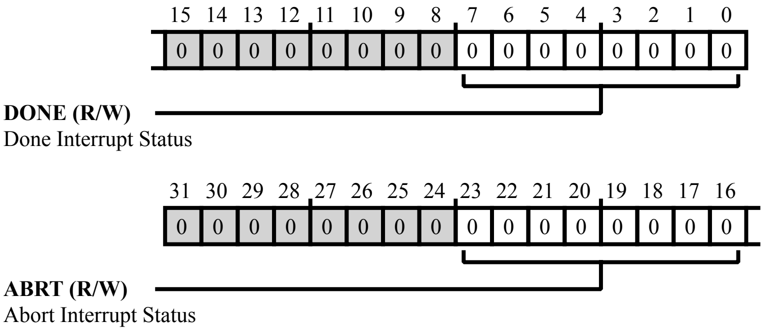


Figure 29-56: PCIE_DMARD_ISTAT_[n] Register Diagram

Table 29-65: PCIE_DMARD_ISTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Status. The `PCIE_DMARD_ISTAT_[n].ABRT` indicates the DMA read channel has detected an error or the transfer was stopped manually. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Programs can read the DMA Read Error Status Low Register and DMA Read Error Status High Register to determine the source of the error. The DMA read interrupt Mask register has no effect on this bit field. Programs must write a 1'b1 to the corresponding channel bit in the DMA read interrupt Clear register to clear this interrupt bit. Programs can write to this bit field to emulate interrupt generation, during software or hardware testing. A write to the address triggers an interrupt, but the DMA does not set the Done or Abort bits in this bit field.
7:0 (R/W)	DONE	Done Interrupt Status. The `PCIE_DMARD_ISTAT_[n].DONE` indicates the DMA read channel has successfully completed the DMA read transfer. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. The DMA read interrupt Mask register has no effect on this bit field. Programs must write a 1'b1 to the corresponding channel bit in the DMA read interrupt Clear register to clear this interrupt bit. Programs can write to this bit field to emulate interrupt generation, during software or hardware testing. A write to the address triggers an interrupt, but the DMA does not set the Done or Abort bits in this bit field.

DMA LLP High Read Channel Register

The `PCIE_DMARD_LLP_HI_[n]` register is implemented in RAM whose contents are uninitialized after power on. This register value must always be initialized because the default is undefined. All fields marked Reserved must be programmed to 1'b0. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

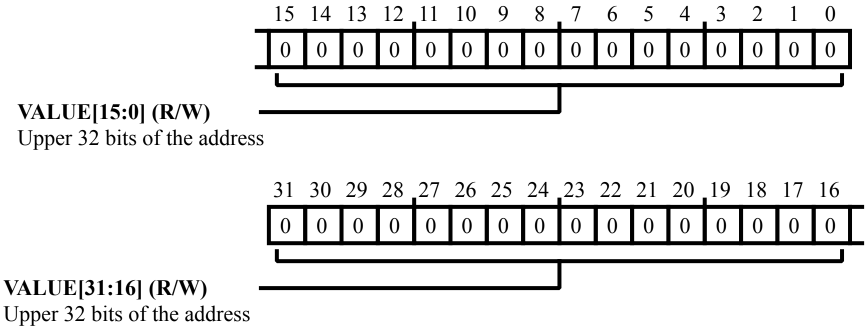

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Upper 32 bits of the address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Upper 32 bits of the address

Figure 29-57: PCIE_DMARD_LLP_HI_[n] Register Diagram

Table 29-66: PCIE_DMARD_LLP_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Upper 32 bits of the address. The `PCIE_DMARD_LLP_HI_[n].VALUE` bit field contains the upper 32 bits of the address of the linked list transfer list in local memory. Used in linked list mode only. The `PCIE_DMARD_LLP_HI_[n].VALUE` bit field is updated by the DMA to point to the next element in the transfer list as elements are consumed.

DMA LLP Low Read Channel Register

The `PCIE_DMARD_LLP_LO_[n]` register is implemented in RAM whose contents are uninitialized after power on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

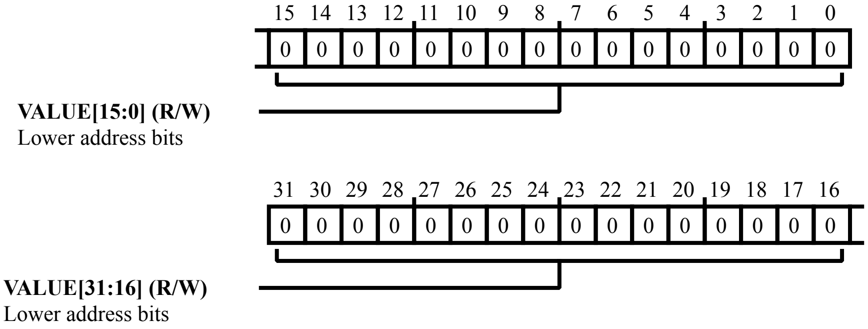

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower address bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower address bits

Figure 29-58: PCIE_DMARD_LLP_LO_[n] Register Diagram

Table 29-67: PCIE_DMARD_LLP_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower address bits. The `PCIE_DMARD_LLP_LO_[n].VALUE` bit field contains the lower bits of the address of the linked list transfer list in local memory. Used in linked list mode only. The `PCIE_DMARD_LLP_LO_[n].VALUE` bit field is updated by the DMA to point to the next element in the transfer list after the previous element is consumed. • When the current element is a data element; this field is incremented by 6. • When the current element is a link element; this field is overwritten by the LL Element Pointer of the element.

DMA Read Linked List Error Enable Register

The `PCIE_DMARD_LLSTERR_EN_[n]` register provides bits in the LL element that enable the channel done and abort interrupts (local and remote).

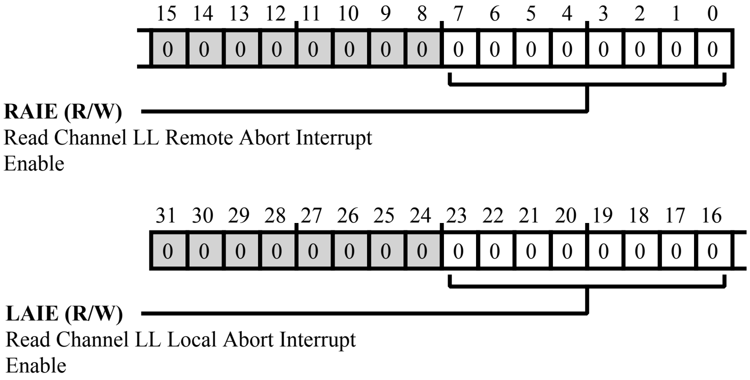

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RAIE (R/W)
Read Channel LL Remote Abort Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LAIE (R/W)
Read Channel LL Local Abort Interrupt Enable

Figure 29-59: PCIE_DMARD_LLSTERR_EN_[n] Register Diagram

Table 29-68: PCIE_DMARD_LLSTERR_EN_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	LAIE	Read Channel LL Local Abort Interrupt Enable. The `PCIE_DMARD_LLSTERR_EN_[n].LAIE` bit field enables the read channel local abort interrupt. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Used in linked list mode only.
7:0 (R/W)	RAIE	Read Channel LL Remote Abort Interrupt Enable. The `PCIE_DMARD_LLSTERR_EN_[n].RAIE` bit field enables the read channel remote abort interrupt. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Used in linked list mode only.

DMA SAR High Read Channel Register

The `PCIE_DMARD_SAR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

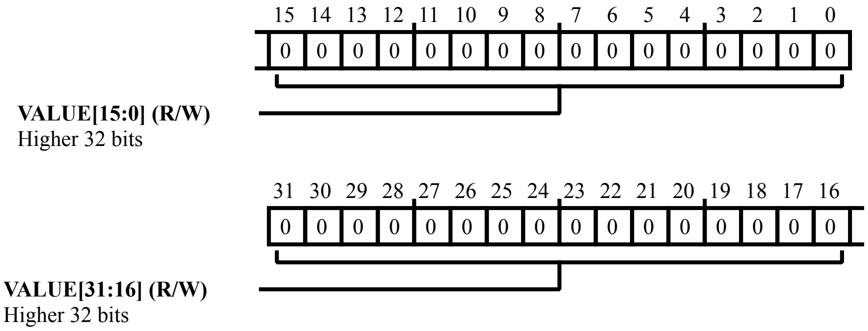

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Higher 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Higher 32 bits

Figure 29-60: PCIE_DMARD_SAR_HI_[n] Register Diagram

Table 29-69: PCIE_DMARD_SAR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Higher 32 bits. In LL mode, the DMA overwrites the `PCIE_DMARD_SAR_HI_[n].VALUE` bit field with the corresponding dword of the LL element.

DMA SAR Low Read Channel Register

The `PCIE_DMARD_SAR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

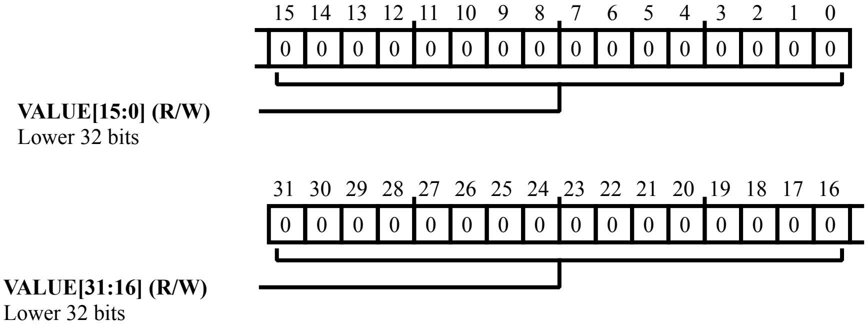

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower 32 bits

Figure 29-61: PCIE_DMARD_SAR_LO_[n] Register Diagram

Table 29-70: PCIE_DMARD_SAR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower 32 bits. The `PCIE_DMARD_SAR_LO_[n].VALUE` bit field indicates the next address to be read from. The DMA increments the SAR as the DMA transfer progresses. In LL mode, the DMA overwrites this with the corresponding dword of the LL element. The SAR is the address of the remote memory.

DMA Transfer Size Read Channel Register

The PCIE_DMARD_XFRSZ_[n] register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

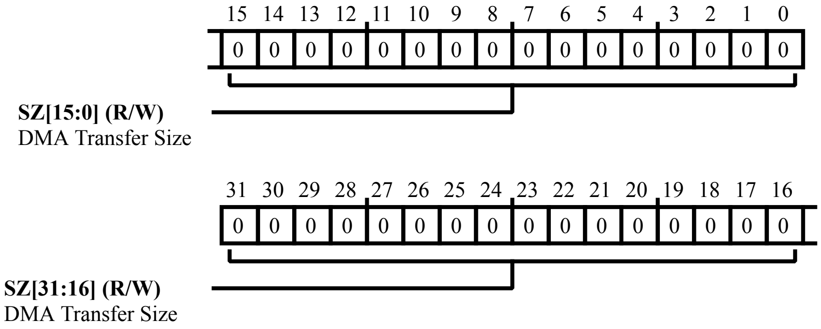

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SZ[15:0] (R/W)
DMA Transfer Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SZ[31:16] (R/W)
DMA Transfer Size

Figure 29-62: PCIE_DMARD_XFRSZ_[n] Register Diagram

Table 29-71: PCIE_DMARD_XFRSZ_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SZ	DMA Transfer Size. The `PCIE_DMARD_XFRSZ_[n].SZ` bit field configures the size of the DMA transfer. The maximum DMA transfer size is 4Gbytes. The minimum transfer size is one byte (0x1). This field is automatically decremented by the DMA as the DMA write and read channel transfer progresses. This field indicates the number bytes remaining to be transferred. When all bytes are successfully transferred the current transfer size is zero. In LL mode, the DMA overwrites the `PCIE_DMARD_XFRSZ_[n].SZ` bit field with the corresponding dword of the LL element.

DMA Write Abort IMWr Address High Register

The `PCIE_DMAWR_ABRT_IMWR_HI[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must program and initialize every field as the default is undefined. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

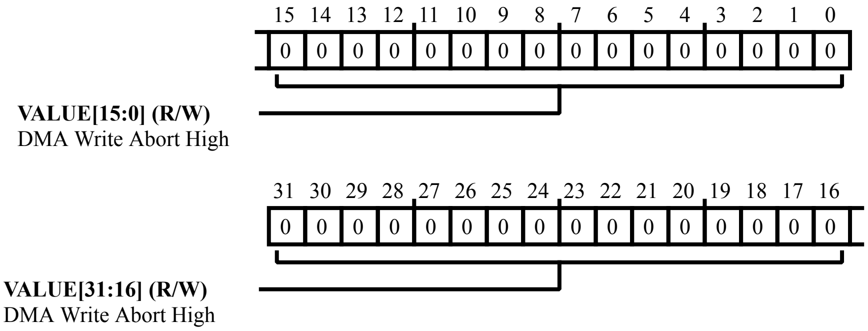

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Write Abort High
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Write Abort High

Figure 29-63: PCIE_DMAWR_ABRT_IMWR_HI[n] Register Diagram

Table 29-72: PCIE_DMAWR_ABRT_IMWR_HI[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Write Abort High. The DMA uses the `PCIE_DMAWR_ABRT_IMWR_HI[n].VALUE` bit field to generate bits [63:32] of the address field for the Abort IMWr TLP.

DMA Write Abort IMWr Address Low Register

The `PCIE_DMAWR_ABRT_IMWR_LO[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must program and initialize every field as the default is undefined. All fields marked Reserved MUST be programmed to 1'b0. This register is not affected by any of the reset signals.

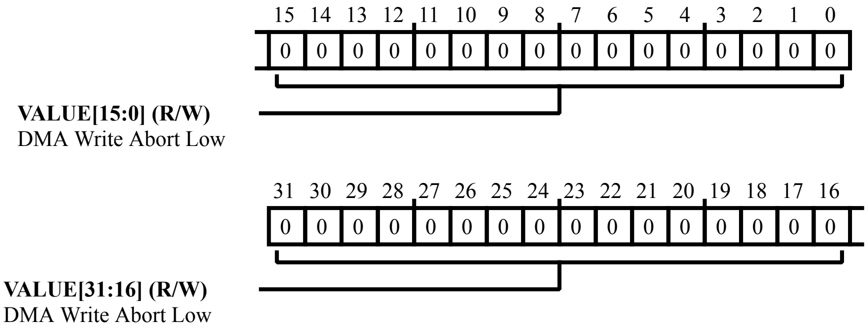

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Write Abort Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Write Abort Low

Figure 29-64: PCIE_DMAWR_ABRT_IMWR_LO[n] Register Diagram

Table 29-73: PCIE_DMAWR_ABRT_IMWR_LO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Write Abort Low. The DMA uses the `PCIE_DMAWR_ABRT_IMWR_LO[n].VALUE` bit field to generate bits [31:0] of the address field for the Abort IMWr TLP it generates. Bits [1:0] must be 00 as this address must be dword aligned.

DMA Write Channel Arbitration Weight Low Off Register

The `PCIE_DMAWR_ARBWGT_LO_[n]` register specifies the number of TLP requests that the DMA can issue for that channel before it must return to the arbitration routine. It does this for the 5-bit channel weight (for write channels 0-3). When the channel weight count is reached, the WWR arbiter selects the next channel to be processed. The software must initialize this register before ringing the doorbell. The value range is (0x00-0x1F) corresponding to (1-32) transactions.

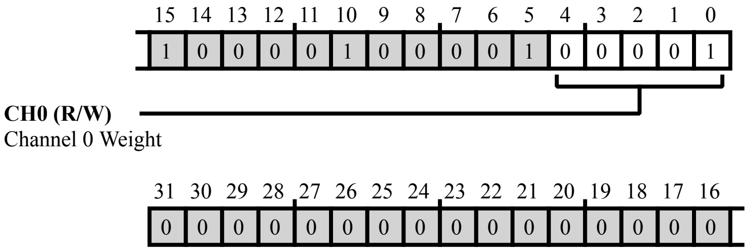

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
CH0 (R/W)
Channel 0 Weight
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-65: PCIE_DMAWR_ARBWGT_LO_[n] Register Diagram

Table 29-74: PCIE_DMAWR_ARBWGT_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	CH0	Channel 0 Weight. The `PCIE_DMAWR_ARBWGT_LO_[n].CH0` bit field value is used by the channel weighted round robin arbiter to select the next channel read request. The weight is initialized by software before ringing the doorbell.

DMA Write Channel 1 and 0 IMWr Data Register

The `PCIE_DMAWR_CH01_IMWR_[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must program and initialize every field as the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

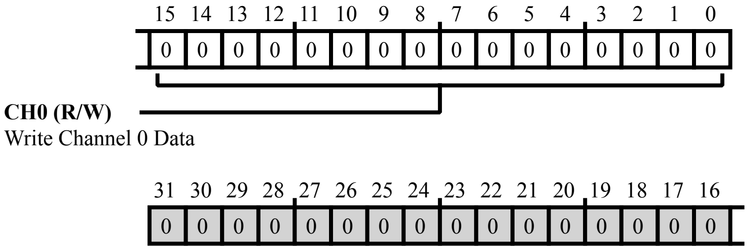

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CH0 (R/W)
Write Channel 0 Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-66: PCIE_DMAWR_CH01_IMWR_[n] Register Diagram

Table 29-75: PCIE_DMAWR_CH01_IMWR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CH0	Write Channel 0 Data. The DMA uses the `PCIE_DMAWR_CH01_IMWR_[n].CH0` bit field to generate the data field for the Done or Abort IMWr TLPs it generates for write channel 0.

DMA Control 1 Write Channel Register

The `PCIE_DMAWR_CTL1_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

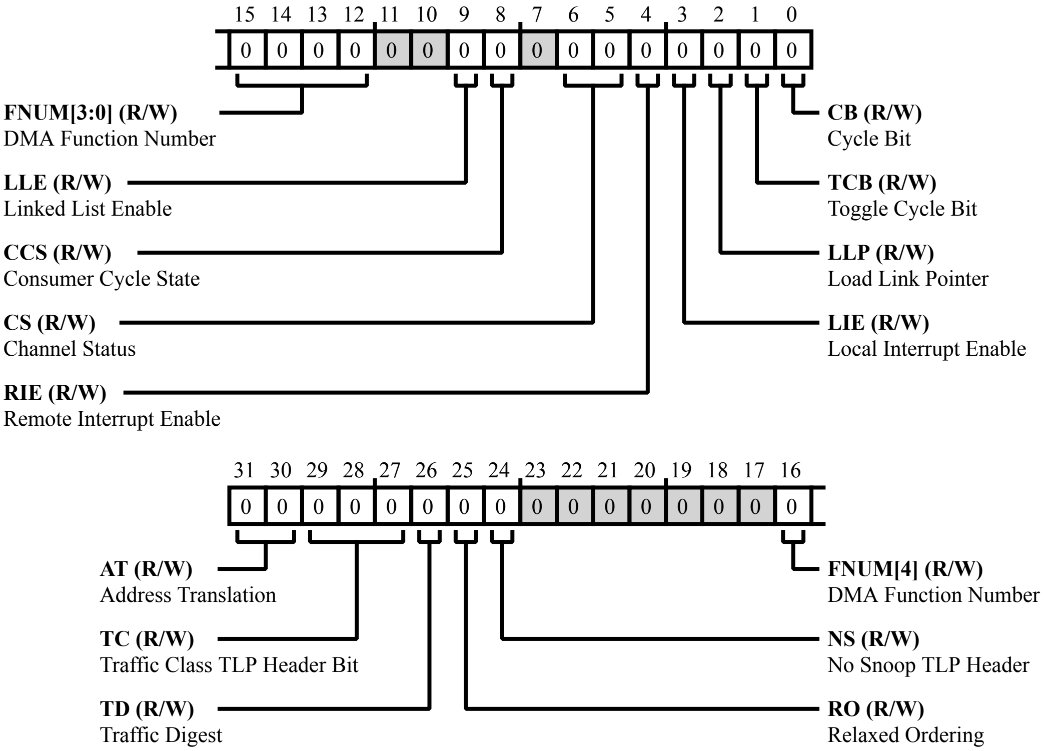

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FNUM[3:0] (R/W)
DMA Function Number
LLE (R/W)
Linked List Enable
CCS (R/W)
Consumer Cycle State
CS (R/W)
Channel Status
RIE (R/W)
Remote Interrupt Enable
CB (R/W)
Cycle Bit
TCB (R/W)
Toggle Cycle Bit
LLP (R/W)
Load Link Pointer
LIE (R/W)
Local Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AT (R/W)
Address Translation
TC (R/W)
Traffic Class TLP Header Bit
TD (R/W)
Traffic Digest
FNUM[4] (R/W)
DMA Function Number
NS (R/W)
No Snoop TLP Header
RO (R/W)
Relaxed Ordering

Figure 29-67: PCIE_DMAWR_CTL1_[n] Register Diagram

Table 29-76: PCIE_DMAWR_CTL1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	AT	Address Translation. The DMA uses the `PCIE_DMAWR_CTL1_[n].AT` TLP header field when generating memory read and memory write (not IMWr) TLPs.
29:27 (R/W)	TC	Traffic Class TLP Header Bit. The DMA uses the `PCIE_DMAWR_CTL1_[n].TC` bit TLP header field when generating memory read and memory write (not IMWr) TLPs.
26 (R/W)	TD	Traffic Digest. The PCIe core adds the ECRC field and sets the `PCIE_DMAWR_CTL1_[n].TD` bit in the TLP header. The DMA uses the `PCIE_DMAWR_CTL1_[n].TD` TLP header field when generating memory read and memory write (not IMWr) TLPs.

Table 29-76: PCIE_DMAWR_CTL1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	RO	Relaxed Ordering. The DMA uses the `PCIE_DMAWR_CTL1_[n].RO` TLP header field when generating memory read and memory write (not IMWr) TLPs.
24 (R/W)	NS	No Snoop TLP Header. The DMA uses the `PCIE_DMAWR_CTL1_[n].NS` TLP header field when generating memory read and memory write (not IMWr) TLPs.
16:12 (R/W)	FNUM	DMA Function Number. The core uses the `PCIE_DMAWR_CTL1_[n].FNUM` bit field when generating the requester ID for the memory read, memory write DMA TLP.
9 (R/W)	LLE	Linked List Enable. The `PCIE_DMAWR_CTL1_[n].LLE` bit enables (=1) and disables (=0) linked list operation.
8 (R/W)	CCS	Consumer Cycle State. The `PCIE_DMAWR_CTL1_[n].CCS` bit is used in linked list mode to synchronize the producer (software) and the consumer (DMA). For more details, see "PCS-CCS-CB-TCB Producer-Consumer Synchronization". This bit must be initialized. The DMA updates this bit during linked list operation.
6:5 (R/W)	CS	Channel Status. The `PCIE_DMAWR_CTL1_[n].CS` bit field identifies the current operational state of the DMA channel. In the case of 2'b11 (Stopped) this means the DMA has transferred all data for this channel, or the program has prematurely stopped this channel by writing to the `PCIE_DMAWR_DRBL_[n].STOP` bit field.
		0 Reserved
		1 Running (channel active and transferring data)
		2 Halted (an error condition has been detected, and the DMA has stopped this channel)
		3 Stopped
4 (R/W)	RIE	Remote Interrupt Enable. Programs must set the `PCIE_DMAWR_CTL1_[n].RIE` bit to enable the generation of the Done or Abort Remote interrupts. For more details, see "Interrupts and Error Handling". In LL mode, the DMA overwrites this with the RIE of the LL element. The `PCIE_DMAWR_CTL1_[n].RIE` bit in a LL element only enables the Done interrupt. In non-LL mode, this bit enables the Done and Abort interrupts. This field is not defined in a link LL element.

Table 29-76: PCIE_DMAWR_CTL1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	LIE	Local Interrupt Enable. Programs must set the `PCIE_DMAWR_CTL1_[n].LIE` bit to enable generation of the Done or Abort Local interrupts. For more details, see "Interrupts and Error Handling". In LL mode, the DMA overwrites this with the LIE of the LL element. The `PCIE_DMAWR_CTL1_[n].LIE` bit in a LL element only enables the Done interrupt. In non-LL mode, this bit enables the Done and Abort interrupts. This field is not defined in a link LL element.
2 (R/W)	LLP	Load Link Pointer. The `PCIE_DMAWR_CTL1_[n].LLP` bit is used in linked list mode only. This bit indicates that this linked list element is a link element, and its LL element pointer dwords are pointing to the next (non-contiguous) element. The DMA loads this field with the LLP of the linked list element.
1 (R/W)	TCB	Toggle Cycle Bit. The `PCIE_DMAWR_CTL1_[n].TCB` bit indicates to the DMA to toggle its interpretation of the CB. Used in linked list mode only. It is used to synchronize the producer (software) and the consumer (DMA). For more details, see "PCS-CCS-CB-TCB Producer-Consumer Synchronization". The DMA loads this field with the TCB of the linked list element. this field is not defined in a data LL element.
0 (R/W)	CB	Cycle Bit. The `PCIE_DMAWR_CTL1_[n].CB` bit is used in linked list mode only. It is used to synchronize the producer (software) and the consumer (DMA). For more details, see "PCS-CCS-CB-TCB Producer-Consumer Synchronization". The DMA loads this field with the CB of the linked list element.

DMA Dar High Write Channel Register

The `PCIE_DMAWR_DAR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

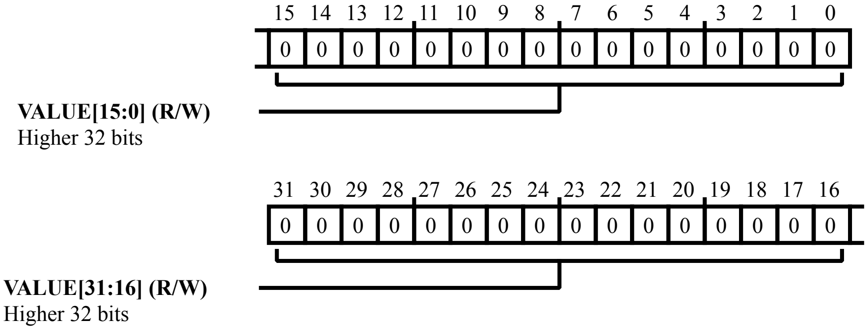

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Higher 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Higher 32 bits

Figure 29-68: PCIE_DMAWR_DAR_HI_[n] Register Diagram

Table 29-77: PCIE_DMAWR_DAR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Higher 32 bits. In LL mode, the DMA overwrites the `PCIE_DMAWR_DAR_HI_[n].VALUE` bit field with the corresponding dword of the LL element.

DMA DAR Low Write Channel Register

The `PCIE_DMAWR_DAR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

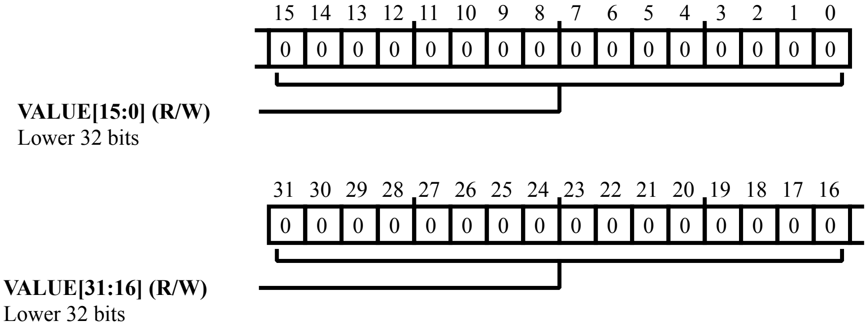

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower 32 bits

Figure 29-69: PCIE_DMAWR_DAR_LO_[n] Register Diagram

Table 29-78: PCIE_DMAWR_DAR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower 32 bits. The `PCIE_DMAWR_DAR_LO_[n].VALUE` bit field indicates the next address to be written to. The DMA increments the DAR as the DMA transfer progresses. The DAR is always dword aligned even if the transfer size is byte aligned. In LL mode, the DMA overwrites this with the corresponding dword of the LL element. The DAR is the address of the remote memory.

DMA Write Done IMWr Interrupt Address High Register

The `PCIE_DMAWR_DONE_IMWR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power on. Programs must initialize every field as the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

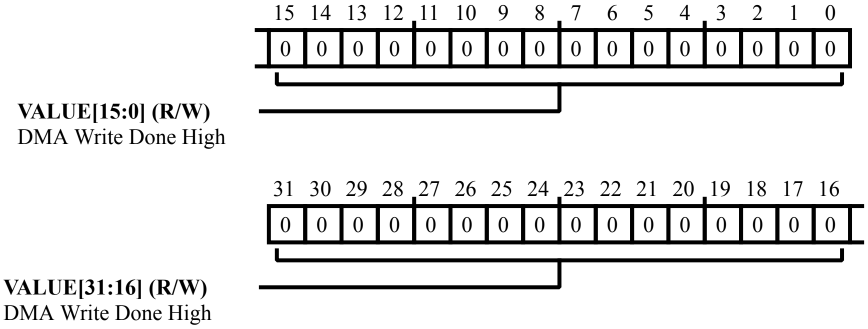

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Write Done High
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Write Done High

Figure 29-70: PCIE_DMAWR_DONE_IMWR_HI_[n] Register Diagram

Table 29-79: PCIE_DMAWR_DONE_IMWR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Write Done High. The DMA uses the `PCIE_DMAWR_DONE_IMWR_HI_[n].VALUE` bit field to generate bits [63:32] of the address field for the Done IMWr TLP.

DMA Write Done IMWr Address Low Register

The `PCIE_DMAWR_DONE_IMWR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power on. Therefore, you must program and initialize every field as the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

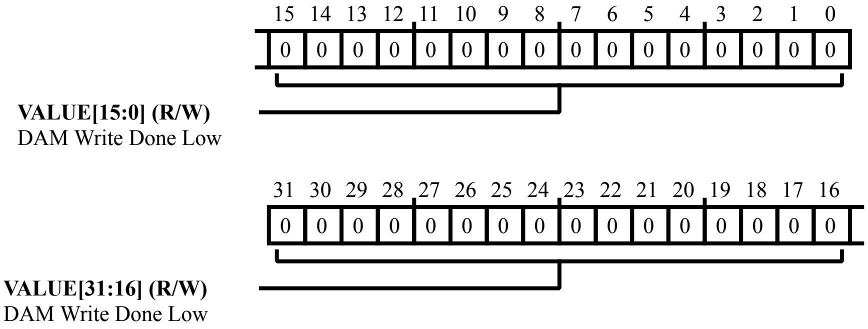

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DAM Write Done Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DAM Write Done Low

Figure 29-71: PCIE_DMAWR_DONE_IMWR_LO_[n] Register Diagram

Table 29-80: PCIE_DMAWR_DONE_IMWR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DAM Write Done Low. The DMA uses the `PCIE_DMAWR_DONE_IMWR_LO_[n].VALUE` bit field to generate bits [31:0] of the address field for the Done IMWr TLP. Bits [1:0] must be 00 as this address must be dword aligned.

DMA Write Doorbell Register

The `PCIE_DMAWR_DRBL_[n]` register contains bits that set a channel number to start the DMA write transfer for that channel and to stop a DMA transfer.

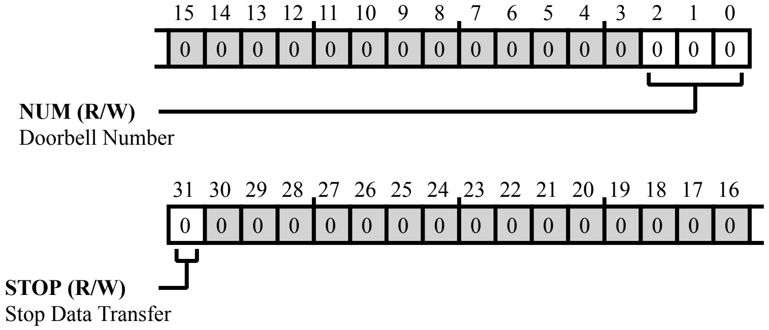

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NUM (R/W)
Doorbell Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STOP (R/W)
Stop Data Transfer

Figure 29-72: PCIE_DMAWR_DRBL_[n] Register Diagram

Table 29-81: PCIE_DMAWR_DRBL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	STOP	Stop Data Transfer. The `PCIE_DMAWR_DRBL_[n].STOP` bit is set in conjunction with the `PCIE_DMAWR_DRBL_[n].NUM` field. The DMA write channel stops issuing requests, sets the channel status to Stopped, and asserts the Abort interrupt if it is enabled. Before setting this bit, the program must read the `PCIE_DMAWR_CTL1_[n].CS` bit field to ensure that the write channel is Running (transferring data).
2:0 (R/W)	NUM	Doorbell Number. The `PCIE_DMAWR_DRBL_[n].NUM` bit field provides the channel number to this register to start the DMA write transfer for that channel. The DMA detects a write to this register field even if the value of this field does not change. Programs do not need to toggle or write any other value to this register to start a new transfer. The range of this field is 0x0 to 0x7, and 0x0 corresponds to channel 0.

DMA Write Engine Enable Register

The `PCIE_DMAWR_EN_[n]` register enables the DMA write engine.

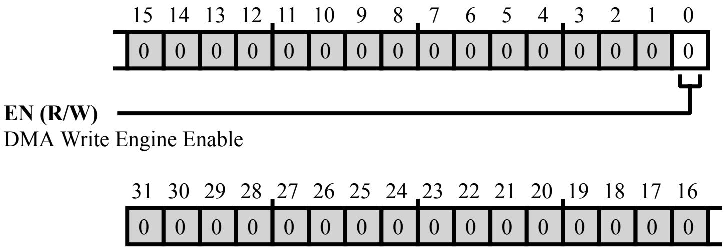


Figure 29-73: PCIE_DMAWR_EN_[n] Register Diagram

Table 29-82: PCIE_DMAWR_EN_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	EN	DMA Write Engine Enable. When set (=1) the `PCIE_DMAWR_EN_[n].EN` bit enables the DMA write engine. For normal operation, set this bit to 1 before performing any other software setup actions. Programs do not need to toggle or rewrite to this bit during normal operation. Programs should set the `PCIE_DMAWR_EN_[n].EN` bit to 0 to perform a soft reset the DMA controller write logic. There are three reasons for resetting the DMA controller write logic • The `PCIE_DMAWR_ISTAT_[n].ABRT` bit is set, and any of the bits is in the `PCIE_DMAWR_ERRSTAT_[n]` register are set. Resetting the DMA controller write logic re-initializes the control logic, ensuring that the next DMA write transfer is executed successfully. • The program has executed the procedure outlined in the `PCIE_DMAWR_DRBL_[n].STOP` bit description, after which, the `PCIE_DMAWR_ISTAT_[n].ABRT` bit is set and the `PCIE_DMAWR_CTL1_[n].CS` bit field is set to Stopped. Resetting the DMA controller write logic re-initializes the control logic ensuring that the next DMA write transfer is executed successfully. • During software development, when the DMA write engine is programmed incorrectly. To perform a soft reset the DMA controller write logic: • De-assert the DMA write engine enable bit. • Wait for the DMA to complete any in-progress TLP transfer, by waiting until a read on the DMA write engine enable bit returns a 0. • Assert the DMA write engine enable bit. This soft reset does not clear the DMA configuration registers. The DMA write transfer does not start until you write to the `PCIE_DMAWR_DRBL_[n].STOP` bit.

DMA Write Error Status Register

The `PCIE_DMAWR_ERRSTAT_[n]` register reports various write errors.

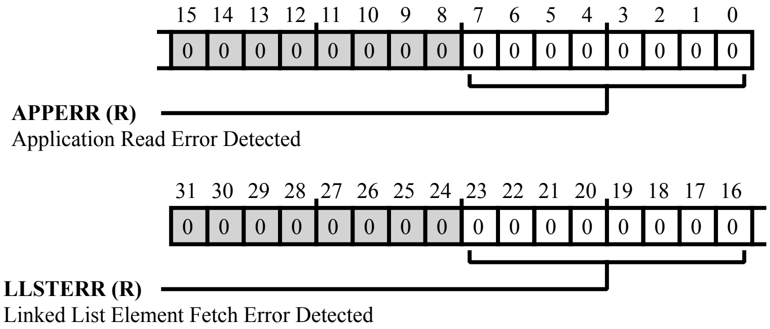

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
APPERR (R)
Application Read Error Detected
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LLSTERR (R)
Linked List Element Fetch Error Detected

Figure 29-74: PCIE_DMAWR_ERRSTAT_[n] Register Diagram

Table 29-83: PCIE_DMAWR_ERRSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	LLSTERR	Linked List Element Fetch Error Detected. The `PCIE_DMAWR_ERRSTAT_[n].LLSTERR` bit provides notice that the DMA write channel has received an error response. This response comes from the peripheral or SCB bus (or the RTRGT1 interface when the peripheral or SCB Bridge is not used) while reading a linked list element from local memory. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enabling: For details, see "Interrupts and Error Handling". • Masking: The DMA write interrupt Mask register has no effect on this register. • Clearing: Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the "DMA Write Interrupt Clear Register" (DMA_WR_INT_CLEAR_OFF) to clear this error bit.
7:0 (R/NW)	APPERR	Application Read Error Detected. The `PCIE_DMAWR_ERRSTAT_[n].APPERR` bit provides notice that the DMA write channel has received an error response. This response comes from the peripheral or SCB bus (or the RTRGT1 interface when the peripheral or SCB Bridge is not used) while reading data from it. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. • Enabling: For details, see "Interrupts and Error Handling". • Masking: The DMA write interrupt Mask register has no effect on this register. • Clearing: Programs must write a 1'b1 to the corresponding channel bit in the Abort interrupt field of the "DMA Write Interrupt Clear Register" (DMA_WR_INT_CLEAR_OFF) to clear this error bit.

DMA Write Interrupt Clear Register

The `PCIE_DMAWR_ICLR_[n]` register provides bits that all the program to clear interrupts from the `PCIE_DMAWR_ISTAT_[n].DONE` and `PCIE_DMAWR_ISTAT_[n].DONE` fields. Each bit corresponds to a DMA channel.

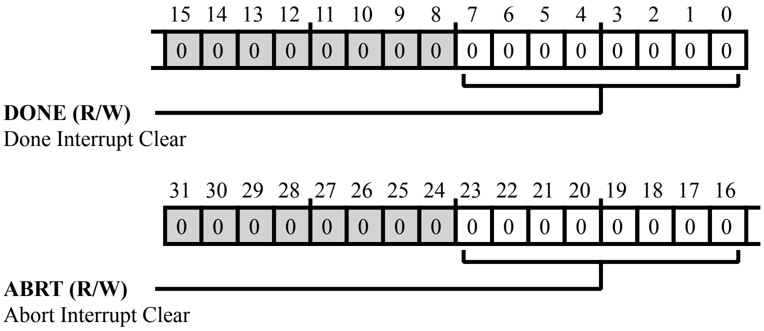

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DONE (R/W)
Done Interrupt Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABRT (R/W)
Abort Interrupt Clear

Figure 29-75: PCIE_DMAWR_ICLR_[n] Register Diagram

Table 29-84: PCIE_DMAWR_ICLR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Clear. Programs must write a 1'b1 to the `PCIE_DMAWR_ICLR_[n].ABRT` bit field clear the corresponding bit in the `PCIE_DMAWR_ISTAT_[n].ABRT` field of the DMA write interrupt status register. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Reading from this self-clearing register field always returns a 0.
7:0 (R/W)	DONE	Done Interrupt Clear. Programs must write a 1'b1 to the `PCIE_DMAWR_ICLR_[n].DONE` bit field clear the corresponding bit in the `PCIE_DMAWR_ISTAT_[n].DONE` field of the DMA write interrupt status register. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Reading from this self-clearing register field always returns a 0.

DMA Write Interrupt Mask Register

The `PCIE_DMAWR_IMSK_[n]` register is used to mask interrupts in the `PCIE_DMAWR_ISTAT_[n]` register.

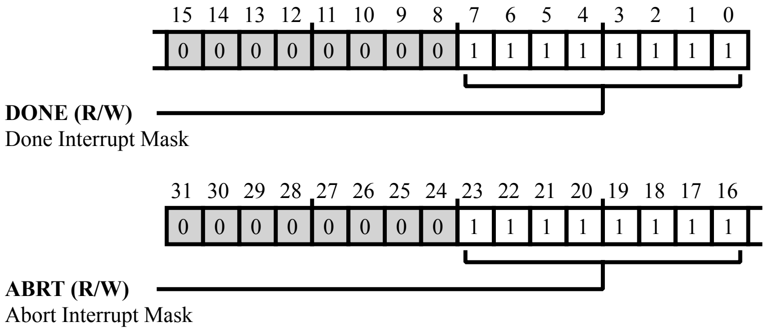

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
DONE (R/W)
Done Interrupt Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
ABRT (R/W)
Abort Interrupt Mask

Figure 29-76: PCIE_DMAWR_IMSK_[n] Register Diagram

Table 29-85: PCIE_DMAWR_IMSK_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Mask. The `PCIE_DMAWR_IMSK_[n].ABRT` bit field prevents the `PCIE_DMAWR_ISTAT_[n].ABRT` status field in the DMA from asserting the edma_int output. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0.
7:0 (R/W)	DONE	Done Interrupt Mask. The `PCIE_DMAWR_IMSK_[n].DONE` bit field prevents the `PCIE_DMAWR_ISTAT_[n].DONE` status field in the DMA from asserting the edma_int output. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0.

DMA Write Interrupt Status Register

The PCIE_DMAWR_ISTAT_[n] register provides status information on DMA writes.

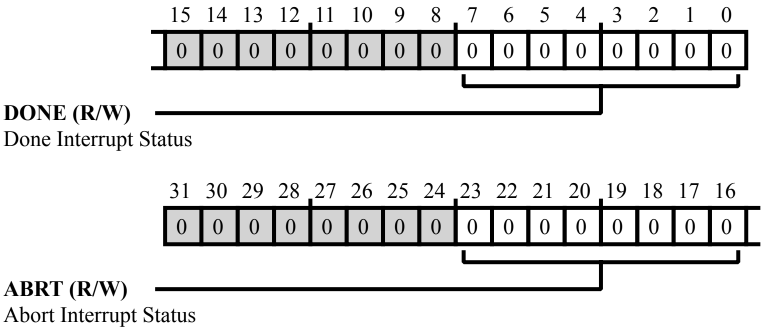


Figure 29-77: PCIE_DMAWR_ISTAT_[n] Register Diagram

Table 29-86: PCIE_DMAWR_ISTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	ABRT	Abort Interrupt Status. The `PCIE_DMAWR_ISTAT_[n].ABRT` indicates the DMA write channel has detected an error or the transfer was stopped manually. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Programs can read the DMA Read Error Status Low Register and DMA Read Error Status High Register to determine the source of the error. The DMA read interrupt Mask register has no effect on this bit field. Programs must write a 1'b1 to the corresponding channel bit in the DMA read interrupt Clear register to clear this interrupt bit. Programs can write to this bit field to emulate interrupt generation, during software or hardware testing. A write to the address triggers an interrupt, but the DMA does not set the Done or Abort bits in this bit field.
7:0 (R/W)	DONE	Done Interrupt Status. The `PCIE_DMAWR_ISTAT_[n].DONE` bit field indicates the DMA write channel has successfully completed the DMA transfer. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. - Enabling: For details, see "Interrupts and Error Handling". - Masking: The DMA write interrupt Mask register has no effect on this register. - Clearing: You must write a 1'b1 to the corresponding channel bit in the DMA write interrupt Clear register to clear this interrupt bit. Note: You can write to this register to emulate interrupt generation, during software or hardware testing. A write to the address triggers an interrupt, but the DMA does not set the Done or Abort bits in this register.

DMA LLP High Write Channel Register

The `PCIE_DMAWR_LLP_HI_[n]` register is implemented in RAM whose contents are uninitialized after power on. This register value must always be initialized because the default is undefined. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

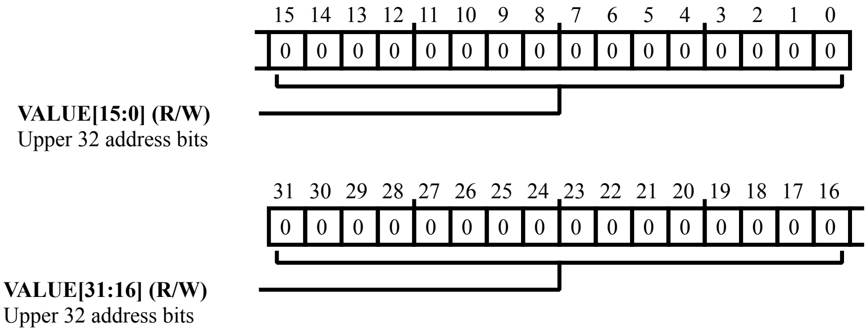

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Upper 32 address bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Upper 32 address bits

Figure 29-78: PCIE_DMAWR_LLP_HI_[n] Register Diagram

Table 29-87: PCIE_DMAWR_LLP_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Upper 32 address bits. The `PCIE_DMAWR_LLP_HI_[n].VALUE` bit field contains the upper 32 bits of the address of the linked list transfer list in local memory. Used in linked list mode only. The `PCIE_DMAWR_LLP_HI_[n].VALUE` bit field is updated by the DMA to point to the next element in the transfer list as elements are consumed.

DMA LLP Low Write Channel Register

The `PCIE_DMAWR_LLP_LO_[n]` register is implemented in RAM whose contents are uninitialized after power on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

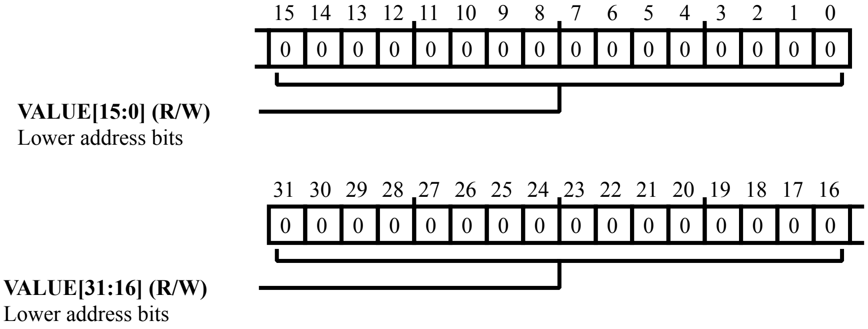

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower address bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower address bits

Figure 29-79: PCIE_DMAWR_LLP_LO_[n] Register Diagram

Table 29-88: PCIE_DMAWR_LLP_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower address bits. The `PCIE_DMAWR_LLP_LO_[n].VALUE` bit field contains the lower bits of the address of the linked list transfer list in local memory. Used in linked list mode only. The `PCIE_DMAWR_LLP_LO_[n].VALUE` bit field is updated by the DMA to point to the next element in the transfer list after the previous element is consumed. • When the current element is a data element; this field is incremented by 6. • When the current element is a link element; this field is overwritten by the LL Element Pointer of the element.

DMA Write Linked List Error Enable Register

The `PCIE_DMAWR_LLSTERR_EN_[n]` register provides bits in the LL element that enable the channel done and abort interrupts (local and remote).

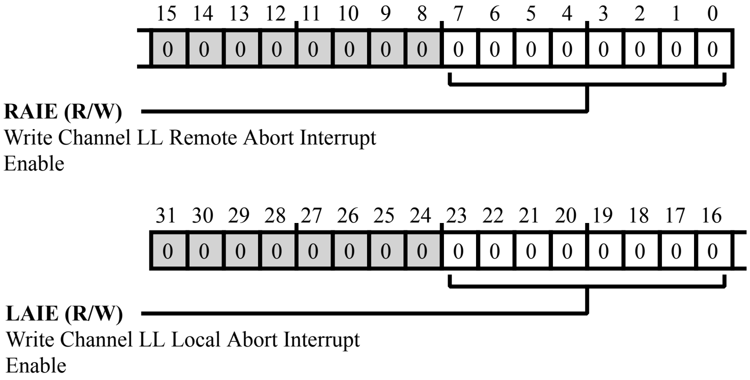

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RAIE (R/W)
Write Channel LL Remote Abort Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LAIE (R/W)
Write Channel LL Local Abort Interrupt Enable

Figure 29-80: PCIE_DMAWR_LLSTERR_EN_[n] Register Diagram

Table 29-89: PCIE_DMAWR_LLSTERR_EN_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	LAIE	Write Channel LL Local Abort Interrupt Enable. The `PCIE_DMAWR_LLSTERR_EN_[n].LAIE` bit field enables the write channel local abort interrupt. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Used in linked list mode only.
7:0 (R/W)	RAIE	Write Channel LL Remote Abort Interrupt Enable. The `PCIE_DMAWR_LLSTERR_EN_[n].RAIE` bit field enables the write channel remote abort interrupt. Each bit corresponds to a DMA channel. Bit [0] corresponds to channel 0. Used in linked list mode only.

DMA Write Posted Request Deadlock Timer Register

The `PCIE_DMAWR_PREQ_TMR_[n]` register sets the time-out value for the write channel DMA transmit write posted request deadlock timer. This timer is only enabled if there is a pending TX non-DMA completion blocked behind the DMA posted request. When the timer expires, the core pulls back the pending DMA posted request and allows non-DMA completions to pass. When all completions have been transmitted, the SCB bridge replays the DMA posted requests. If the DMA posted TLPs are not making progress and posted credits have not been returned within this time frame, then is safe to assume that the DMA posted TLPs are blocked.

The timeout value increments in steps of 2 usec. The default value is 30 usec based on the PCIe specification's maximum latency between posted credit updates from the remote link partner. The timeout value must be less than the remote link partner's completion timeout value to avoid the remote link partner timing out its non-posted requests. The minimum completion timeout value is 50 usec but is typically in the order of tens of milliseconds. If the deadlock timeout value is too close to the completion timeout value then you can reprogram this timer and decrease the timeout value.

To allow sufficient time for posted credits to be updated and avoid unnecessary flushing of DMA posted requests, the recommended minimum timer value is 10 usec.

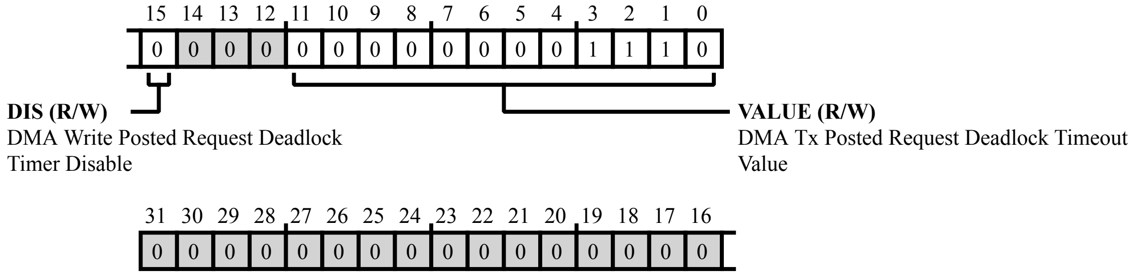


Figure 29-81: PCIE_DMAWR_PREQ_TMR_[n] Register Diagram

Table 29-90: PCIE_DMAWR_PREQ_TMR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	DIS	DMA Write Posted Request Deadlock Timer Disable. The `PCIE_DMAWR_PREQ_TMR_[n].DIS` bit disables the deadlock timer.
11:0 (R/W)	VALUE	DMA Tx Posted Request Deadlock Timeout Value. The `PCIE_DMAWR_PREQ_TMR_[n].VALUE` bit field sets the DMA Tx posted request deadlock timeout value. In normal operation the receiver must process a posted request within 10 usec. The default value is 30 usec and encoding is 0 = 2us, 1 = 4us, ..., 4095 = 8192us.

DMA SAR High Write Channel Register

The `PCIE_DMAWR_SAR_HI_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

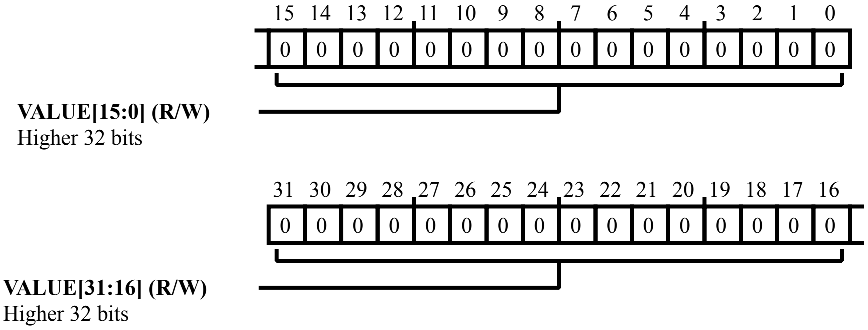

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Higher 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Higher 32 bits

Figure 29-82: PCIE_DMAWR_SAR_HI_[n] Register Diagram

Table 29-91: PCIE_DMAWR_SAR_HI_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Higher 32 bits. In LL mode, the DMA overwrites the `PCIE_DMAWR_SAR_HI_[n].VALUE` bit field with the corresponding dword of the LL element.

DMA SAR Low Write Channel Register

The `PCIE_DMAWR_SAR_LO_[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

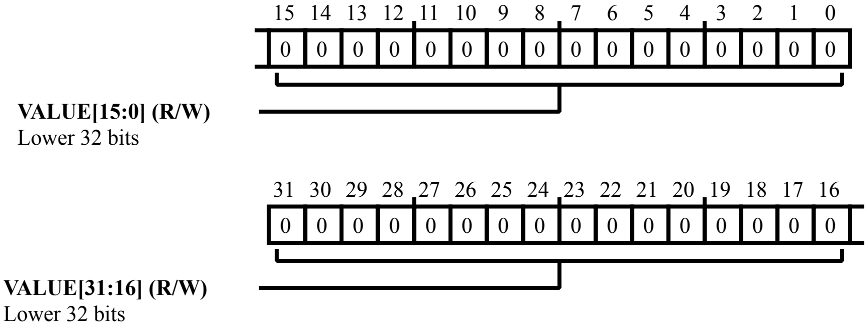

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Lower 32 bits

Figure 29-83: PCIE_DMAWR_SAR_LO_[n] Register Diagram

Table 29-92: PCIE_DMAWR_SAR_LO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower 32 bits. The `PCIE_DMAWR_SAR_LO_[n].VALUE` register indicates the next address to be read from. The DMA increments the SAR as the DMA transfer progresses. In LL mode, the DMA overwrites this with the corresponding dword of the LL element. The SAR is the address of the local memory.

DMA Transfer Size Write Channel Register

The `PCIE_DMAWR_XFRSZ_0[n]` register is implemented in RAM whose contents are uninitialized after power-on. All fields marked Reserved must be programmed to 1'b0. This register is not affected by any of the reset signals.

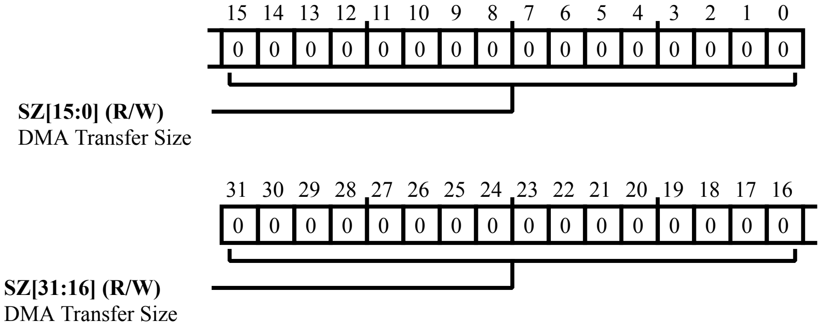

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SZ[15:0] (R/W)
DMA Transfer Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SZ[31:16] (R/W)
DMA Transfer Size

Figure 29-84: PCIE_DMAWR_XFRSZ_0[n] Register Diagram

Table 29-93: PCIE_DMAWR_XFRSZ_0[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	SZ	DMA Transfer Size. The `PCIE_DMAWR_XFRSZ_0[n].SZ` bit field configures the size of the DMA transfer. The maximum DMA transfer size is 4Gbytes. The minimum transfer size is one byte (0x1). This field is automatically decremented by the DMA as the DMA write and read channel transfer progresses. This field indicates the number bytes remaining to be transferred. When all bytes are successfully transferred the current transfer size is zero. In LL mode, the DMA overwrites the `PCIE_DMAWR_XFRSZ_0[n].SZ` bit field with the corresponding dword of the LL element.

DMA Number of Channels Register

The `PCIE_DMA_CTL_[n]` register reports the number of read and write channels that are configured.

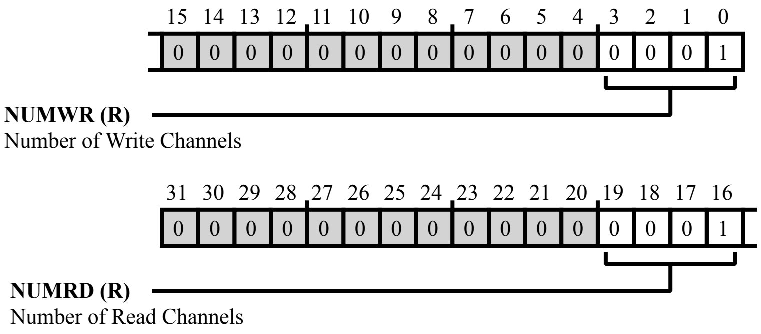

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
NUMWR (R)
Number of Write Channels
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
NUMRD (R)
Number of Read Channels

Figure 29-85: PCIE_DMA_CTL_[n] Register Diagram

Table 29-94: PCIE_DMA_CTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/NW)	NUMRD	Number of Read Channels. The `PCIE_DMA_CTL_[n].NUMRD` bit field reports the number of read channels the DMA controller is configured to support.
3:0 (R/NW)	NUMWR	Number of Write Channels. The `PCIE_DMA_CTL_[n].NUMWR` bit field reports the number of write channels that the DMA controller is configured to support.

DMA Channel Context Index Register

The `PCIE_DMA_VWPRT_SEL_[n]` register programs the DMA registers using an indirect addressing scheme to reduce the address footprint in the PCI Express extended configuration space.

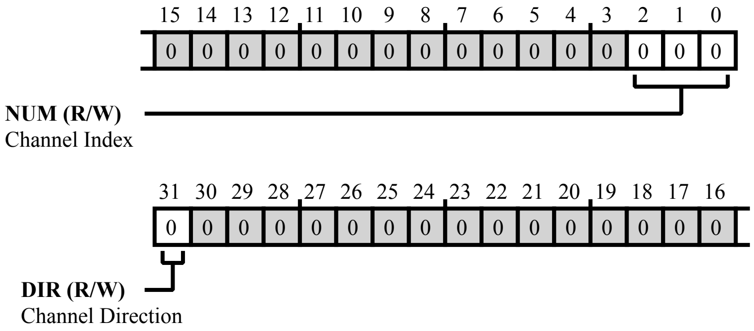

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NUM (R/W)
Channel Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIR (R/W)
Channel Direction

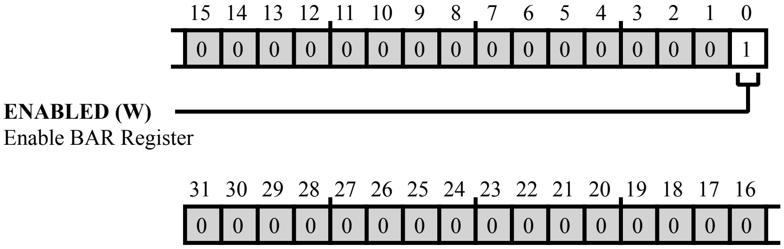
Figure 29-86: PCIE_DMA_VWPRT_SEL_[n] Register Diagram

Table 29-95: PCIE_DMA_VWPRT_SEL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	DIR	Channel Direction. The `PCIE_DMA_VWPRT_SEL_[n].DIR` bit field sets the channel direction. Selects a write channel or read channel context.	
		0	DMA Write
		1	DMA Read
2:0 (R/W)	NUM	Channel Index. The `PCIE_DMA_VWPRT_SEL_[n].NUM` bit field sets the channel number. This field must be set to 0x0.	

Endpoint Base Address Mask Register 0

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but this register can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-87: PCIE_EP_BAR0_MASK[n] Register Diagram

Table 29-96: PCIE_EP_BAR0_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR0_MASK[n].ENABLED` bit indicates memory space.

Endpoint Base Address Register 0

The PCIE_EP_BAR0_[n] register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

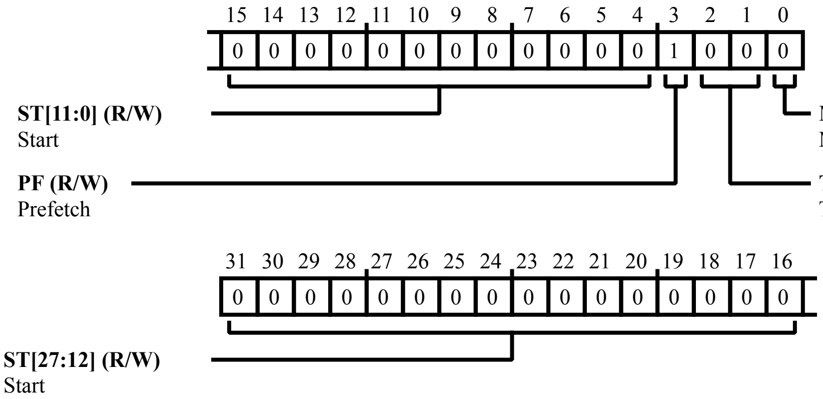

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
PF (R/W)
Prefetch
MIO (R/W)
Memory I/O
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-88: PCIE_EP_BAR0_[n] Register Diagram

Table 29-97: PCIE_EP_BAR0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The PCIE_EP_BAR0_[n].ST bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The PCIE_EP_BAR0_[n].PF bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/W)	MIO	Memory I/O. The PCIE_EP_BAR0_[n].MIO bit indicates memory space.	
		0	Memory space
		1	I/O space

Endpoint Base Address Mask Register 1

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but this register (`PCIE_EP_BAR1_MASK[n]`) can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.

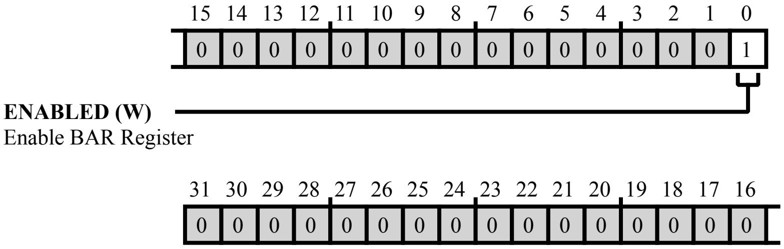

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-89: PCIE_EP_BAR1_MASK[n] Register Diagram

Table 29-98: PCIE_EP_BAR1_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR1_MASK[n].ENABLED` bit indicates memory space.	
		0	Register Disabled
		1	Register Enabled

Endpoint Base Address Register 1

The `PCIE_EP_BAR1_[n]` register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

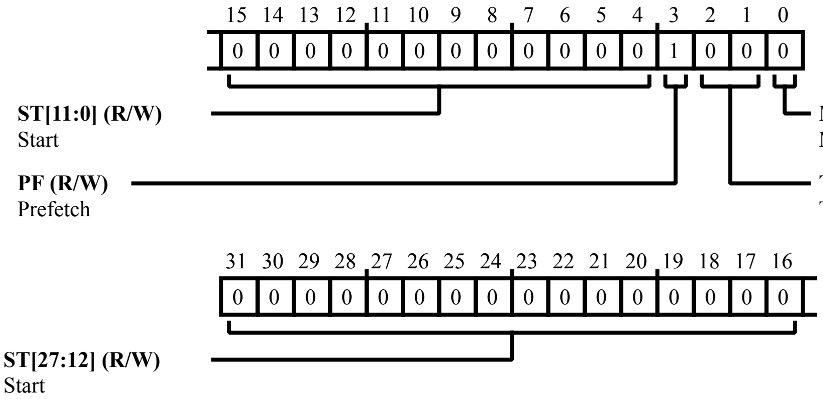

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
MIO (R/W)
Memory I/O
PF (R/W)
Prefetch
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-90: PCIE_EP_BAR1_[n] Register Diagram

Table 29-99: PCIE_EP_BAR1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The `PCIE_EP_BAR1_[n].ST` bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The `PCIE_EP_BAR1_[n].PF` bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/W)	MIO	Memory I/O. The `PCIE_EP_BAR1_[n].MIO` bit selects the memory space indicator.	
		0	Memory space
		1	I/O space

Endpoint Base Address Mask Register 2

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but but this register (`PCIE_EP_BAR2_MASK[n]`) can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.

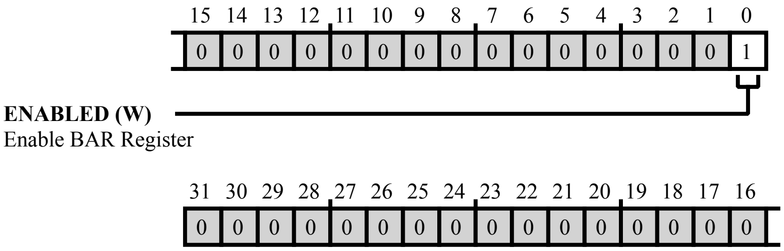

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-91: PCIE_EP_BAR2_MASK[n] Register Diagram

Table 29-100: PCIE_EP_BAR2_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR2_MASK[n].ENABLED` bit indicates memory space.	
		0	Register Disabled
		1	Register Enabled

Endpoint Base Address Register 2

The `PCIE_EP_BAR2_[n]` register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

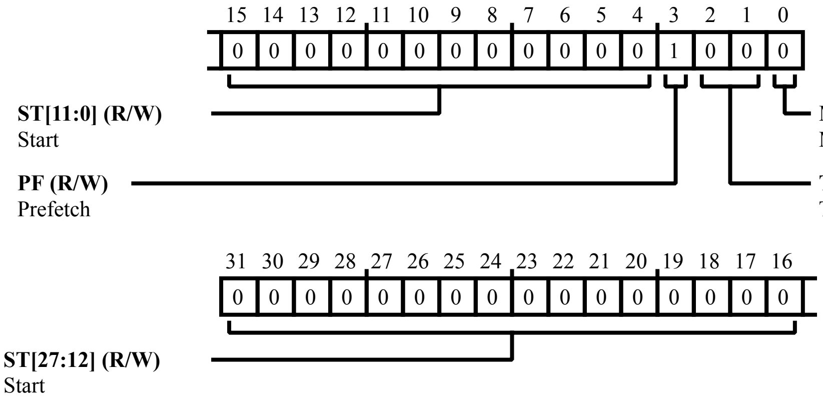

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
PF (R/W)
Prefetch
MIO (R/W)
Memory I/O
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-92: PCIE_EP_BAR2_[n] Register Diagram

Table 29-101: PCIE_EP_BAR2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The `PCIE_EP_BAR2_[n].ST` bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The `PCIE_EP_BAR2_[n].PF` bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/W)	MIO	Memory I/O. The `PCIE_EP_BAR2_[n].MIO` bit indicates memory space.	
		0	Memory space
		1	I/O space

Endpoint Base Address Mask Register 3

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but this register (`PCIE_EP_BAR3_MASK[n]`) can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.

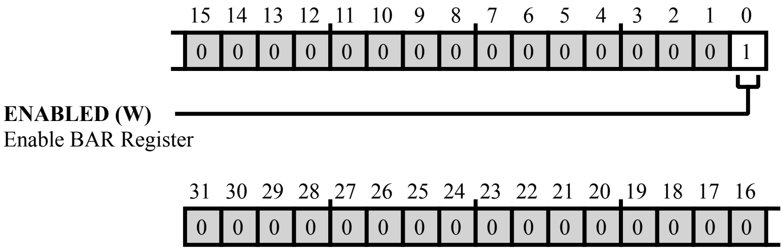

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-93: PCIE_EP_BAR3_MASK[n] Register Diagram

Table 29-102: PCIE_EP_BAR3_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR3_MASK[n].ENABLED` bit indicates memory space.	
		0	Register Disabled
		1	Register Enabled

Endpoint Base Address Register 3

The `PCIE_EP_BAR3_[n]` register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

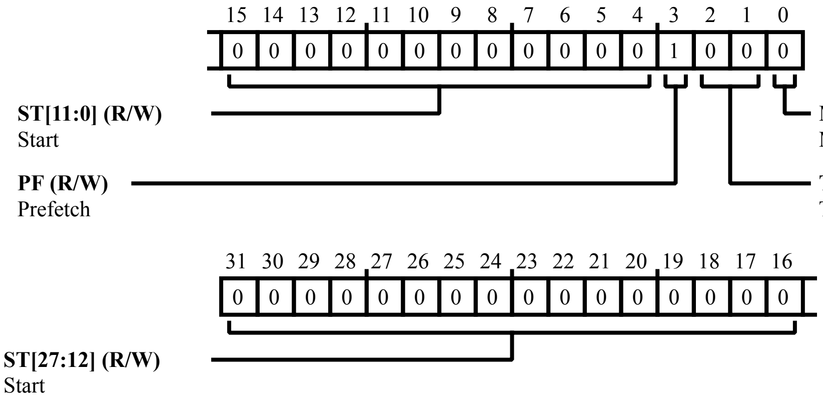

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
PF (R/W)
Prefetch
MIO (R/W)
Memory I/O
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-94: PCIE_EP_BAR3_[n] Register Diagram

Table 29-103: PCIE_EP_BAR3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The `PCIE_EP_BAR3_[n].ST` bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The `PCIE_EP_BAR3_[n].PF` bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/W)	MIO	Memory I/O. The `PCIE_EP_BAR3_[n].MIO` bit indicates memory space.	
		0	Memory space
		1	I/O space

Endpoint Base Address Mask Register 4

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but but this register (`PCIE_EP_BAR4_MASK[n]`) can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.

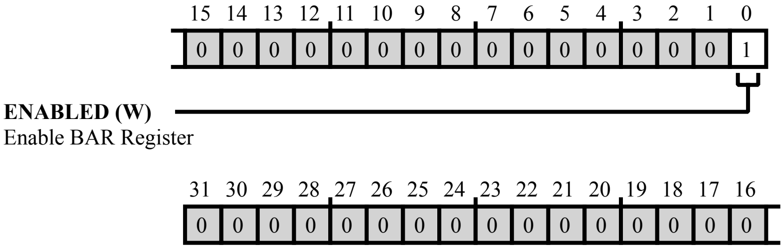

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-95: PCIE_EP_BAR4_MASK[n] Register Diagram

Table 29-104: PCIE_EP_BAR4_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR4_MASK[n].ENABLED` bit indicates memory space.	
		0	Register Disabled
		1	Register Enabled

Endpoint Base Address Register 4

The `PCIE_EP_BAR4_[n]` register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

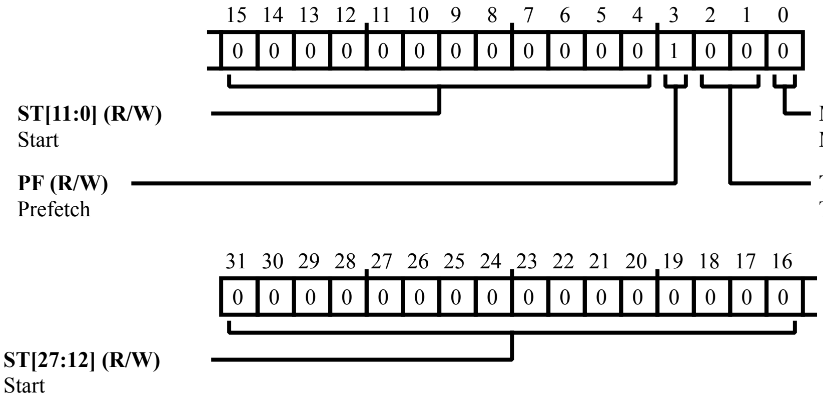

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
PF (R/W)
Prefetch
MIO (R/W)
Memory I/O
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-96: PCIE_EP_BAR4_[n] Register Diagram

Table 29-105: PCIE_EP_BAR4_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The `PCIE_EP_BAR4_[n].ST` bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The `PCIE_EP_BAR4_[n].PF` bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/W)	MIO	Memory I/O. The `PCIE_EP_BAR4_[n].MIO` bit indicates memory space.	
		0	Memory space
		1	I/O space

Endpoint Base Address Mask Register 5

The mask for this BAR exists as a shadow register at this address. The `PCIE_EP_BAR0_[n]` register cannot be disabled but but this register (`PCIE_EP_BAR5_MASK[n]`) can be modified to change the size of memory advertised by `PCIE_EP_BAR0_[n]` to the host processor. Writing to this register is enabled if the `PCIE_MISC_CTL1_[n].RODBIWREN` bit is set.

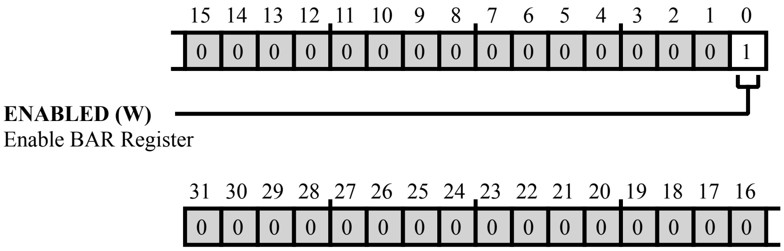

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ENABLED (W)
Enable BAR Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-97: PCIE_EP_BAR5_MASK[n] Register Diagram

Table 29-106: PCIE_EP_BAR5_MASK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RX/W)	ENABLED	Enable BAR Register. The `PCIE_EP_BAR5_MASK[n].ENABLED` bit indicates memory space.	
		0	Register Disabled
		1	Register Enabled

End Point Base Address Register 5

The `PCIE_EP_BAR5_[n]` register forms bits [31:x] of the start address of the address region to be translated. This register also determines the use of the prefetch capability. For more information, see the PCI Express Base Specification, Rev. 3.0.

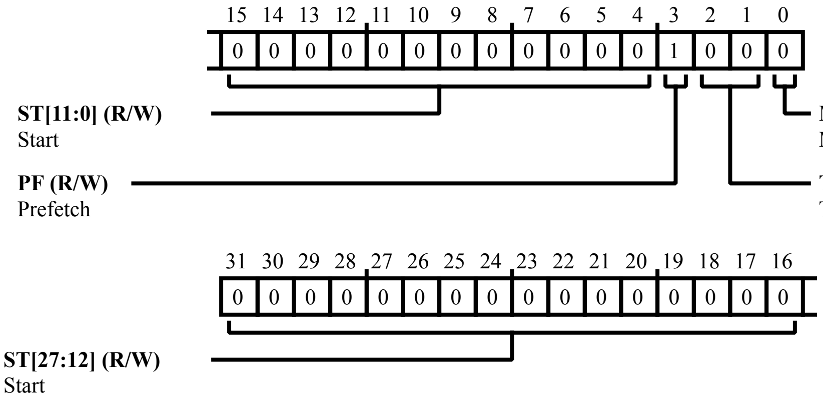

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
ST[11:0] (R/W)
Start
PF (R/W)
Prefetch
MIO (R)
Memory I/O
TYPE (R/W)
Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ST[27:12] (R/W)
Start

Figure 29-98: PCIE_EP_BAR5_[n] Register Diagram

Table 29-107: PCIE_EP_BAR5_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:4 (R/W)	ST	Start. The `PCIE_EP_BAR5_[n].ST` bit field forms bits [31:x] of the start address of the address region to be translated.	
3 (R/W)	PF	Prefetch. The `PCIE_EP_BAR5_[n].PF` bit is usually set to one if there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables, and host bridges can merge processor writes into this range without causing errors. Bit must be set to zero otherwise.	
2:1 (R/W)	TYPE	Type.	
		0	Locate anywhere in 32-bit access space
		1	Reserved
		2	Locate anywhere in 62-bit access space
		3	Reserved
0 (R/NW)	MIO	Memory I/O. The `PCIE_EP_BAR5_[n].MIO` bit indicates memory space.	
		0	Memory space
		1	I/O space

Capability Pointer Register

The PCIE_EP_CAPBPTR_[n] register points to the first item in the list of capabilities.

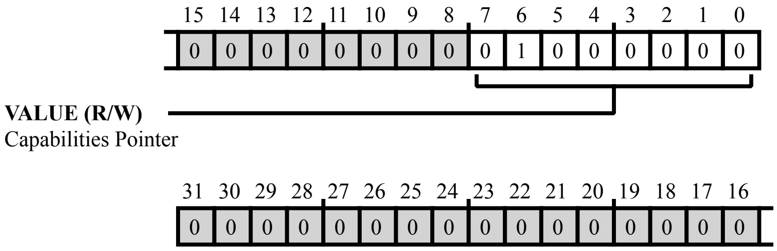

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
VALUE (R/W)
Capabilities Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-99: PCIE_EP_CAPBPTR_[n] Register Diagram

Table 29-108: PCIE_EP_CAPBPTR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Capabilities Pointer. The `PCIE_EP_CAPBPTR_[n].VALUE` bit field points to the first item in the list of capabilities. Each capability in the list consists of an 8-bit ID field assigned by the PCI SIG, an 8 bit pointer in configuration space to the next capability, and some number of additional registers immediately following the pointer to implement that capability. Each capability must be DWORD aligned. The bottom two bits of all pointers (including the initial pointer at 0x34) are reserved and must be implemented as 00b although software must mask them to allow for future uses of these bits. A pointer value of 0x00 is used to indicate the last capability in the list

Class Code and Revision ID Register

The `PCIE_EP_CCRID_[n]` register identifies the function of a device.

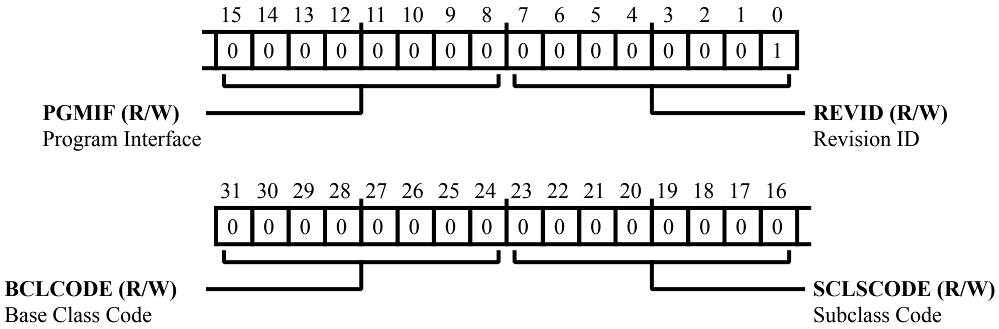

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
PGMIF (R/W)
Program Interface
REVID (R/W)
Revision ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BCLCODE (R/W)
Base Class Code
SCLSCODE (R/W)
Subclass Code

Figure 29-100: PCIE_EP_CCRID_[n] Register Diagram

Table 29-109: PCIE_EP_CCRID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BCLCODE	Base Class Code. The `PCIE_EP_CCRID_[n].BCLCODE` bit field broadly classifies the type of function the device performs.
23:16 (R/W)	SCLSCODE	Subclass Code. The `PCIE_EP_CCRID_[n].SCLSCODE` bit field identifies more specifically the function of the device.
15:8 (R/W)	PGMIF	Program Interface. The `PCIE_EP_CCRID_[n].PGMIF` bit field identifies a specific register-level programming interface (if any) so that device independent software can interact with the device.
7:0 (R/W)	REVID	Revision ID. The `PCIE_EP_CCRID_[n].REVID` bit field is the device-specific revision identifier.

End Point Configuration Register

The `PCIE_EP_CFG_[n]` register configures the header type, cache line size, and latency timer.

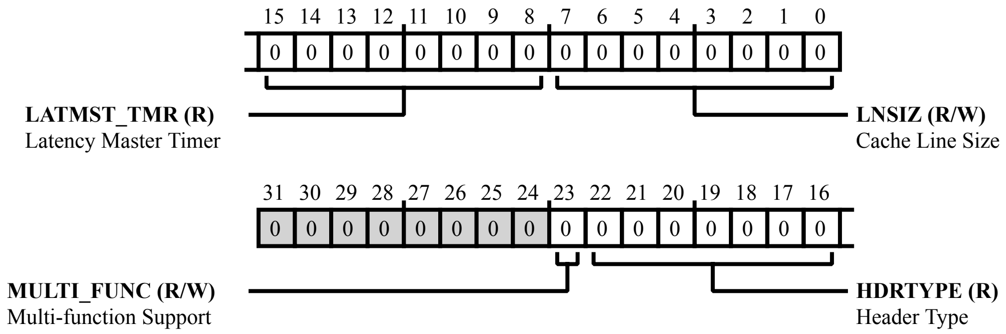

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LATMST_TMR (R)
Latency Master Timer
LNSIZ (R/W)
Cache Line Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MULTI_FUNC (R/W)
Multi-function Support
HDRTYPE (R)
Header Type

Figure 29-101: PCIE_EP_CFG_[n] Register Diagram

Table 29-110: PCIE_EP_CFG_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	MULTI_FUNC	Multi-function Support. The `PCIE_EP_CFG_[n].MULTI_FUNC` bit must be set to zero.
22:16 (R/NW)	HDRTYPE	Header Type. The `PCIE_EP_CFG_[n].HDRTYPE` bit field identifies the layout of the second part of the predefined header and also whether or not the device contains multiple functions. Bit 7 in this register is used to identify a multifunction device. If the bit is 0, then the device is single function. If the bit is 1, then the device has multiple functions. Bits 6 through 0 identify the layout of the second part of the predefined header.
15:8 (R/NW)	LATMST_TMR	Latency Master Timer. The `PCIE_EP_CFG_[n].LATMST_TMR` bit field specifies the value of the latency timer in units of PCI bus clocks.
7:0 (R/W)	LNSIZ	Cache Line Size. The `PCIE_EP_CFG_[n].LNSIZ` bit field specifies the system cache line size in units of DWORDS.

CardBus CIS Pointer Register

The `PCIE_EP_CRDBPTR_[n]` register is optionally used by devices that want to share silicon between CardBus and PCI.

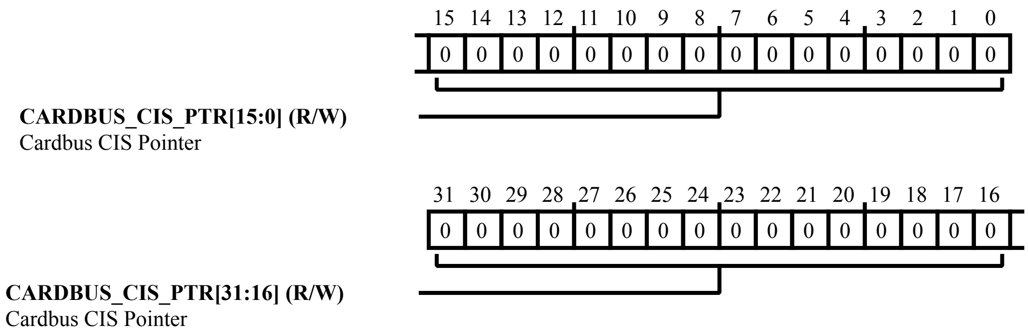

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CARDBUS_CIS_PTR[15:0] (R/W)
Cardbus CIS Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CARDBUS_CIS_PTR[31:16] (R/W)
Cardbus CIS Pointer

Figure 29-102: PCIE_EP_CRDBPTR_[n] Register Diagram

Table 29-111: PCIE_EP_CRDBPTR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CARDBUS_CIS_PTR	Cardbus CIS Pointer. The `PCIE_EP_CRDBPTR_[n].CARDBUS_CIS_PTR` bit field is used to point to the Card Information Structure (CIS) for the CardBus card. For a detailed explanation of the CIS, refer to the PCMCIA v2.10 specification.

Device Capabilities Register

The `PCIE_EP_DEVCAPB_[n]` register identifies PCI Express device Function specific capabilities.

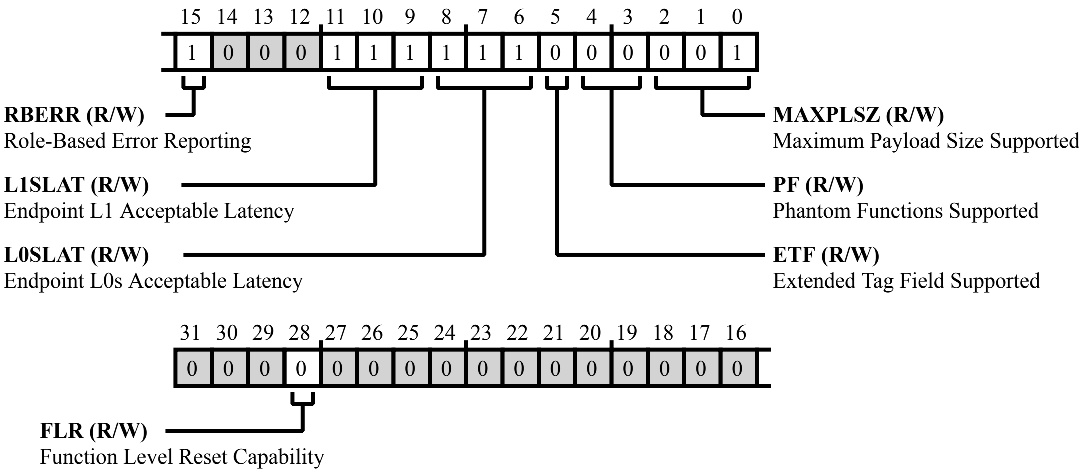

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1
RBERR (R/W)
Role-Based Error Reporting
L1SLAT (R/W)
Endpoint L1 Acceptable Latency
L0SLAT (R/W)
Endpoint L0s Acceptable Latency
MAXPLSZ (R/W)
Maximum Payload Size Supported
PF (R/W)
Phantom Functions Supported
ETF (R/W)
Extended Tag Field Supported
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FLR (R/W)
Function Level Reset Capability

Figure 29-103: PCIE_EP_DEVCAPB_[n] Register Diagram

Table 29-112: PCIE_EP_DEVCAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	FLR	Function Level Reset Capability. A value of 1 in the `PCIE_EP_DEVCAPB_[n].FLR` bit indicates that a Function supports the optional Function Level Reset mechanism. This bit applies to Endpoints only. For all other Function types this bit must be set to 0.	
15 (R/W)	RBERR	Role-Based Error Reporting. When set, the `PCIE_EP_DEVCAPB_[n].RBERR` bit indicates that the Function implements the functionality originally defined in the PCI Express Base Specification. This bit is read-only and always set to 1.	
11:9 (R/W)	L1SLAT	Endpoint L1 Acceptable Latency. The `PCIE_EP_DEVCAPB_[n].L1SLAT` bit field indicates the acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state. It is essentially an indirect measure of the Endpoints internal buffering. For Functions other than Endpoints, this field is Reserved and must be hardwired to 000.	
		0	Maximum of 1 us
		1	Maximum of 2 us
		2	Maximum of 4 us
		3	Maximum of 8 us
		4	Maximum of 16 us
		5	Maximum of 32 us

Table 29-112: PCIE_EP_DEVCAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	Maximum of 64 us
		7	No limit
8:6 (R/W)	L0SLAT	Endpoint L0s Acceptable Latency. The `PCIE_EP_DEVCAPB_[n].L0SLAT` bit field indicates the acceptable total latency that an Endpoint can withstand due to the transition from L0s state to the L0 state. It is essentially an indirect measure of the Endpoints internal buffering. For Functions other than Endpoints, this field is Reserved and must be hardwired to 000.	
		0	Maximum of 64 ns
		1	Maximum of 128 ns
		2	Maximum of 256 ns
		3	Maximum of 512 ns
		4	Maximum of 1 us
		5	Maximum of 2 us
		6	Maximum of 4 us
		7	No limit
5 (R/W)	ETF	Extended Tag Field Supported. The `PCIE_EP_DEVCAPB_[n].ETF` bit indicates the maximum supported size of the Tag field as a Requester. Note that 8-bit Tag field generation must be enabled by the Extended Tag Field Enable bit in the Device Control register of the Requester Function before 8-bit Tags can be generated by the Requester.	
		0	5-bit Tag field supported
		1	8-bit Tag field supported
4:3 (R/W)	PF	Phantom Functions Supported. The `PCIE_EP_DEVCAPB_[n].PF` bit field indicates the support for use of unclaimed Function Numbers to extend the number of outstanding transactions allowed by logically combining unclaimed Function Numbers (called Phantom Functions) with the Tag identifier.	
2:0 (R/W)	MAXPLSZ	Maximum Payload Size Supported. The `PCIE_EP_DEVCAPB_[n].MAXPLSZ` bit field indicates the maximum payload size that the Function can support for TLPs. The Functions of a multi-Function device are permitted to report different values for this field.	
		0	128 bytes max payload size
		1	256 bytes max payload size
		2	512 bytes max payload size

Table 29-112: PCIE_EP_DEVCAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	1024 bytes max payload size
		4	2048 bytes max payload size
		5	4096 bytes max payload size
		6	Reserved
		7	Reserved

Device ID and Vendor ID Register

The `PCIE_EP_ID_[n]` register indicates the assigned IDs.

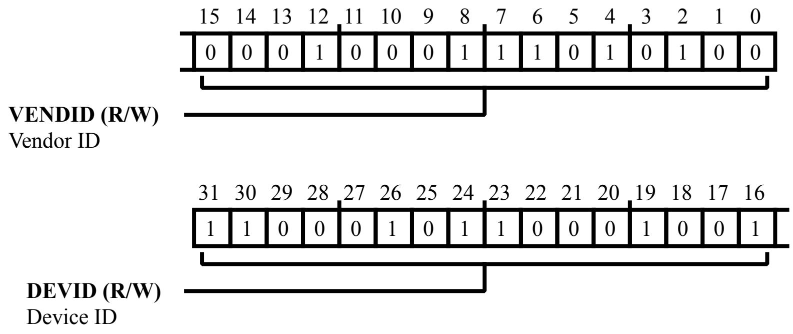

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0
VENDID (R/W)
Vendor ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1
DEVID (R/W)
Device ID

Figure 29-104: PCIE_EP_ID_[n] Register Diagram

Table 29-113: PCIE_EP_ID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DEVID	Device ID. The `PCIE_EP_ID_[n].DEVID` bit field indicates the vendor assigned field that provides a means for a vendor to classify a particular RCRB.
15:0 (R/W)	VENDID	Vendor ID. The `PCIE_EP_ID_[n].VENDID` bit field indicates the PCI-SIG assigned field that provides a means to associate an RCRB (root complex register block) with a particular vendor.

Interrupt Line and Pin Register

The `PCIE_EP_PINLN_INT_[n]` register provides interrupt and line routing information.

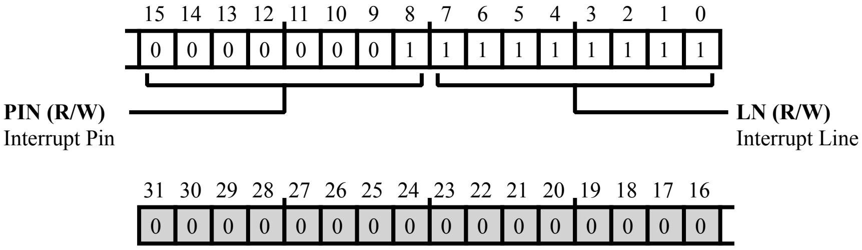

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
PIN (R/W)
Interrupt Pin
LN (R/W)
Interrupt Line
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-105: PCIE_EP_PINLN_INT_[n] Register Diagram

Table 29-114: PCIE_EP_PINLN_INT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	PIN	Interrupt Pin. The `PCIE_EP_PINLN_INT_[n].PIN` bit field identifies the legacy interrupt Message(s) the Function uses. Valid values are 0x1, 0x2, 0x3, and 0x4 that map to legacy interrupt Messages for INTA, INTB, INTC, and INTD respectively. A value of 0x0 indicates that the Function uses no legacy interrupt Message(s).
7:0 (R/W)	LN	Interrupt Line. The `PCIE_EP_PINLN_INT_[n].LN` bit field communicates interrupt line routing information. The field is read/write and must be implemented by any Function that uses an interrupt pin. Values in this bit field are programmed by system software and are system architecture specific. The Function itself does not use this value; rather the value in this field is used by device drivers and operating systems.

End Point Expansion ROM Base Address Register

The `PCIE_EP_ROMCFG_[n]` register contains bits that indicate the upper 21 bits of the Expansion ROM base address and bits that control whether or not the device accepts accesses to its expansion ROM.

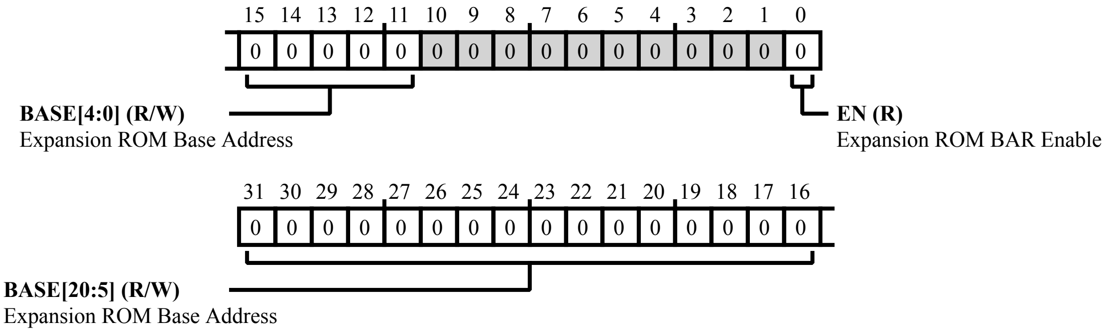

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE[4:0] (R/W)
Expansion ROM Base Address
EN (R)
Expansion ROM BAR Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE[20:5] (R/W)
Expansion ROM Base Address

Figure 29-106: PCIE_EP_ROMCFG_[n] Register Diagram

Table 29-115: PCIE_EP_ROMCFG_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:11 (R/W)	BASE	Expansion ROM Base Address. The `PCIE_EP_ROMCFG_[n].BASE` bit field correspond to the upper 21 bits of the Expansion ROM base address. The number of bits (out of these 21) that a device actually implements depends on how much address space the device requires. For instance, a device that requires a 64 KB area to map its expansion ROM would implement the top 16 bits in the register, leaving the bottom 5 (out of these 21) hardwired to 0. Devices that support an expansion ROM must implement this register.
0 (R/NW)	EN	Expansion ROM BAR Enable. The `PCIE_EP_ROMCFG_[n].EN` bit controls whether or not the device accepts accesses to its expansion ROM. When this bit is 0, the devices expansion ROM address space is disabled. When the bit is 1, address decoding is enabled using the parameters in the other part of the base register. This allows a device to be used with or without an expansion ROM depending on system configuration.

Subsystem ID and Subsystem Vendor ID Register

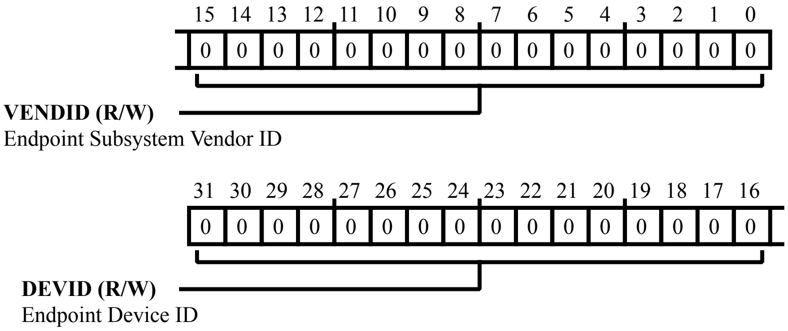

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VENDID (R/W)
Endpoint Subsystem Vendor ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DEVID (R/W)
Endpoint Device ID

Figure 29-107: PCIE_EP_SSVID_[n] Register Diagram

Table 29-116: PCIE_EP_SSVID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DEVID	Endpoint Device ID. Subsystem device ID
15:0 (R/W)	VENDID	Endpoint Subsystem Vendor ID. Subsystem vendor ID

Command and Status Register

The `PCIE_EP_STATCMD_[n]` register contains bits that provide status information. This register also contains bits that enable various configuration options.

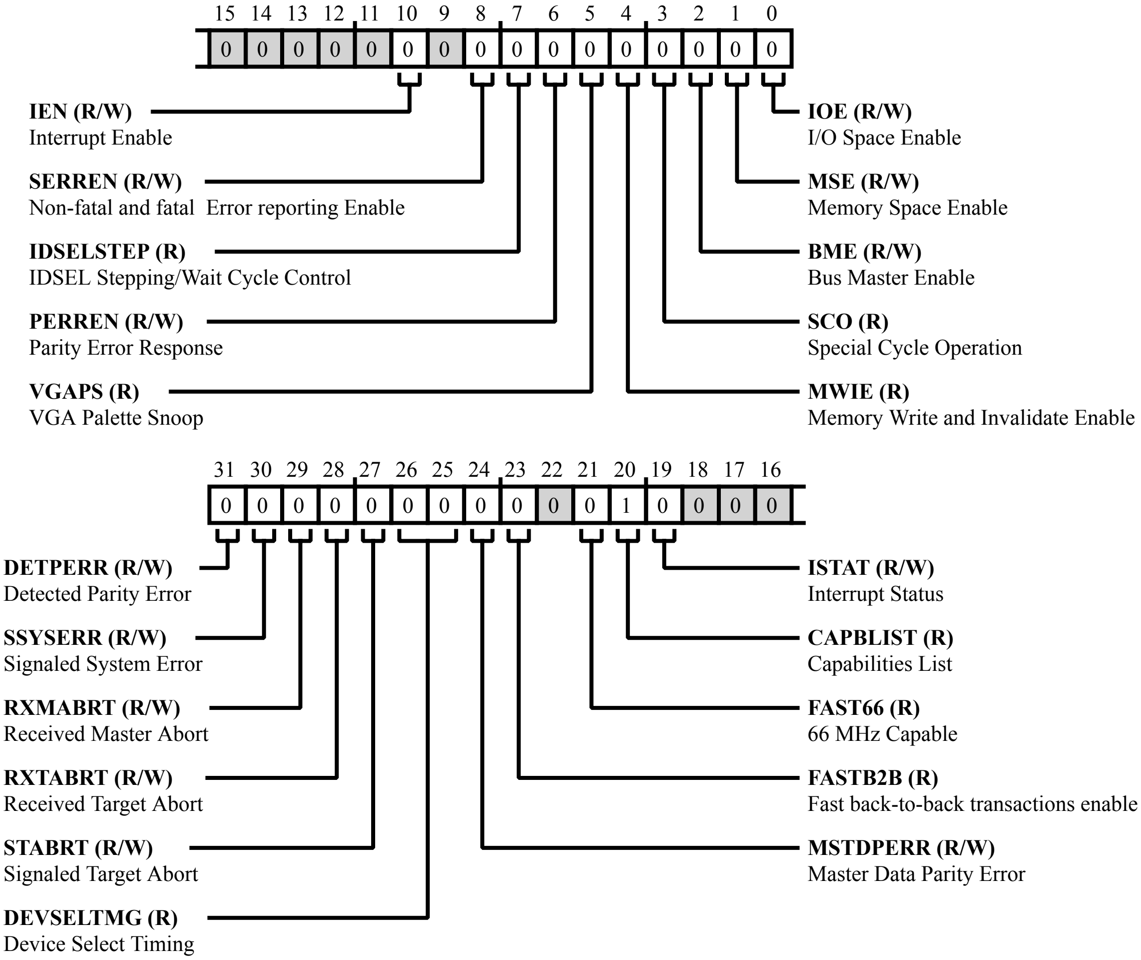

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IEN (R/W)
Interrupt Enable
SERREN (R/W)
Non-fatal and fatal Error reporting Enable
IDSELSTEP (R)
IDSEL Stepping/Wait Cycle Control
PERREN (R/W)
Parity Error Response
VGAPS (R)
VGA Palette Snoop
IOE (R/W)
I/O Space Enable
MSE (R/W)
Memory Space Enable
BME (R/W)
Bus Master Enable
SCO (R)
Special Cycle Operation
MWIE (R)
Memory Write and Invalidate Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
DETPERR (R/W)
Detected Parity Error
SSYSERR (R/W)
Signaled System Error
RXMABRT (R/W)
Received Master Abort
RXTABRT (R/W)
Received Target Abort
STABRT (R/W)
Signaled Target Abort
DEVSELTMG (R)
Device Select Timing
ISTAT (R/W)
Interrupt Status
CAPBLIST (R)
Capabilities List
FAST66 (R)
66 MHz Capable
FASTB2B (R)
Fast back-to-back transactions enable
MSTDPERR (R/W)
Master Data Parity Error

Figure 29-108: PCIE_EP_STATCMD_[n] Register Diagram

Table 29-117: PCIE_EP_STATCMD_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	DETPERR	Detected Parity Error. The `PCIE_EP_STATCMD_[n].DETPERR` bit is set by a function whenever it receives a poisoned TLP, regardless of the state of the `PCIE_EP_STATCMD_[n].PERREN` bit. On a function with a Type 1 Configuration header, the bit is set when the poisoned TLP is received by its primary side. The default value of this bit is 0.

Table 29-117: PCIE_EP_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W)	SSYSERR	Signaled System Error. The `PCIE_EP_STATCMD_[n].SSYSERR` bit is set when a function sends an ERR_FATAL or ERR_NONFATAL message, and the `PCIE_EP_STATCMD_[n].SERREN` bit is set to 1. The default value of this bit is 0.
29 (R/W)	RXMABRT	Received Master Abort. The `PCIE_EP_STATCMD_[n].RXMABRT` bit is set when a Requester receives a completion with unsupported request completion status. On a function with a Type 1 Configuration header, the `PCIE_EP_STATCMD_[n].RXMABRT` bit is set when the unsupported request is received by its Primary side. The default value of this bit is 0.
28 (R/W)	RXTABRT	Received Target Abort. The `PCIE_EP_STATCMD_[n].RXTABRT` bit is set when a requester receives a completion with completer abort completion status. On a function with a Type 1 Configuration header, the `PCIE_EP_STATCMD_[n].RXTABRT` bit is set when the completer abort is received by its primary side. The default value of this bit is 0.
27 (R/W)	STABRT	Signaled Target Abort. The `PCIE_EP_STATCMD_[n].STABRT` bit is set when a function completes a posted or non-posted request as a Completer Abort error. This applies to a function with a Type 1 Configuration header when the completer abort was generated by its primary side. The default value of this bit is 0.
26:25 (R/NW)	DEVSELTMG	Device Select Timing. Does not apply to PCI Express and must be set to 0.
24 (R/W)	MSTDPERR	Master Data Parity Error. The `PCIE_EP_STATCMD_[n].MSTDPERR` bit is set by a Root port, Switch Upstream port, or Switch Downstream port if the `PCIE_EP_STATCMD_[n].PERREN` bit in the command register is set to 1 and either of the following two conditions occurs. • A port receives a poisoned completion going downstream • A port transmits a poisoned request upstream If the `PCIE_EP_STATCMD_[n].PERREN` bit is cleared (=0), the `PCIE_EP_STATCMD_[n].MSTDPERR` bit is never set. The default value of this bit is 0.
23 (R/NW)	FASTB2B	Fast back-to-back transactions enable. Does not apply to PCI Express and must be set to 0.
21 (R/NW)	FAST66	66 MHz Capable. Does not apply to PCI Express and must be set to 0.

Table 29-117: PCIE_EP_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	CAPBLIST	Capabilities List. The `PCIE_EP_STATCMD_[n].CAPBLIST` bit indicates the presence of an Extended Capability list item. Since all PCI Express device functions are required to implement the PCI Express Capability structure, this bit is always set to 1.
19 (R/W)	ISTAT	Interrupt Status. When the `PCIE_EP_STATCMD_[n].ISTAT` bit is set, an INTx emulation interrupt is pending internally in the function. INTx emulation interrupts forwarded by Root and Switch ports from devices downstream of the Root or Switch port are not reflected in this bit. The default value of this bit is 0.
10 (R/W)	IEN	Interrupt Enable. The `PCIE_EP_STATCMD_[n].IEN` bit controls the ability of a PCI Express Function to generate INTx interrupts. When set, functions are prevented from asserting INTx interrupts. Any INTx emulation interrupts already asserted by the function must be de-asserted when this bit is set. When the `PCIE_EP_STATCMD_[n].IEN` bit is set, INTx interrupts use virtual wires that must, if asserted, be de-asserted using the appropriate Deassert_INTx message(s). Only the INTx virtual wire interrupt(s) associated with the function(s) for which this bit is set are affected. For Endpoints that generate INTx interrupts, this bit is required. For Endpoints that generate INTx interrupts, this bit is required. For Endpoints that do not generate INTx interrupts this bit is optional. If not implemented, this bit must be set to 0.
8 (R/W)	SERREN	Non-fatal and fatal Error reporting Enable. When the `PCIE_EP_STATCMD_[n].SERREN` bit is set, reports of non-fatal and fatal errors detected by the function to the root complex are issued. Errors are reported either through this bit or through the PCI Express specific bits in the `PCIE_EP_CCRID_[n]` register. In addition, for functions with Type 1 Configuration Space headers, this bit controls transmission by the primary interface of ERR_NONFATAL and ERR_FATAL error messages forwarded from the secondary interface. The `PCIE_EP_STATCMD_[n].SERREN` bit does not affect the transmission of forwarded ERR_COR messages. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. The default value of this bit is 0.
7 (R/NW)	IDSELSTEP	IDSEL Stepping/Wait Cycle Control. Does not apply to PCI Express and must be set to 0.

Table 29-117: PCIE_EP_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	PERREN	Parity Error Response. The `PCIE_EP_STATCMD_[n].PERREN` bit controls the logging of poisoned TLPs in the `PCIE_EP_STATCMD_[n].MSTDPERR` bit. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. The default value of this bit is 0.
5 (R/NW)	VGAPS	VGA Palette Snoop. Does not apply to PCI Express and must be set to 0.
4 (R/NW)	MWIE	Memory Write and Invalidate Enable. Does not apply to PCI Express and must be set to 0.
3 (R/NW)	SCO	Special Cycle Operation. Does not apply to PCI Express and must be set to 0.
2 (R/W)	BME	Bus Master Enable. The `PCIE_EP_STATCMD_[n].BME` bit controls the ability of a PCI Express Endpoint to issue memory and I/O read/write requests, and the ability of a Root or Switch Port to forward memory and I/O read/write requests in the upstream direction. For end points, when the `PCIE_EP_STATCMD_[n].BME` bit is set, the PCI Express function is allowed to issue memory or I/O requests. When this bit is cleared, the PCI Express function is not allowed to issue any memory or I/O requests. MSI/MSI-X interrupt messages are in-band memory writes and setting this bit to 0 also disables MSI/MSI-X interrupt messages. Requests other than memory or I/O requests are not controlled by this bit. The default value of this bit is 0.
1 (R/W)	MSE	Memory Space Enable. The `PCIE_EP_STATCMD_[n].MSE` bit enables memory space.
0 (R/W)	IOE	I/O Space Enable. The `PCIE_EP_STATCMD_[n].IOE` bit enables I/O space.

Error Source Identification Register

The PCIE_ERRSRC_ID_[n] register identifies the source (Requester ID) of first correctable and uncorrectable (Non- Fatal/Fatal) errors reported in the Root Error Status register. This register is updated regardless of the settings of the Root Control register and the Root Error Command register.

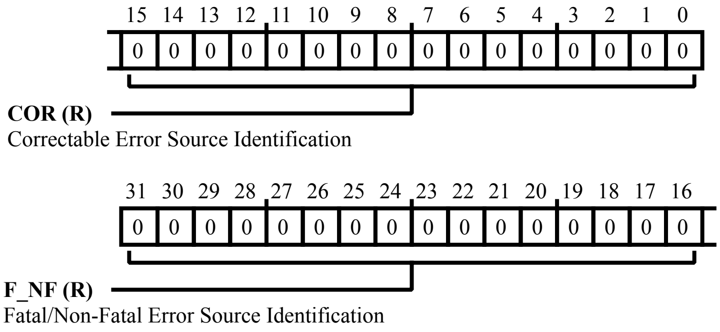

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COR (R)
Correctable Error Source Identification
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F_NF (R)
Fatal/Non-Fatal Error Source Identification

Figure 29-109: PCIE_ERRSRC_ID_[n] Register Diagram

Table 29-118: PCIE_ERRSRC_ID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	F_NF	Fatal/Non-Fatal Error Source Identification. The PCIE_ERRSRC_ID_[n].F_NF bit field is loaded with the Requester ID indicated in the received Fatal/Non-Fatal Error Message when the Fatal/Non-Fatal Error Message Received bit is not already set. Default value of this field is 0000.
15:0 (R/NW)	COR	Correctable Error Source Identification. The PCIE_ERRSRC_ID_[n].COR bit field is loaded with the Requester ID indicated in the received Correctable Error Message when the Correctable Error Received bit is not already set. Default value of this field is 0000.

Filter Mask 2 Register

The `PCIE_FILTMSK2_[n]` register is used to mask RADM Filtering and Error Handling rules.

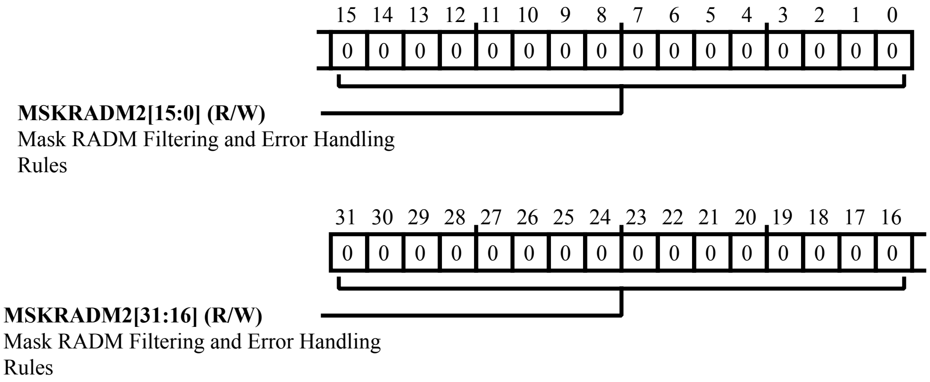

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSKRADM2[15:0] (R/W)
Mask RADM Filtering and Error Handling Rules
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSKRADM2[31:16] (R/W)
Mask RADM Filtering and Error Handling Rules

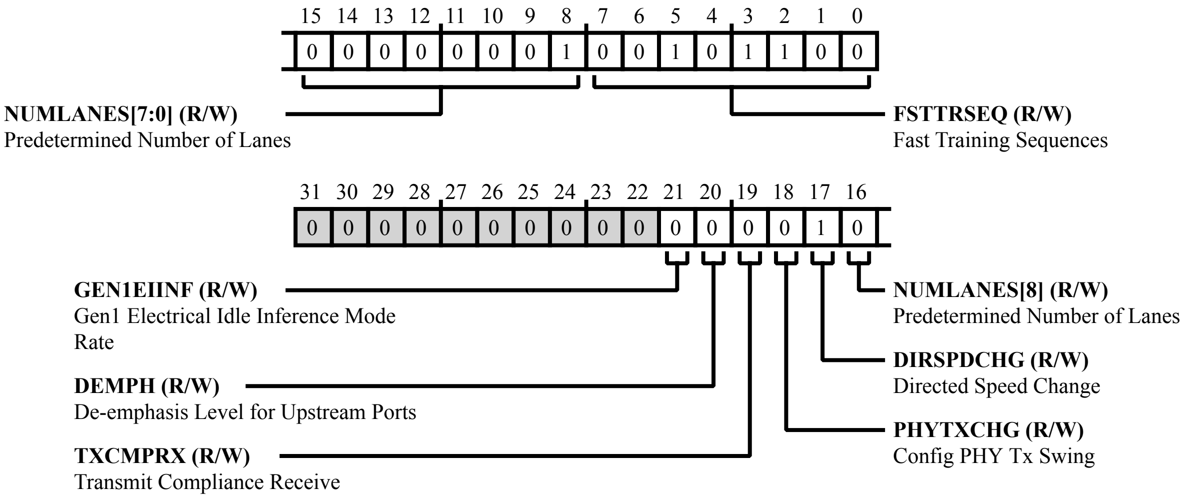
Figure 29-110: PCIE_FILTMSK2_[n] Register Diagram

Table 29-119: PCIE_FILTMSK2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MSKRADM2	Mask RADM Filtering and Error Handling Rules. Mask RADM Filtering and Error Handling Rules: Mask 2. There are several mask bits used to turn off the filtering and error handling rules. [31:4]: Reserved [3]: CX_FLT_MSK_HANDLE_FLUSH - 0: Disable core Filter to handle flush request - 1: Enable core Filter to handle flush request [2]: CX_FLT_MSK_DABRT_4UCPL - 0: Enable DLLP abort for unexpected completion - 1: Do not enable DLLP abort for unexpected completion [1]: CX_FLT_MSK_VENMSG1_DROP - 0: Vendor MSG Type 1 dropped silently - 1: Vendor MSG Type 1 not dropped [0]: CX_FLT_MSK_VENMSG0_DROP - 0: Vendor MSG Type 0 dropped with UR error reporting - 1: Vendor MSG Type 0 not dropped This register field is sticky.

Link Width and Speed Change Control Register

This register is used to configure some of the PCIe PHY features. The For details see the bit field descriptions.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0
NUMLANES[7:0] (R/W)
Predetermined Number of Lanes
FSTTRSEQ (R/W)
Fast Training Sequences
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
GEN1EIINF (R/W)
Gen1 Electrical Idle Inference Mode Rate
DEMPH (R/W)
De-emphasis Level for Upstream Ports
TXCMPRX (R/W)
Transmit Compliance Receive
NUMLANES[8] (R/W)
Predetermined Number of Lanes
DIRSPDCHG (R/W)
Directed Speed Change
PHYTXCHG (R/W)
Config PHY Tx Swing

Figure 29-111: PCIE_GEN2_CTL_[n] Register Diagram

Table 29-120: PCIE_GEN2_CTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GEN1EIINF	Gen1 Electrical Idle Inference Mode Rate. The `PCIE_GEN2_CTL_[n].GEN1EIINF` bit is used to determine inferred electrical idle (EI) in recovery. This bit does this by looking at the speed or Loopback.Active (as slave) state at Gen1 speed by looking for a 1 value on RxElecIdle instead of looking for a 0 on RxValid. If the PHY fails to deassert the RxValid signal in Recovery.Speed or Loopback.Active (because of corrupted EIOS for example), then electrical idle cannot be inferred successfully in the core by just detecting the condition RxValid=0. This bit is sticky.	
		0	Use RxElecIdle signal to infer Electrical Idle
		1	Use RxValid signal to infer Electrical Idle
20 (R/W)	DEMPH	De-emphasis Level for Upstream Ports. The `PCIE_GEN2_CTL_[n].DEMPH` bit is used to set the de-emphasis level for upstream ports. This bit is sticky.	
		0	-6 dB
		1	-3.5 dB
19 (R/W)	TXCMPRX	Transmit Compliance Receive. When the `PCIE_GEN2_CTL_[n].TXCMPRX` bit is set (=1), it signals the LTSSM to transmit TS ordered sets with the compliance receive bit assert (equal to 1). This bit sticky.	

Table 29-120: PCIE_GEN2_CTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	PHYTXCHG	Config PHY Tx Swing. The `PCIE_GEN2_CTL_[n].PHYTXCHG` bit controls the PHY transmitter voltage swing level. The core drives the mac_phy_txswing output from this bit. This bit is sticky.	
		0	Full swing
		1	Low swing
17 (R/W)	DIRSPDCHG	Directed Speed Change. Setting (=1) the `PCIE_GEN2_CTL_[n].DIRSPDCHG` bit instructs the LTSSM to initiate a speed change to Gen2 or Gen3 after the link is initialized at Gen1 speed. The speed change is initiated automatically after link up, and the core clears the contents of this field. To prevent this automatic speed change, write a lower speed value to the `PCIE_LNK_CTLSTAT2_[n].TRGSPD` bit field through the DBI before link up. When the speed change occurs, the core clears the contents of this field. When the program reads this bit it returns 0. To manually initiate the speed change: • Write to `PCIE_LNK_CTLSTAT2_[n].TRGSPD` in the local device • Deassert the `PCIE_GEN2_CTL_[n].DIRSPDCHG` bit • Assert the `PCIE_GEN2_CTL_[n].DIRSPDCHG` bit	
16:8 (R/W)	NUMLANES	Predetermined Number of Lanes. The `PCIE_GEN2_CTL_[n].NUMLANES` bit field defines the number of lanes which are connected and not bad. This value is used to limit the effective link width to ignore broken or unused lanes that detect a receiver. Indicates the number of lanes to check for exit from Electrical Idle in Polling. Active and L2.Idle. It is possible that the LTSSM might detect a receiver on a bad or broken lane during the detect substate. However, it is also possible that such a lane might also fail to exit Electrical Idle and therefore prevent a valid link from being configured. This value is referred to as the "Predetermined Number of Lanes" in section 4.2.6.2.1 of the PCI Express Base 3.0 Specification, revision 1.0. Encoding is as follows: 0x01: 1 lane 0x02: 2 lanes 0x03: 3 lanes ... When there are unused lanes in the system, then the value in this register must change to reflect the number of lanes. The value in the Link Mode Enable field of the Port Link Control Register must also be changed. The value in this register is normally the same as the encoded value in the Port Link Control Register. If one of the used lanes is bad then the value in this register must be reduced. This register field is sticky.	

Table 29-120: PCIE_GEN2_CTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FSTTRSEQ	Fast Training Sequences. The `PCIE_GEN2_CTL_[n].FSTTRSEQ` bit field configures the number of Fast Training Sequences (N_FTS) that the core advertises as its N_FTS during Gen2 or Gen3 link training. This value is used to inform the link partner about the PHY"s ability to recover synchronization after a low power state. This value is provided by the PHY vendor. Do not set N_FTS to zero; doing so can cause the LTSSM to go into the recovery state when exiting from L0s. This register field is sticky.

Header Log Register 0

The `PCIE_HDRLOG0_[n]` register contains bytes 0-3 of the Header of the TLP associated with an error.

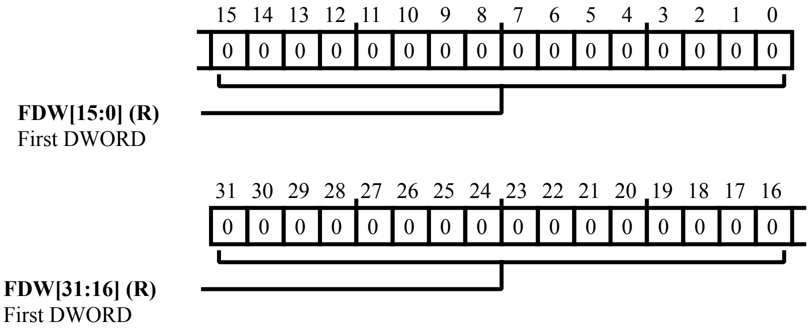

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FDW[15:0] (R)
First DWORD
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FDW[31:16] (R)
First DWORD

Figure 29-112: PCIE_HDRLOG0_[n] Register Diagram

Table 29-121: PCIE_HDRLOG0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	FDW	First DWORD. The `PCIE_HDRLOG0_[n].FDW` bit field is bytes 0-3 of the Header of the TLP associated with an error.

Header Log Register 1

The `PCIE_HDRLOG1_[n]` register contains bytes 4-7 of the Header of the TLP associated with an error.

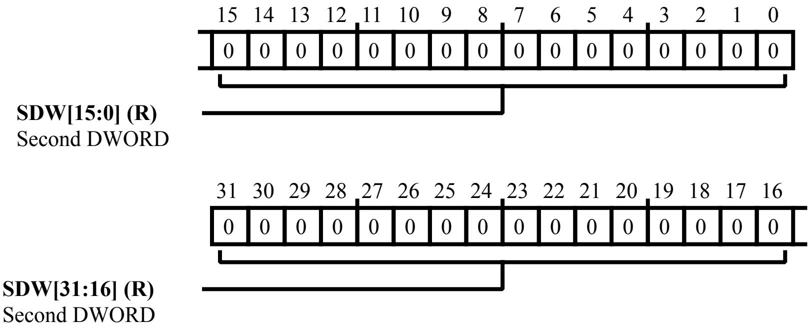

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SDW[15:0] (R)
Second DWORD
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SDW[31:16] (R)
Second DWORD

Figure 29-113: PCIE_HDRLOG1_[n] Register Diagram

Table 29-122: PCIE_HDRLOG1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	SDW	Second DWORD. The `PCIE_HDRLOG1_[n].SDW` bit field is bytes 4-7 of the Header of the TLP associated with an error.

Header Log Register 2

The `PCIE_HDRLOG2_[n]` register contains bytes 8-11 of the Header of the TLP associated with an error.

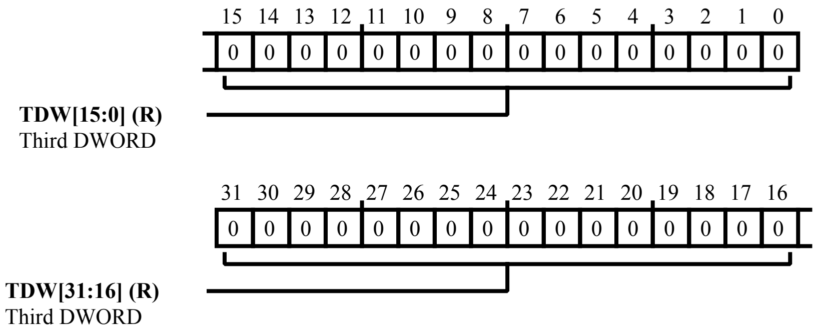

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TDW[15:0] (R)
Third DWORD
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TDW[31:16] (R)
Third DWORD

Figure 29-114: PCIE_HDRLOG2_[n] Register Diagram

Table 29-123: PCIE_HDRLOG2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TDW	Third DWORD. The `PCIE_HDRLOG2_[n].TDW` bit field is bytes 8-11 of the Header of the TLP associated with an error.

Header Log Register 3

The `PCIE_HDRLOG3_[n]` register contains bytes 12-15 of the Header of the TLP associated with an error.

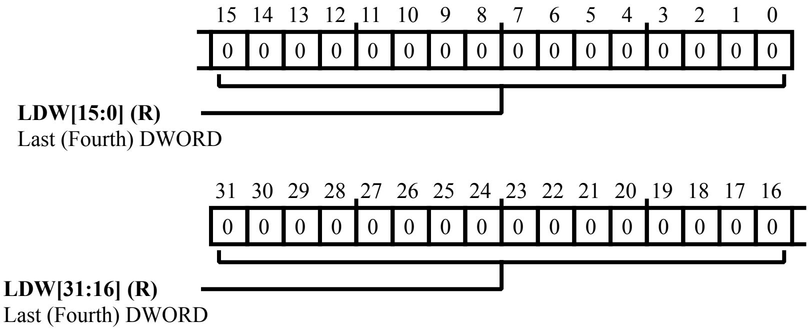

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LDW[15:0] (R)
Last (Fourth) DWORD
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LDW[31:16] (R)
Last (Fourth) DWORD

Figure 29-115: PCIE_HDRLOG3_[n] Register Diagram

Table 29-124: PCIE_HDRLOG3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	LDW	Last (Fourth) DWORD. The `PCIE_HDRLOG3_[n].LDW` bit field is bytes 12-15 of the Header of the TLP associated with an error.

iATU Region Control 1 Inbound Register

The `PCIE_IATU_CTL1_INB_[n]` register controls several aspects of inbound address translation used for mapping different address ranges to different memory spaces.

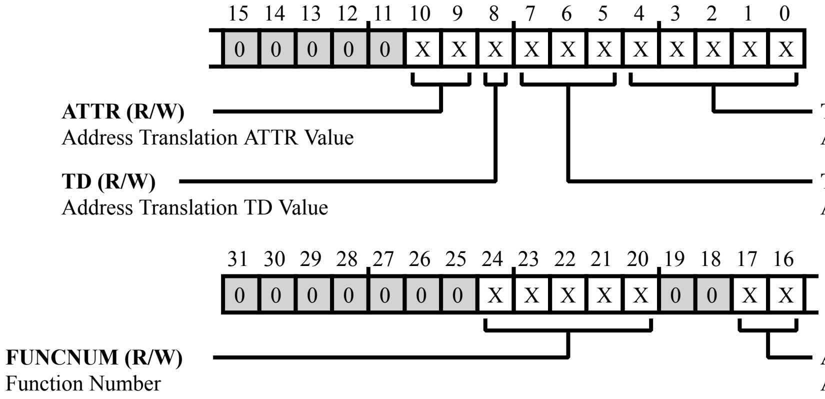

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 X X X X X X X X X X X
ATTR (R/W)
Address Translation ATTR Value
TD (R/W)
Address Translation TD Value
TYPE (R/W)
Address Translation TYPE Value
TC (R/W)
Address Translation TC Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 X X X X X 0 0 X X
FUNCNUM (R/W)
Function Number
AT (R/W)
Address Translation AT Value

Figure 29-116: PCIE_IATU_CTL1_INB_[n] Register Diagram

Table 29-125: PCIE_IATU_CTL1_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:20 (R/W)	FUNCNUM	Function Number. The `PCIE_IATU_CTL1_INB_[n].FUNCNUM` bit field is used for MEM-I/O and CFG0/CFG1 transactions as described below. • MEM-I/O - EP. When the Address and BAR matching logic in the core indicate that a MEM-I/O transaction matches a BAR in the function corresponding to this value, then address translation proceeds. This check is only performed if the `PCIE_IATU_CTL2_INB_[n].FNM` bit is set. • CFG0/CFG1 - EP. When the destination function number as specified in the routing ID of the TLP header matches the function, then address translation proceeds. This check is only performed if the `PCIE_IATU_CTL2_INB_[n].FNM` bit is set. • For RC this field is reserved and must be set to 0.
17:16 (R/W)	AT	Address Translation AT Value. When the value in the `PCIE_IATU_CTL1_INB_[n].AT` bit field matches the AT field of an inbound TLP, then address translation proceeds (when all other enabled field-matches are successful). This check is only performed if the `PCIE_IATU_CTL2_INB_[n].MAT` bit is set. Not valid, ATS is not supported.

Table 29-125: PCIE_IATU_CTL1_INB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10:9 (R/W)	ATTR	Address Translation ATTR Value. When the value in the `PCIE_IATU_CTL1_INB_[n].ATTR` bit field matches the ATTR field of an inbound TLP, then address translation proceeds (when all other enabled field-matches are successful). This check is only performed if the `PCIE_IATU_CTL2_INB_[n].ATTRM` bit is set.
8 (R/W)	TD	Address Translation TD Value. When the value in the `PCIE_IATU_CTL1_INB_[n].TD` bit field matches the TD field of an inbound TLP, then address translation proceeds (when all other enabled field-matches are successful). This check is only performed if the `PCIE_IATU_CTL2_INB_[n].TDM` bit is set.
7:5 (R/W)	TC	Address Translation TC Value. When the value in the `PCIE_IATU_CTL1_INB_[n].TC` bit field matches the TC field of an inbound TLP, then address translation proceeds (when all other enabled field-matches are successful). This check is only performed if the `PCIE_IATU_CTL2_INB_[n].TCM` bit is set.
4:0 (R/W)	TYPE	Address Translation TYPE Value. When the value in the `PCIE_IATU_CTL1_INB_[n].TYPE` bit field matches the TYPE field of an inbound TLP, then address translation proceeds (when all other enable field-matches are successful).

iATU Region Control 1 Outbound Register

The `PCIE_IATU_CTL1_OUTB_[n]` register controls several aspects of outbound address translation used for mapping different address ranges to different memory spaces.

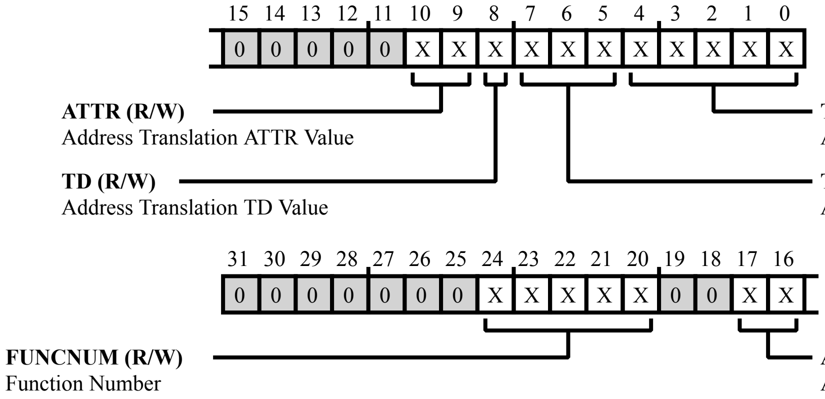

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 X X X X X X X X X X X
ATTR (R/W)
Address Translation ATTR Value
TD (R/W)
Address Translation TD Value
TYPE (R/W)
Address Translation TYPE Value
TC (R/W)
Address Translation TC Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 X X X X X 0 0 X X
FUNCNUM (R/W)
Function Number
AT (R/W)
Address Translation AT Value

Figure 29-117: PCIE_IATU_CTL1_OUTB_[n] Register Diagram

Table 29-126: PCIE_IATU_CTL1_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:20 (R/W)	FUNCNUM	Function Number. For an EP: When the value in the `PCIE_IATU_CTL1_OUTB_[n].FUNCNUM` bit field is matched to the address of an outbound TLP and the `PCIE_IATU_CTL2_OUTB_[n].FUNCBYP` bit field =0, then the function number used in generating the function part of the requester ID (RID) field of the TLP is taken from this 5-bit register. The value in this register must be 0x0. For a RC: The `PCIE_IATU_CTL1_OUTB_[n].FUNCNUM` bit field is swapped before SCB decomposition occurs so that the correct "Max_Read_Request_Size" and "Max_Payload_Size" values are used.
17:16 (R/W)	AT	Address Translation AT Value. When the value in the `PCIE_IATU_CTL1_OUTB_[n].AT` bit field matches the address of an outbound TLP, then the AT field of the TLP is changed to the value in this register. Not valid. ATS is not supported.
10:9 (R/W)	ATTR	Address Translation ATTR Value. When the value in the `PCIE_IATU_CTL1_OUTB_[n].ATTR` bit field matches the address of an outbound TLP, then the ATTR field of the TLP is changed to the value in this register.
8 (R/W)	TD	Address Translation TD Value. When the value in the `PCIE_IATU_CTL1_OUTB_[n].TD` bit field matches the address of an outbound TLP, then the TD field of the TLP is changed to the value in this register.

Table 29-126: PCIE_IATU_CTL1_OUTB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:5 (R/W)	TC	Address Translation TC Value. When the value in the `PCIE_IATU_CTL1_OUTB_[n].TC` bit field matches the address of an outbound TLP, then the TC field of the TLP is changed to the value in this register.
4:0 (R/W)	TYPE	Address Translation TYPE Value. When the value in the `PCIE_IATU_CTL1_OUTB_[n].TYPE` bit field matches the address of an outbound TLP, then the TYPE field of the TLP is changed to the value in this register.

iATU Region Control 2 Inbound Register

The `PCIE_IATU_CTL2_INB_[n]` register controls several aspects of inbound address translation used for mapping different address ranges to different memory spaces.

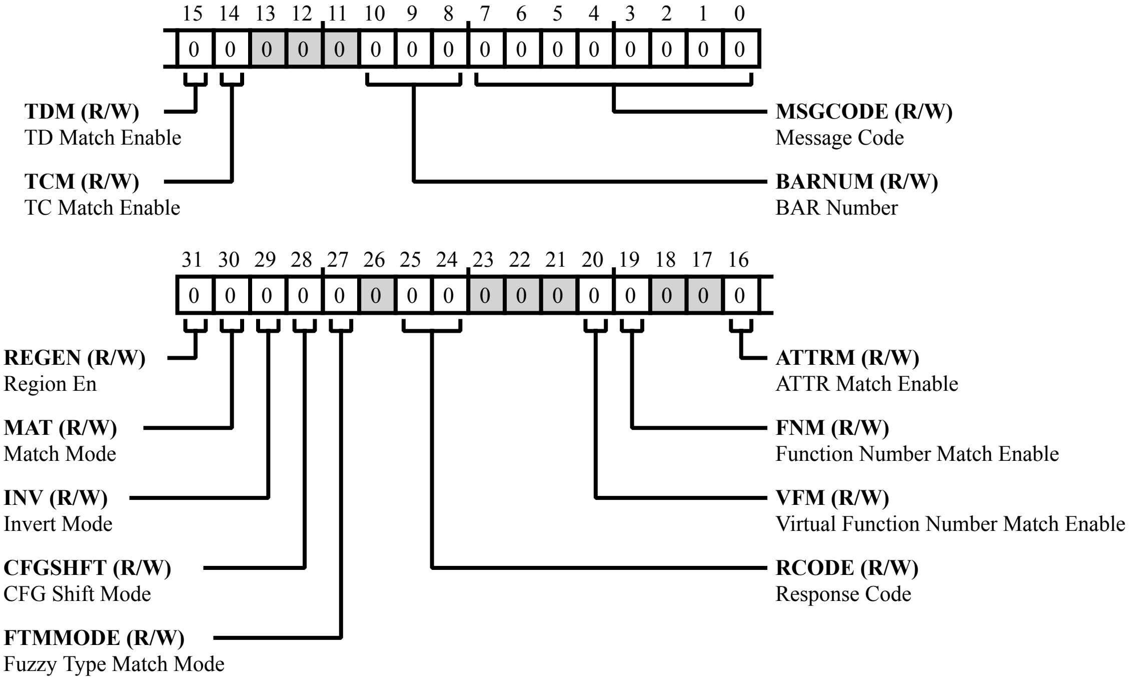

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TDM (R/W)
TD Match Enable
TCM (R/W)
TC Match Enable
MSGCODE (R/W)
Message Code
BARNUM (R/W)
BAR Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
REGEN (R/W)
Region En
MAT (R/W)
Match Mode
INV (R/W)
Invert Mode
CFGSHFT (R/W)
CFG Shift Mode
FTMMODE (R/W)
Fuzzy Type Match Mode
ATTRM (R/W)
ATTR Match Enable
FNM (R/W)
Function Number Match Enable
VFM (R/W)
Virtual Function Number Match Enable
RCODE (R/W)
Response Code

Figure 29-118: PCIE_IATU_CTL2_INB_[n] Register Diagram

Table 29-127: PCIE_IATU_CTL2_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	REGEN	Region En. The `PCIE_IATU_CTL2_INB_[n].REGEN` bit must be set to 1 for address translation to take place.

Table 29-127: PCIE_IATU_CTL2_INB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W)	MAT	Match Mode. The `PCIE_IATU_CTL2_INB_[n].MAT` bit determines Inbound matching mode for TLPs. The mode depends on the type of TLP that is received. For MEM-I/O TLPs, this field is interpreted as: • 0=Address Match Mode. The iATU operates using addresses as in the outbound direction. The Region Base and Limit Registers must be setup. • 1=BAR Match Mode. BAR matching is used. The "BAR Number" field is relevant. Not used for RC. For CFG0 TLPs, this field is interpreted as: • 0=Routing ID Match Mode. The iATU interprets the Routing ID (Bytes 8 to 11 of TLP header) as an address. This corresponds to the upper 16 bits of the address in MEM-I/O transactions. The Routing ID of the TLP must be within the base and limit of the iATU region for matching to proceed. • 1=Accept Mode. The iATU accepts all CFG0 transactions as address matches. The routing ID in the CFG0 TLP is ignored. This is useful as all received CFG0 TLPs should be processed regardless of the Bus number. For MSG/MSGD TLPs, this field is interpreted as: • 0=Address Match Mode. The iATU treats the third dword and fourth dword of the inbound MSG/MSGD TLP as an address and it is matched against the Region Base and Limit Registers. • 1=Vendor ID Match Mode. This mode is relevant for ID-routed Vendor Defined Messages. The iATU ignores the Routing ID (Bus, Device, Function) in bits [31:16] of the third dword of the TLP header, but matches against the Vendor ID in bits [15:0] of the third dword of the TLP header. Bits [15:0] of the Region Upper Base register should be programmed with the required Vendor ID. The lower Base and Limit Register should be programmed to translate TLPs based on vendor specific information in the fourth dword of the TLP header.
29 (R/W)	INV	Invert Mode. When the `PCIE_IATU_CTL2_INB_[n].INV` bit is set the address matching region is inverted. Therefore, an address match occurs when the untranslated address is in the region outside the defined range (Base Address to Limit Address).
28 (R/W)	CFGSHFT	CFG Shift Mode. The `PCIE_IATU_CTL2_INB_[n].CFGSHFT` bit is useful for CFG transactions where the PCIe configuration mechanism maps bits [27:12] of the address to the bus/device and function number. This allows a CFG configuration space to be located in any 256MB window of the application memory space using a 28-bit effective address.Shifts bits [27:12] of the untranslated address to form bits [31:16] of the translated address.

Table 29-127: PCIE_IATU_CTL2_INB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	FTMMODE	Fuzzy Type Match Mode. When the `PCIE_IATU_CTL2_INB_[n].FTMMODE` bit is set, the iATU relaxes the matching of the TLP TYPE field against the expected TYPE field so that CfgRd0/CfgRd1 and CfgWr0/CfgWr1 and MRd/MRdLk TLPs are seen as identical. The Routing field of Msg/MsgD TLPs is ignored. For example, CFG0 in the `PCIE_IATU_CTL1_INB_[n].TYPE` field matches against an inbound CfgRd0, CfgRd1, CfgWr0 or CfgWr1 TLP.	
25:24 (R/W)	RCODE	Response Code. The `PCIE_IATU_CTL2_INB_[n].RCODE` bit field defines the type of response to give for accesses matching this region. This overrides the normal RADM filter response.	
		0	Normal RADM filter response is used
		1	Unsupported request (UR)
		2	Completer abort (CA)
		3	Reserved
20 (R/W)	VFM	Virtual Function Number Match Enable. For an EP the `PCIE_IATU_CTL2_INB_[n].VFM` bit ensures that a successful Virtual Function Number TLP field comparison match (see `PCIE_IATU_CTL1_INB_[n].FUNCNUM`) occurs (in MEM-I/O and CFG0/CFG1 transactions) for address translation to proceed. For RC the `PCIE_IATU_CTL2_INB_[n].VFM` bit is reserved and must be set to 0.	
19 (R/W)	FNM	Function Number Match Enable. For an EP the `PCIE_IATU_CTL2_INB_[n].FNM` bit ensures that a successful Function Number TLP field comparison match (see `PCIE_IATU_CTL1_INB_[n].FUNCNUM`) occurs (in MEM-I/O and CFG0/CFG1 transactions) for address translation to proceed. For RC the `PCIE_IATU_CTL2_INB_[n].FNM` bit is reserved and must be set to 0.	
16 (R/W)	ATTRM	ATTR Match Enable. The `PCIE_IATU_CTL2_INB_[n].ATTRM` bit ensures that a successful ATTR TLP field comparison match (see `PCIE_IATU_CTL1_INB_[n].ATTR`) occurs for address translation to proceed.	
15 (R/W)	TDM	TD Match Enable. The `PCIE_IATU_CTL2_INB_[n].TDM` bit ensures that a successful TD TLP field comparison match (see `PCIE_IATU_CTL1_INB_[n].TD`) occurs for address translation to proceed.	

Table 29-127: PCIE_IATU_CTL2_INB_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>14
(R/W)</td><td>TCM</td><td colspan="2">TC Match Enable.
The <code>PCIE_IATU_CTL2_INB_[n].TCM</code> bit ensures that a successful TC TLP field comparison match (see <code>PCIE_IATU_CTL1_INB_[n].TC</code>) occurs for address translation to proceed.</td></tr>
<tr><td rowspan="9">10:8
(R/W)</td><td rowspan="9">BARNUM</td><td colspan="2">BAR Number.
For an EP, when the value in the <code>PCIE_IATU_CTL2_INB_[n].BARNUM</code> bit field matches the BAR number of an inbound MEM "that is matched by the normal internal BAR address matching mechanism ", address translation proceeds (when all other enabled field-matches are successful). This check is only performed if the <code>PCIE_IATU_CTL2_INB_[n].MAT</code> bit is set.
For RC this field is reserved and must be set to 0.</td></tr>
<tr><td>0</td><td>BAR0</td></tr>
<tr><td>1</td><td>BAR1</td></tr>
<tr><td>2</td><td>BAR2</td></tr>
<tr><td>3</td><td>BAR3</td></tr>
<tr><td>4</td><td>BAR4</td></tr>
<tr><td>5</td><td>BAR5</td></tr>
<tr><td>6</td><td>ROM</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>7:0
(R/W)</td><td>MSGCODE</td><td colspan="2">Message Code.
When the value in the <code>PCIE_IATU_CTL2_INB_[n].MSGCODE</code> bit field matches the TYPE field of an inbound Msg/MsgD TLP, then address translation proceeds (when all other enabled field-matches are successful).</td></tr>
</table>

iATU Region Control 2 Register Outbound

The `PCIE_IATU_CTL2_OUTB_[n]` register controls several aspects of outbound address translation used for mapping different address ranges to different memory spaces.

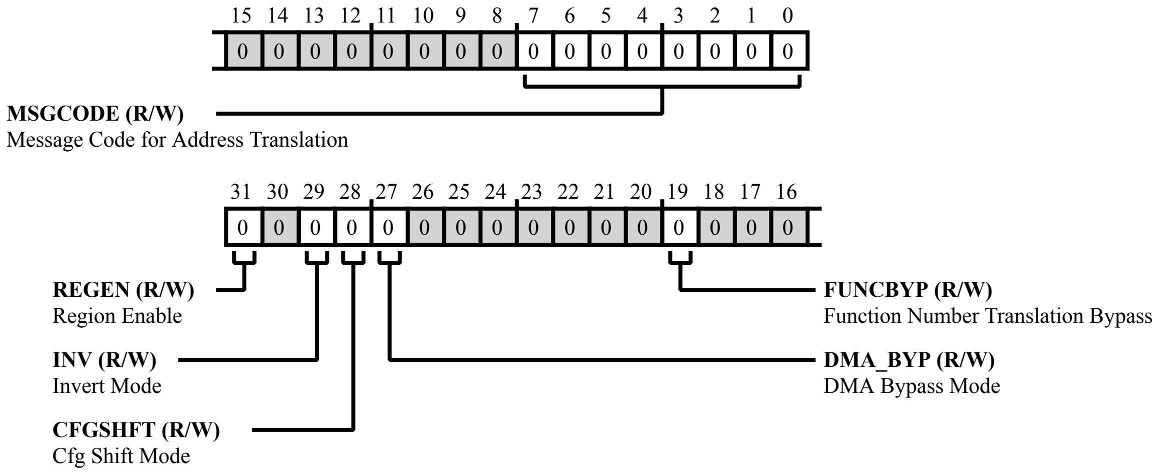

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSGCODE (R/W)
Message Code for Address Translation
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
REGEN (R/W)
Region Enable
INV (R/W)
Invert Mode
CFGSHFT (R/W)
Cfg Shift Mode
FUNCBYP (R/W)
Function Number Translation Bypass
DMA_BYP (R/W)
DMA Bypass Mode

Figure 29-119: PCIE_IATU_CTL2_OUTB_[n] Register Diagram

Table 29-128: PCIE_IATU_CTL2_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	REGEN	Region Enable. The `PCIE_IATU_CTL2_OUTB_[n].REGEN` bit must be set to 1 for address translation to take place.
29 (R/W)	INV	Invert Mode. When the `PCIE_IATU_CTL2_OUTB_[n].INV` bit is set the address matching region is inverted. Therefore, an address match occurs when the untranslated address is in the region outside the defined range (Base Address to Limit Address).
28 (R/W)	CFGSHFT	Cfg Shift Mode. The `PCIE_IATU_CTL2_OUTB_[n].CFGSHFT` bit is used for CFG transactions where the PCIe configuration mechanism maps bits [27:12] of the address to the bus/device and function number. This allows a CFG configuration space to be located in any 256MB window of the application memory space using a 28-bit effective address. Shifts bits [27:12] of the untranslated address to form bits [31:16] of the translated address.
27 (R/W)	DMA_BYP	DMA Bypass Mode. The `PCIE_IATU_CTL2_OUTB_[n].DMA_BYP` bit allows request TLPs which are initiated by the DMA controller to pass through the iATU untranslated.

Table 29-128: PCIE_IATU_CTL2_OUTB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	FUNCBYP	Function Number Translation Bypass. The `PCIE_IATU_CTL2_OUTB_[n].FUNCBYP` bit field configures the iATU to use the function number of the translated TLP from the Slave SCB interface and not from the `PCIE_IATU_CTL1_OUTB_[n].FUNCNUM` or the `PCIE_IATU_CTL3_OUTB_[n].VFNUM` field.
7:0 (R/W)	MSGCODE	Message Code for Address Translation. When the address of an inbound TLP is matched to the `PCIE_IATU_CTL2_OUTB_[n].MSGCODE` bit field, and the translated TLP TYPE field is Msg or MsgD; then the message field of the TLP is changed to the value in this register.

iATU Region Control 3 Register

The `PCIE_IATU_CTL3_INB_[n]` register controls virtual function of inbound address translation used for mapping different address ranges to different memory spaces.

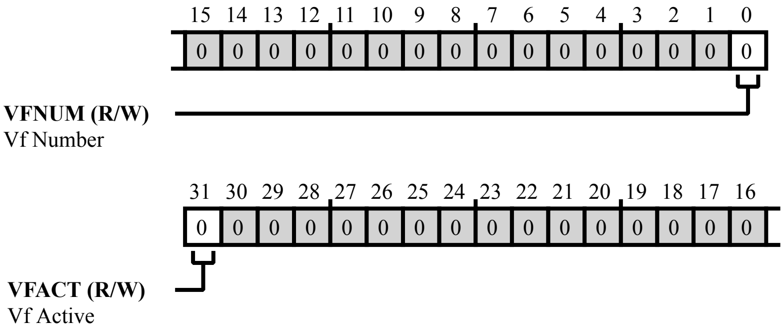

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VFNUM (R/W)
Vf Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VFACT (R/W)
Vf Active

Figure 29-120: PCIE_IATU_CTL3_INB_[n] Register Diagram

Table 29-129: PCIE_IATU_CTL3_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	VFACT	Vf Active. The `PCIE_IATU_CTL3_INB_[n].VFACT` bit must be set to 0.
0 (R/W)	VFNUM	Vf Number. The `PCIE_IATU_CTL3_INB_[n].VFNUM` bit must be set to 0.

iATU Region Control 3 Register

The `PCIE_IATU_CTL3_OUTB_[n]` register controls virtual function of outbound address translation used for mapping different address ranges to different memory spaces.

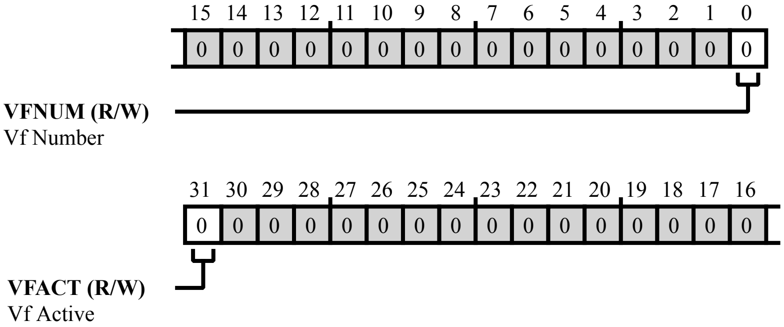

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VFNUM (R/W)
Vf Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VFACT (R/W)
Vf Active

Figure 29-121: PCIE_IATU_CTL3_OUTB_[n] Register Diagram

Table 29-130: PCIE_IATU_CTL3_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	VFACT	Vf Active. The `PCIE_IATU_CTL3_OUTB_[n].VFACT` bit must be set to 0.
0 (R/W)	VFNUM	Vf Number. The `PCIE_IATU_CTL3_OUTB_[n].VFNUM` bit must be set to 0.

IATU Lower Base Inbound Address Register

The `PCIE_IATU_LBADDR_INB_[n]` register contains the minimum size of an address translation region to 64 KB. The lower 16 bits of the Base, Limit and Target registers are zero and all address regions are aligned on 64 KB boundaries. More precisely, the lower 16 bits are zero.

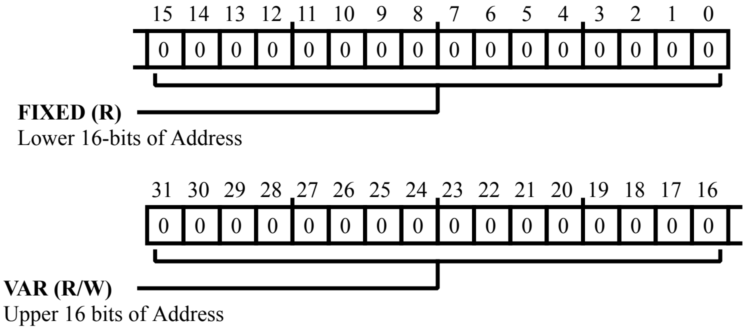

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FIXED (R)
Lower 16-bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR (R/W)
Upper 16 bits of Address

Figure 29-122: PCIE_IATU_LBADDR_INB_[n] Register Diagram

Table 29-131: PCIE_IATU_LBADDR_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of Address. The `PCIE_IATU_LBADDR_INB_[n].VAR` bit field forms bits [31:16] of the start address of the address region to be translated. This bit field is writeable.
15:0 (R/NW)	FIXED	Lower 16-bits of Address. The `PCIE_IATU_LBADDR_INB_[n].FIXED` bit field forms bits [15:0] of the start address of the address region to be translated. The start address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core. This field is read-only.

IATU Lower Base Outbound Address Register

The `PCIE_IATU_LBADDR_OUTB_[n]` register contains the minimum size of an address translation region to 64 KB. The lower 16 bits of the Base, Limit and Target registers are zero and all address regions are aligned on 64 KB boundaries. More precisely, the lower 16 bits are zero.

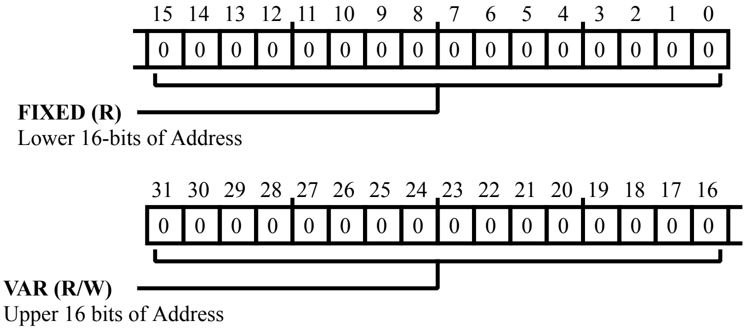

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FIXED (R)
Lower 16-bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR (R/W)
Upper 16 bits of Address

Figure 29-123: PCIE_IATU_LBADDR_OUTB_[n] Register Diagram

Table 29-132: PCIE_IATU_LBADDR_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of Address. The `PCIE_IATU_LBADDR_OUTB_[n].VAR` bit field forms bits [31:16] of the start address of the address region to be translated. This bit field is writeable.
15:0 (R/NW)	FIXED	Lower 16-bits of Address. The `PCIE_IATU_LBADDR_OUTB_[n].FIXED` bit field forms bits [15:0] of the start address of the address region to be translated. The start address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core.

IATU Inbound Limit Address Register

The `PCIE_IATU_LMTADDR_INB_[n]` register includes the 32 bits end address of the address region to be translated.

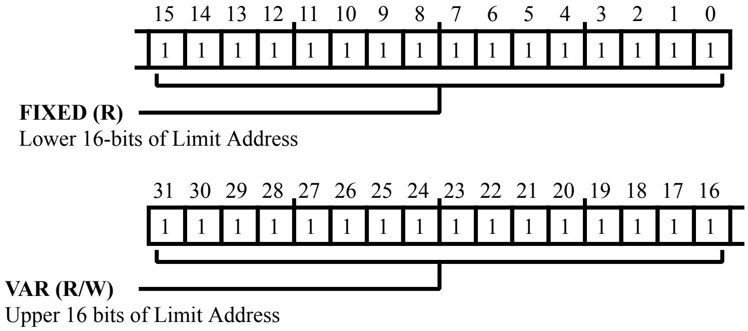

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
FIXED (R)
Lower 16-bits of Limit Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
VAR (R/W)
Upper 16 bits of Limit Address

Figure 29-124: PCIE_IATU_LMTADDR_INB_[n] Register Diagram

Table 29-133: PCIE_IATU_LMTADDR_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of Limit Address. The `PCIE_IATU_LMTADDR_INB_[n].VAR` bit field forms bits [31:16] of the end address of the address region to be translated. This bit field is writeable.
15:0 (R/NW)	FIXED	Lower 16-bits of Limit Address. The `PCIE_IATU_LMTADDR_INB_[n].FIXED` bit field forms bits [15:0] of the end address of the address region to be translated. The end address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core.

IATU Outbound Limit Address Register

The `PCIE_IATU_LMTADDR_OUTB_[n]` register includes the 32 bits end address of the address region to be translated.

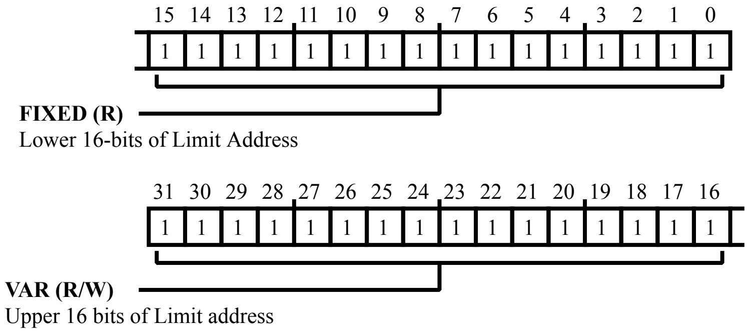

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
FIXED (R)
Lower 16-bits of Limit Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
VAR (R/W)
Upper 16 bits of Limit address

Figure 29-125: PCIE_IATU_LMTADDR_OUTB_[n] Register Diagram

Table 29-134: PCIE_IATU_LMTADDR_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of Limit address. The `PCIE_IATU_LMTADDR_OUTB_[n].VAR` bit field forms bits [31:16] of the end address of the address region to be translated. This bit field is writeable.
15:0 (R/NW)	FIXED	Lower 16-bits of Limit Address. The `PCIE_IATU_LMTADDR_OUTB_[n].FIXED` bit field forms bits [15:0] of the end address of the address region to be translated. The end address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core.

IATU Lower Target Address Inbound Register

The `PCIE_IATU_LTADDR_INB_[n]` register includes the least significant 32 bits of the 64-bit new address of the address region to be translated.

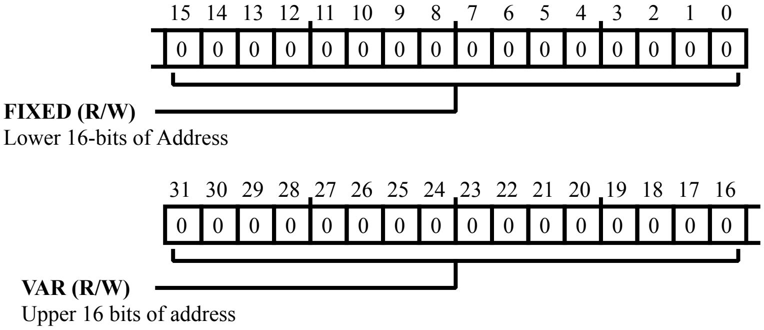

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FIXED (R/W)
Lower 16-bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR (R/W)
Upper 16 bits of address

Figure 29-126: PCIE_IATU_LTADDR_INB_[n] Register Diagram

Table 29-135: PCIE_IATU_LTADDR_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of address. The `PCIE_IATU_LTADDR_INB_[n].VAR` bit field forms bits [31:16] of the new address of the translated region. This bit field is writeable.
15:0 (R/W)	FIXED	Lower 16-bits of Address. The `PCIE_IATU_LTADDR_INB_[n].FIXED` bit field forms bits [15:0] of the start address of the new address of the translated region. The start address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core. This bit field is read-only.

IATU Lower Target Address Outbound Register

The `PCIE_IATU_LTADDR_OUTB_[n]` register includes the least significant 32 bits of the 64-bit new address of the address region to be translated.

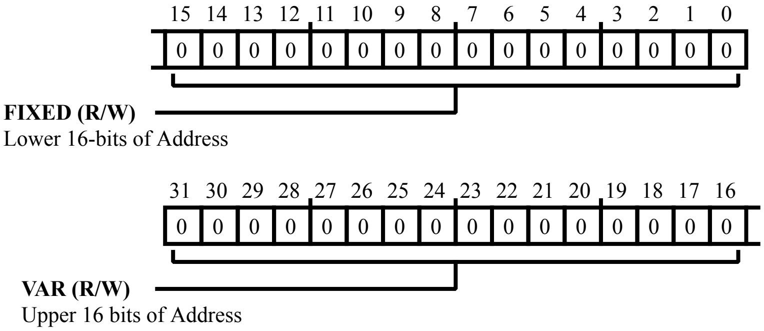

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FIXED (R/W)
Lower 16-bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR (R/W)
Upper 16 bits of Address

Figure 29-127: PCIE_IATU_LTADDR_OUTB_[n] Register Diagram

Table 29-136: PCIE_IATU_LTADDR_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	VAR	Upper 16 bits of Address. The `PCIE_IATU_LTADDR_OUTB_[n].VAR` bit field forms bits [31:16] of the new address of the translated region. This bit field is writeable.
15:0 (R/W)	FIXED	Lower 16-bits of Address. The `PCIE_IATU_LTADDR_OUTB_[n].FIXED` bit field forms bits [15:0] of the start address of the new address of the translated region. The start address must be aligned to 64 KB, so these bits are always 0. A write to this location is ignored by the PCIe core. This bit field is read-only.

IATU Upper Base Address Inbound Register

The `PCIE_IATU_UBADDR_INB_[n]` register forms bits [63:32] of the start (and end) address of the address region to be translated. In systems with a 32-bit address space, this register is not used and therefore writing to this register has no effect.

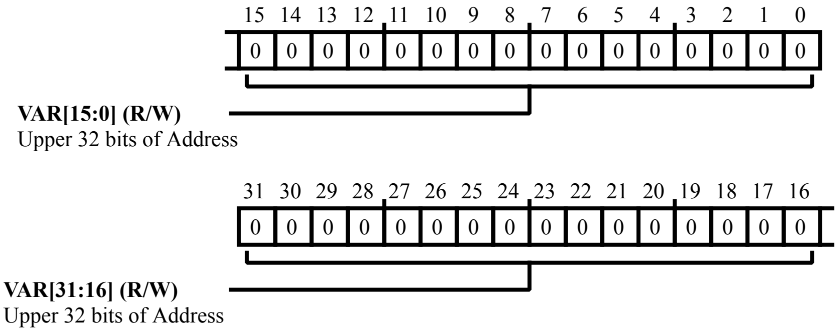

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[15:0] (R/W)
Upper 32 bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[31:16] (R/W)
Upper 32 bits of Address

Figure 29-128: PCIE_IATU_UBADDR_INB_[n] Register Diagram

Table 29-137: PCIE_IATU_UBADDR_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VAR	Upper 32 bits of Address. The `PCIE_IATU_UBADDR_INB_[n].VAR` bit field forms bits [63:32] of the start (and end) address of the address region to be translated. In systems with a 32-bit address space, this register is not used and therefore writing to this register has no effect.

IATU Upper Base Address Outbound Register

The `PCIE_IATU_UBADDR_OUTB_[n]` register forms bits [63:32] of the start (and end) address of the address region to be translated. In systems with a 32-bit address space, this register is not used and therefore writing to this register has no effect.

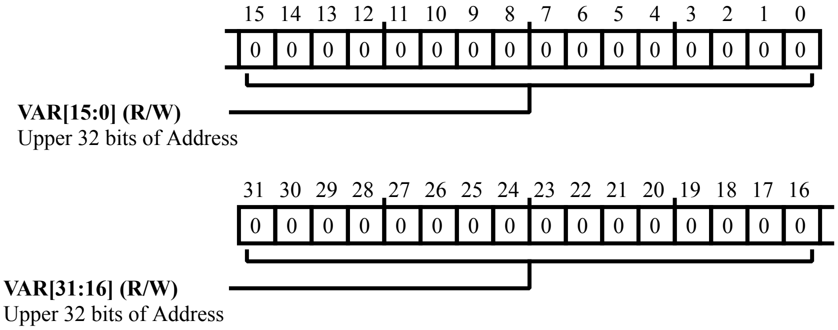

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[15:0] (R/W)
Upper 32 bits of Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[31:16] (R/W)
Upper 32 bits of Address

Figure 29-129: PCIE_IATU_UBADDR_OUTB_[n] Register Diagram

Table 29-138: PCIE_IATU_UBADDR_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VAR	Upper 32 bits of Address. The `PCIE_IATU_UBADDR_OUTB_[n].VAR` bit field forms bits [63:32] of the start (and end) address of the address region to be translated. In systems with a 32-bit address space, this register is not used and therefore writing to this register has no effect.

iATU Upper Target Address Inbound Register

The `PCIE_IATU_UTADDR_INB_[n]` register forms bits [63:32] of the start address of the new address of the translated region.

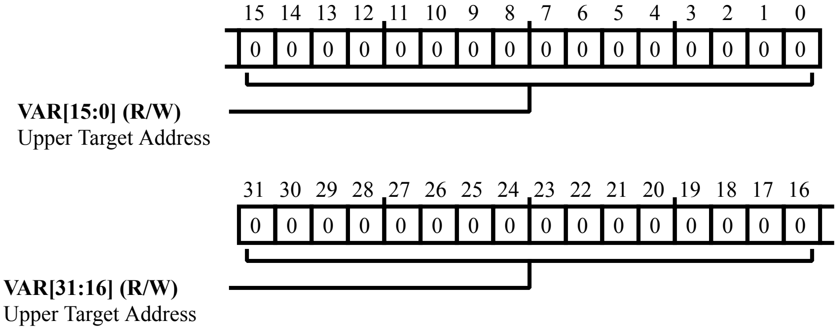

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[15:0] (R/W)
Upper Target Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[31:16] (R/W)
Upper Target Address

Figure 29-130: PCIE_IATU_UTADDR_INB_[n] Register Diagram

Table 29-139: PCIE_IATU_UTADDR_INB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VAR	Upper Target Address. The `PCIE_IATU_UTADDR_INB_[n].VAR` bit field forms bits [63:32] of the start address of the new address of the translated region. This bit field is writeable.

iATU Upper Target Address Outbound Register

The `PCIE_IATU_UTADDR_OUTB_[n]` register contains bits [63:32] of the start address of the new address of the translated region.

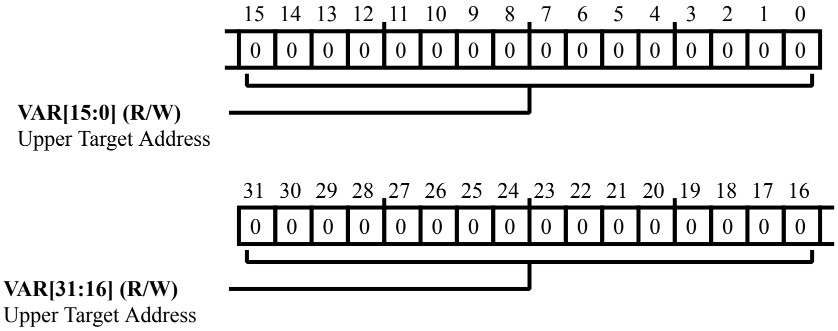

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[15:0] (R/W)
Upper Target Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VAR[31:16] (R/W)
Upper Target Address

Figure 29-131: PCIE_IATU_UTADDR_OUTB_[n] Register Diagram

Table 29-140: PCIE_IATU_UTADDR_OUTB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VAR	Upper Target Address. The `PCIE_IATU_UTADDR_OUTB_[n].VAR` bit field forms bits [63:32] of the start address of the new address of the translated region.

IATU View Port Register

The `PCIE_IATU_VWPRT_[n]` register programs the iATU registers through an indirect addressing scheme to reduce the address footprint in the PCI Express extended configuration space. The size of the required port logic space does not depend on the number of regions defined as the index register is used to select which memory region is being accessed.

There are 28 bytes of register space implemented per address region per direction. The number of address regions that are remapped by the iATU is configurable in each direction from 0 (none) to 256. However, only 32 bytes of the PCIe extended configuration space address map is used.

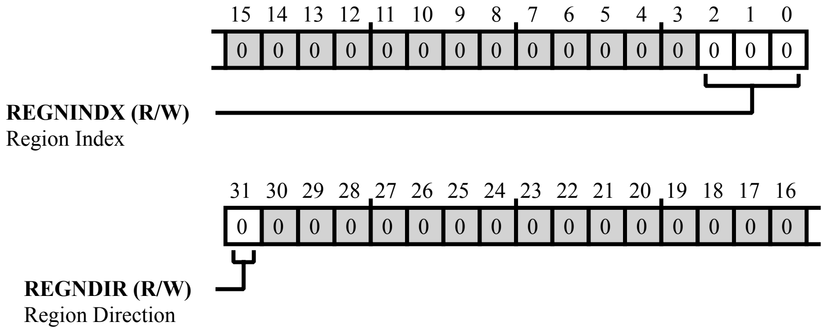


Figure 29-132: PCIE_IATU_VWPRT_[n] Register Diagram

Table 29-141: PCIE_IATU_VWPRT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	REGNDIR	Region Direction. The `PCIE_IATU_VWPRT_[n].REGNDIR` bit defines the region being accessed as either inbound or outbound. 0: Outbound 1: Inbound
2:0 (R/W)	REGNINDX	Region Index. The `PCIE_IATU_VWPRT_[n].REGNINDX` bit field defines which region is being accessed when writing to the control, base, limit and target registers. The value must not be set to a number greater than 3 when an outbound region is being accessed or a value greater than 5 when an inbound region is being accessed. X = is log2(CX_ATU_NUM_INBOUND_REGIONS and CX_ATU_NUM_OUTBOUND_REGIONS).

Lane Skew Register

The `PCIE_LANE_SKEW_[n]` register contains bits that control lane skew and Data Link Layer Packet operations.

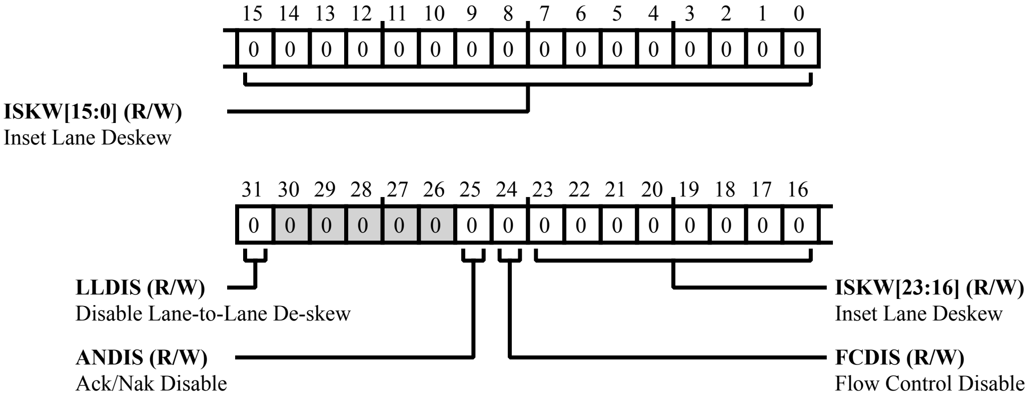

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ISKW[15:0] (R/W)
Inset Lane Deskew
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LLDIS (R/W)
Disable Lane-to-Lane De-skew
ISKW[23:16] (R/W)
Inset Lane Deskew
ANDIS (R/W)
Ack/Nak Disable
FCDIS (R/W)
Flow Control Disable

Figure 29-133: PCIE_LANE_SKEW_[n] Register Diagram

Table 29-142: PCIE_LANE_SKEW_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LLDIS	Disable Lane-to-Lane De-skew. The `PCIE_LANE_SKEW_[n].LLDIS` bit causes the core to disable the internal Lane-to-Lane de-skew logic. This register field is sticky.	
		0	Lane-to-Lane De-skew enabled
		1	Lane-to-Lane De-skew disabled
25 (R/W)	ANDIS	Ack/Nak Disable. The `PCIE_LANE_SKEW_[n].ANDIS` bit prevents the core from sending Ack and Nak DLLPs. This register field is sticky.	
		0	Do not send Ack and Nak DLLP
		1	Send Ack and Nak DLLP
24 (R/W)	FCDIS	Flow Control Disable. The `PCIE_LANE_SKEW_[n].FCDIS` bit prevents the core from sending FC Data Link Layer Packets (DLLPs). This register field is sticky.	
		0	Do not send FC DLLP
		1	Send FC DLLP

Table 29-142: PCIE_LANE_SKEW_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	ISKW	Inset Lane Deskew. The `PCIE_LANE_SKEW_[n]` bit field configures the optional feature that causes the core to insert skew between lanes for test purposes. There are three bits per lane. The value is in units of one symbol time. For example, the value 010b for a lane forces a skew of two symbol times for that Lane. The maximum skew value for any lane is 5 symbol times. This register field is sticky.

Link Capabilities 2 Register

The PCIE_LNK_CAPB2_[n] register contains bits that indicate the link speed vector and port cross link status.

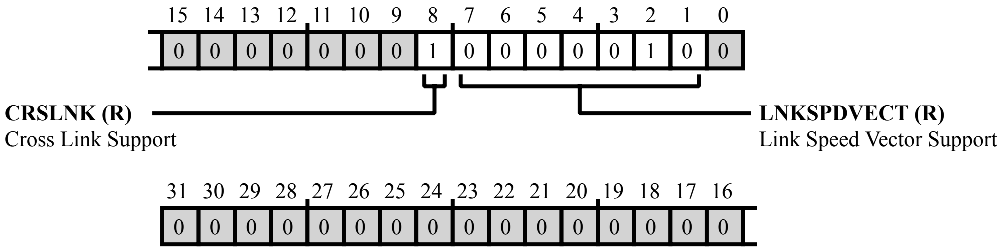

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
CRSLNK (R)
Cross Link Support
LNKSPDVECT (R)
Link Speed Vector Support
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-134: PCIE_LNK_CAPB2_[n] Register Diagram

Table 29-143: PCIE_LNK_CAPB2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	CRSLNK	Cross Link Support. When set to 1, the PCIE_LNK_CAPB2_[n].CRSLNK bit indicates that the associated port supports cross links. When set to 0 on a port that supports link speeds of 8.0 GT/s or higher, this bit indicates that the associated port does not support cross links. When cleared (=0) on a port that only supports link speeds of 2.5 GT/s or 5.0 GT/s, this bit provides no information regarding the ports level of cross link support.	
7:1 (R/NW)	LNKSPDVECT	Link Speed Vector Support. The PCIE_LNK_CAPB2_[n].LNKSPDVECT bit field indicates the supported link speed(s) of the associated port. For each bit, a value of 1 indicates that the corresponding link speed is supported; otherwise, the link speed is not supported.	
		0	2.5 GT/sec
		1	5.0 GT/sec
		2	8.0 GT/sec

Link Capabilities Register

The PCIE_LNK_CAPB_[n] register contains bits that indicate the status of links and various link capabilities.

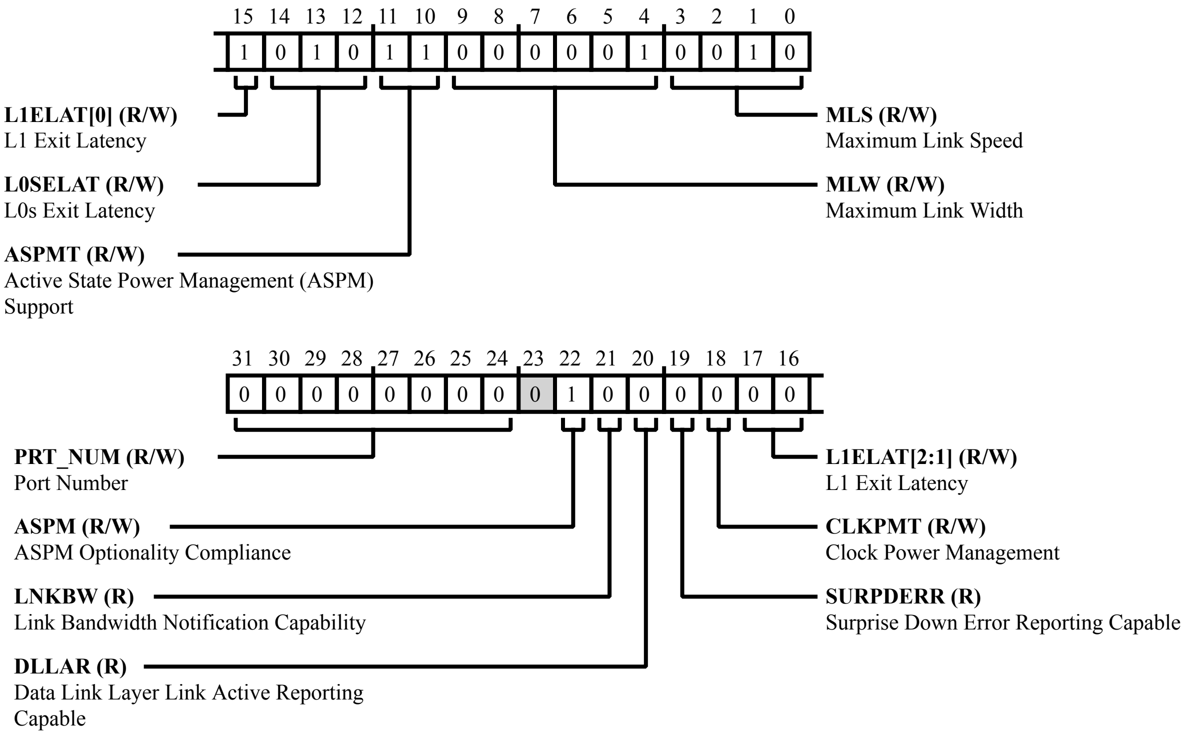

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0
L1ELAT[0] (R/W)
L1 Exit Latency
L0SELAT (R/W)
L0s Exit Latency
ASPMT (R/W)
Active State Power Management (ASPM) Support
MLS (R/W)
Maximum Link Speed
MLW (R/W)
Maximum Link Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
PRT_NUM (R/W)
Port Number
ASPM (R/W)
ASPM Optionality Compliance
LNKBW (R)
Link Bandwidth Notification Capability
DLLAR (R)
Data Link Layer Link Active Reporting Capable
L1ELAT[2:1] (R/W)
L1 Exit Latency
CLKPMT (R/W)
Clock Power Management
SURPDERR (R)
Surprise Down Error Reporting Capable

Figure 29-135: PCIE_LNK_CAPB_[n] Register Diagram

Table 29-144: PCIE_LNK_CAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	PRT_NUM	Port Number. The PCIE_LNK_CAPB_[n].PRT_NUM bit field indicates the PCI Express Port number for the given PCI Express Link. Multi-function devices associated with an upstream port must report the same value in this field for all functions.
22 (R/W)	ASPM	ASPM Optionality Compliance. The PCIE_LNK_CAPB_[n].ASPM bit must be set to 1 in all functions. Components implemented against certain earlier versions of this specification will have this bit set to 0. Software is permitted to use the value of this bit to help determine whether to enable ASPM or whether to run ASPM compliance tests.

Table 29-144: PCIE_LNK_CAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/NW)	LNKBW	Link Bandwidth Notification Capability. A value of 1 in the `PCIE_LNK_CAPB_[n].LNKBW` bit indicates support for the Link Bandwidth Notification status and interrupt mechanisms. This capability is required for all Root Ports and Switch Downstream Ports supporting links wider than x1 and/or multiple Link speeds. This bit is not applicable and is Reserved for Endpoints, PCI Express to PCI/PCI-X bridges, and upstream ports of switches. Functions that do not implement the Link Bandwidth Notification Capability must hardwire this bit to 0.
20 (R/NW)	DLLAR	Data Link Layer Link Active Reporting Capable. For a downstream port, the `PCIE_LNK_CAPB_[n].DLLAR` bit must be hardwired to 1 if the component supports the optional capability of reporting the DL_Active state of the Data Link Control and Management State Machine. For a hot-plug capable downstream port (as indicated by the Hot-Plug Capable bit of the Slot Capabilities register) or a downstream port that supports Link speeds greater than 5.0 GT/s, this bit must be hardwired to 1. For upstream ports and components that do not support this optional capability, this bit must be hardwired to 0.
19 (R/NW)	SURPDERR	Surprise Down Error Reporting Capable. For a downstream port, the `PCIE_LNK_CAPB_[n].SURPDERR` bit must be set (=1) if the component supports the optional capability of detecting and reporting a Surprise Down error condition. For upstream ports and components that do not support this optional capability, this bit must be hardwired to 0.
18 (R/W)	CLKPMT	Clock Power Management. For upstream ports, a 1 the `PCIE_LNK_CAPB_[n].CLKPMT` bit indicates that the component tolerates the removal of any reference clock(s) via the clock request (CLKREQ#) mechanism when the Link is in the L1 and L2/L3 Ready Link states. A value of 0 indicates the component does not have this capability and that reference clock(s) must not be removed in these Link states. This capability is applicable only in form factors that support clock request (CLKREQ#) capability. For a multi-Function device associated with an upstream port, each function indicates its capability independently. Power Management configuration software must only permit reference clock removal if all functions of the multi-function device indicate a 1b in this bit. For ARI devices, all functions must indicate the same value in this bit. For downstream ports, this bit must be set to 0.

Table 29-144: PCIE_LNK_CAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:15 (R/W)	L1ELAT	L1 Exit Latency. The `PCIE_LNK_CAPB_[n].L1ELAT` bit field indicates the L1 exit latency for the given PCI Express Link. The value reported indicates the length of time this Port requires to complete transition from ASPM L1 to L0. If ASPM L1 is not supported, the value is undefined. Note that exit latencies may be influenced by PCI Express reference clock configuration depending upon whether a component uses a common or separate reference clock. Multi-Function devices associated with an Upstream Port must report the same value in this field for all Functions.	
		0	Less than 1s
		1	1 s to less than 2 s
		2	2 s to less than 4 s
		3	4 s to less than 8 s
		4	8 s to less than 16 s
		5	16 s to less than 32 s
		6	32 s-64 s
		7	More than 64 s
14:12 (R/W)	L0SELAT	L0s Exit Latency. The `PCIE_LNK_CAPB_[n].L0SELAT` bit field indicates the L0s exit latency for the given PCI Express Link. The value reported indicates the length of time this port requires to complete transition from L0s to L0. If L0s is not supported, the value is undefined. Note that exit latencies may be influenced by PCI Express reference clock configuration depending upon whether a component uses a common or separate reference clock. Multi-Function devices associated with an upstream port must report the same value in this field for all functions.	
		0	Less than 64 ns
		1	64 ns to less than 128 ns
		2	128 ns to less than 256 ns
		3	256 ns to less than 512 ns
		4	512 ns to less than 1 s
		5	1 s to less than 2 s
		6	2 s-4 s
		7	More than 4 s

Table 29-144: PCIE_LNK_CAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	ASPMT	Active State Power Management (ASPM) Support. The `PCIE_LNK_CAPB_[n].ASPMT` bit field indicates the level of ASPM supported on the given PCI Express Link. Multi-function devices associated with an Upstream Port must report the same value in this field for all Functions.	
		0	No ASPM support
		1	0s support
		2	L1 supported
		3	L0s and L1 supported
9:4 (R/W)	MLW	Maximum Link Width. The `PCIE_LNK_CAPB_[n].MLW` bit field indicates the maximum link width (xN corresponding to N Lanes) implemented by the component. This value is permitted to exceed the number of lanes routed to the slot (downstream port), adapter connector (upstream port), or in the case of component-to- component connections, the actual wired connection width. Multi-Function devices associated with an upstream port must report the same value in this field for all Functions.	
		0	Reserved
		1	x1
		2	x2
		3	x4
		4	x8
		5	x12
		6	x16
		7	x32
3:0 (R/W)	MLS	Maximum Link Speed. The `PCIE_LNK_CAPB_[n].MLS` bit field indicates the maximum link speed of the associated port. The encoded value specifies a bit location in the supported link speeds vector (in the `PCIE_LNK_CAPB2_[n]` register) that corresponds to the maximum link speed. All encodings not listed below are reserved. Multi-Function devices associated with an Upstream Port must report the same value in this field for all Functions.	
		0	Supported Link Speeds Vector field bit 0
		1	Supported Link Speeds Vector field bit 1
		2	Supported Link Speeds Vector field bit 2

Table 29-144: PCIE_LNK_CAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Supported Link Speeds Vector field bit 3
		4	Supported Link Speeds Vector field bit 4
		5	Supported Link Speeds Vector field bit 5
		6	Supported Link Speeds Vector field bit 6

Link Control 2 and Status 2 Register

The `PCIE_LNK_CTLSTAT2_[n]` register is used to configure various link attributes and reports various link status.

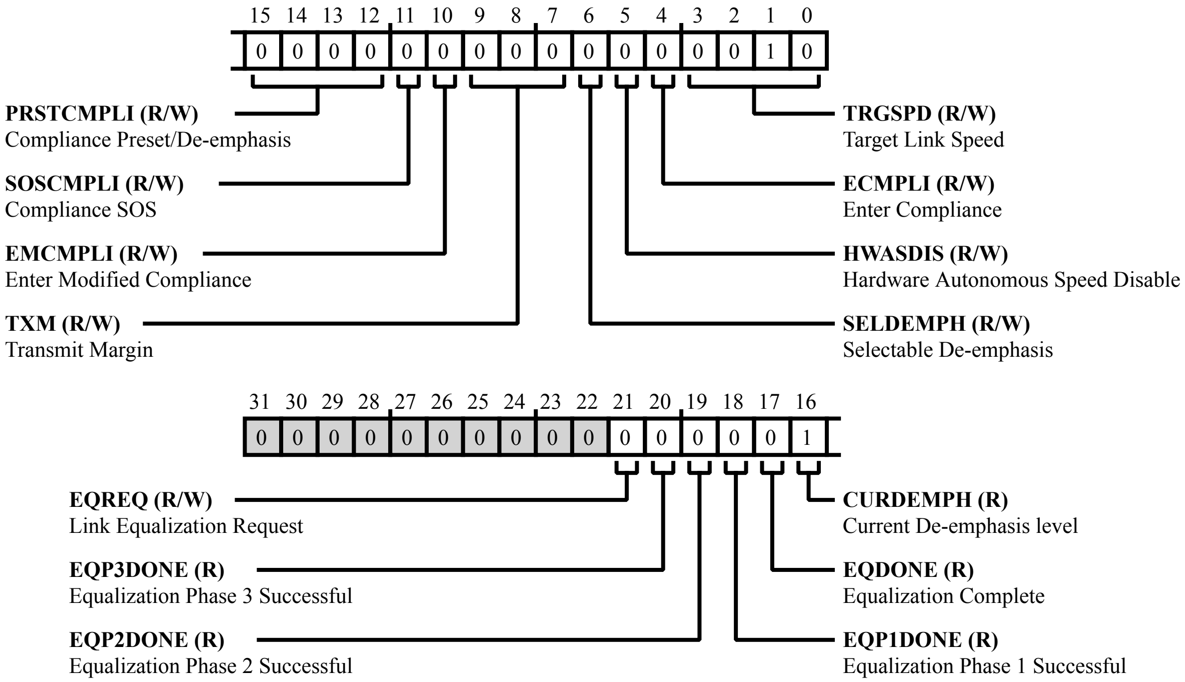

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PRSTCMPLI (R/W)
Compliance Preset/De-emphasis
SOSCMPLI (R/W)
Compliance SOS
EMCMPLI (R/W)
Enter Modified Compliance
TXM (R/W)
Transmit Margin
TRGSPD (R/W)
Target Link Speed
ECMPLI (R/W)
Enter Compliance
HWASDIS (R/W)
Hardware Autonomous Speed Disable
SELDEMPH (R/W)
Selectable De-emphasis
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
EQREQ (R/W)
Link Equalization Request
EQP3DONE (R)
Equalization Phase 3 Successful
EQP2DONE (R)
Equalization Phase 2 Successful
CURDEMPH (R)
Current De-emphasis level
EQDONE (R)
Equalization Complete
EQP1DONE (R)
Equalization Phase 1 Successful

Figure 29-136: PCIE_LNK_CTLSTAT2_[n] Register Diagram

Table 29-145: PCIE_LNK_CTLSTAT2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	EQREQ	Link Equalization Request. The `PCIE_LNK_CTLSTAT2_[n].EQREQ` bit is set by hardware to request the link equalization process to be performed on the link.
20 (R/NW)	EQP3DONE	Equalization Phase 3 Successful. When set (=1), the `PCIE_LNK_CTLSTAT2_[n].EQP3DONE` bit indicates that Phase 3 of the Transmitter Equalization procedure has successfully completed.
19 (R/NW)	EQP2DONE	Equalization Phase 2 Successful. When set (=1), the `PCIE_LNK_CTLSTAT2_[n].EQP2DONE` bit indicates that Phase 2 of the Transmitter Equalization procedure has successfully completed.
18 (R/NW)	EQP1DONE	Equalization Phase 1 Successful. When set (=1), the `PCIE_LNK_CTLSTAT2_[n].EQP1DONE` bit indicates that Phase 1 of the Transmitter Equalization procedure has successfully completed.

Table 29-145: PCIE_LNK_CTLSTAT2_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	EQDONE	Equalization Complete. For a description of this standard PCIe register field, see the PCI Express Base Specification 3.0. This register field is sticky.
16 (R/NW)	CURDEMPH	Current De-emphasis level. When the link is operating at 5.0 GT/sec speed, this bit reflects the level of de-emphasis. The value in this bit is undefined when the link is not operating at 5.0 GT/sec speed. Components that support only the 2.5 GT/sec speed are permitted to set this bit to 0. 0 -6.0 dB 1 -3.5 dB
15:12 (R/W)	PRSTCMPLI	Compliance Preset/De-emphasis. The `PCIE_LNK_CTLSTAT2_[n].PRSTCMPLI` bit field sets the de-emphasis level in Polling for 5.0 GT/sec data rate. Compliance state if the entry occurred due to the Enter Compliance bit being 1. Defined Encodings are 0001=3.5 dB and 0000=6 dB. When the Link is operating at 2.5 GT/sec, this field has no effect. Components that support only 2.5 GT/sec speed are permitted to set this field to 0000. This field is intended for debug and compliance testing purposes. System firmware and software is allowed to modify this field only during debug or compliance testing. In all other cases, the system must ensure that this field is set to the default value.
11 (R/W)	SOSCMPLI	Compliance SOS. When the `PCIE_LNK_CTLSTAT2_[n].SOSCMPLI` bit is set (=1), the LTSSM is required to send SKP Ordered Sets between sequences when sending the Compliance Pattern or Modified Compliance Pattern. For a Multi-Function device associated with an Upstream Port, the bit in Function 0 is of type RWS, and only Function 0 controls the components link behavior. In all other Functions of that device, this bit is of type RsvdP. The default value of this bit is 0. The `PCIE_LNK_CTLSTAT2_[n].SOSCMPLI` bit is applicable when the Link is operating at 2.5 GT/sec or 5.0 GT/sec data rates only. Components that support only the 2.5 GT/sec speed are permitted to hardwire this bit to 0.
10 (R/W)	EMCMPLI	Enter Modified Compliance. When the `PCIE_LNK_CTLSTAT2_[n].EMCMPLI` bit is set (=1), the device transmits Modified Compliance Pattern if the LTSSM enters Polling Compliance state. Components that support only the 2.5 GT/sec speed are permitted to set this bit to 0. The `PCIE_LNK_CTLSTAT2_[n].EMCMPLI` bit is intended for debug and compliance testing purposes. System firmware and software is allowed to modify this bit only during debug or compliance testing. In all other cases, the system must ensure that this bit is set to the default value.

Table 29-145: PCIE_LNK_CTLSTAT2_[n] Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>9:7
(R/W)</td><td>TXM</td><td colspan="2">Transmit Margin.
The `PCIE_LNK_CTLSTAT2_[n].TXM` bit field controls the value of the non-de-emphasized voltage level at the transmitter pins. This field is reset to 0x0 on entry to the LTSSM Polling.</td></tr>
<tr><td rowspan="3">6
(R/W)</td><td rowspan="3">SELDEMPH</td><td colspan="2">Selectable De-emphasis.
The `PCIE_LNK_CTLSTAT2_[n].SELDEMPH` bit is used to control the transmit deemphasis of the link in specific situations when the link is operating at 5.0 GT/sec.
When the link is not operating at 5.0 GT/sec speed, the setting of this bit has no effect. Components that support only the 2.5 GT/sec speed are permitted to hardwire this bit to 0. This bit is not applicable and reserved for Endpoints, PCI Express to PCI/PCI-X bridges, and upstream ports of switches.</td></tr>
<tr><td>0</td><td>-6 dB</td></tr>
<tr><td>1</td><td>-3.5 dB</td></tr>
<tr><td>5
(R/W)</td><td>HWASDIS</td><td colspan="2">Hardware Autonomous Speed Disable.
When the `PCIE_LNK_CTLSTAT2_[n].HWASDIS` bit is set (=1), hardware cannot change the link speed for device-specific reasons other than attempting to correct unreliable link operation. Initial transition to the highest supported common link speed is not blocked by this bit.</td></tr>
<tr><td>4
(R/W)</td><td>ECMPLI</td><td colspan="2">Enter Compliance.
When set (=1) the `PCIE_LNK_CTLSTAT2_[n].ECMPLI` bit allows software to force a link to enter Compliance mode (at the speed indicated in the Target Link Speed field and the de-emphasis/preset level indicated by the Compliance Preset/De-emphasis field) in both components on a link and then initiating a hot reset on the link.</td></tr>
</table>

Table 29-145: PCIE_LNK_CTLSTAT2_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	TRGSPD	Target Link Speed. For downstream ports, the `PCIE_LNK_CTLSTAT2_[n].TRGSPD` bit field sets an upper limit on Link operational speed by restricting the values advertised by the Upstream component in its training sequences. The encoded value specifies a bit location in the Supported Link Speeds Vector (in the Link Capabilities 2 register) that corresponds to the desired target Link speed. All encodings not listed below are reserved. If a value is written to the `PCIE_LNK_CTLSTAT2_[n].TRGSPD` bit field that does not correspond to a supported speed (as indicated by the Supported Link Speeds Vector), the result is undefined. If either of the Enter Compliance or Enter Modified Compliance bits are implemented, then this field must also be implemented. The default value of this field is the highest Link speed supported by the component (as reported in the `PCIE_LNK_CAPB_[n].MLS` bit field) unless the corresponding platform/form factor requires a different default value. For both upstream and downstream ports, this field is used to set the target compliance mode speed when software is using the Enter Compliance bit to force a Link into compliance mode. Components that support only the 2.5 GT/s speed are permitted to set this field to 0000.	
		0	Supported Link Speeds Vector field bit 0
		1	Supported Link Speeds Vector field bit 1
		2	Supported Link Speeds Vector field bit 2
		3	Supported Link Speeds Vector field bit 3
		4	Supported Link Speeds Vector field bit 4
		5	Supported Link Speeds Vector field bit 5
		6	Supported Link Speeds Vector field bit 6

Link Control and Status Register

The `PCIE_LNK_CTLSTAT_[n]` register controls PCI Express Link specific parameters.

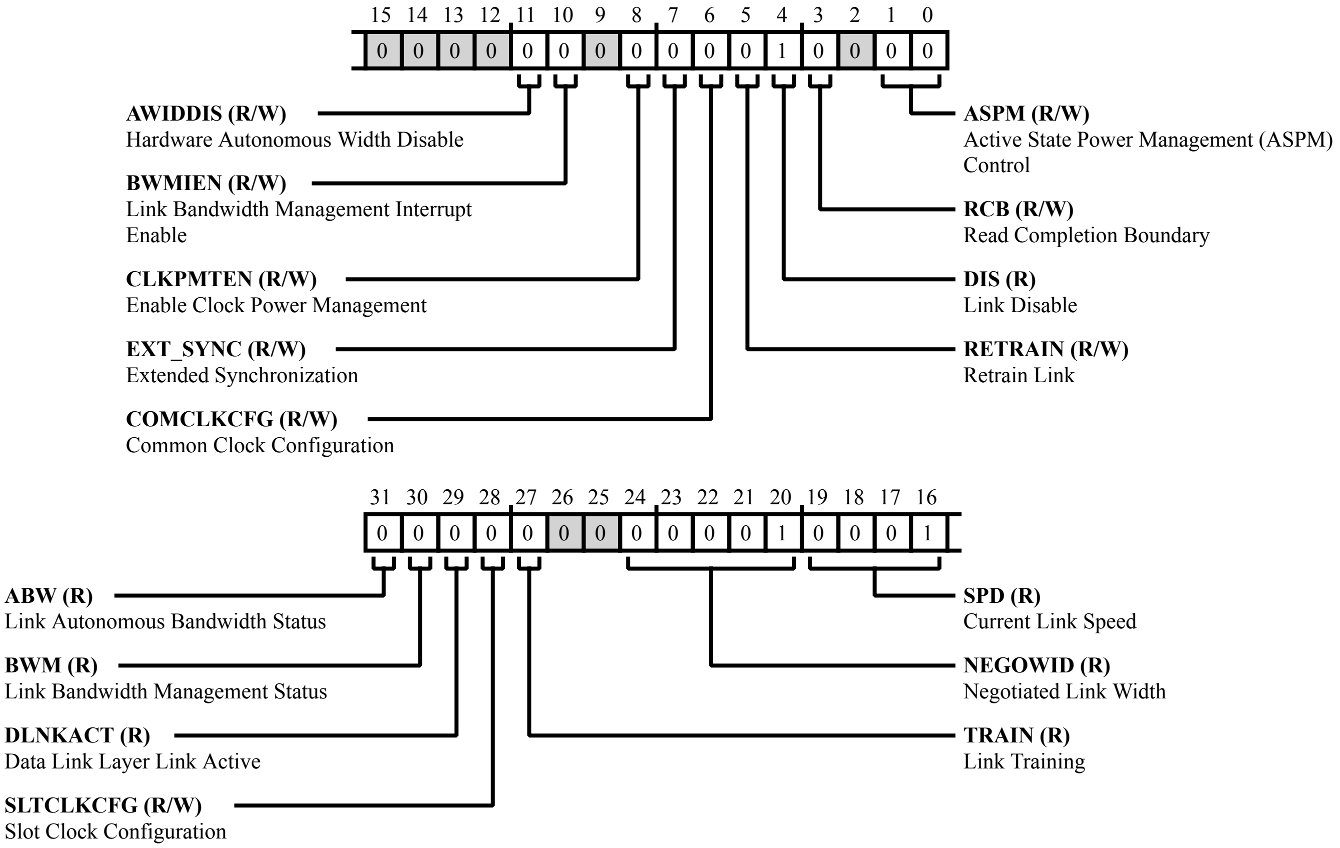

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
AWIDDIS (R/W)
Hardware Autonomous Width Disable
BWMIEN (R/W)
Link Bandwidth Management Interrupt Enable
CLKPMTEN (R/W)
Enable Clock Power Management
EXT_SYNC (R/W)
Extended Synchronization
COMCLKCFG (R/W)
Common Clock Configuration
ASPM (R/W)
Active State Power Management (ASPM) Control
RCB (R/W)
Read Completion Boundary
DIS (R)
Link Disable
RETRAIN (R/W)
Retrain Link
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
ABW (R)
Link Autonomous Bandwidth Status
BWM (R)
Link Bandwidth Management Status
DLNKACT (R)
Data Link Layer Link Active
SLTCLKCFG (R/W)
Slot Clock Configuration
SPD (R)
Current Link Speed
NEGOWID (R)
Negotiated Link Width
TRAIN (R)
Link Training

Figure 29-137: PCIE_LNK_CTLSTAT_[n] Register Diagram

Table 29-146: PCIE_LNK_CTLSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	ABW	Link Autonomous Bandwidth Status. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].ABW` bit, see the PCI Express Base Specification 3.0.
30 (R/NW)	BWM	Link Bandwidth Management Status. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].BWM` bit, see the PCI Express Base Specification 3.0.
29 (R/NW)	DLNKACT	Data Link Layer Link Active. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].DLNKACT` bit, see the PCI Express Base Specification 3.0.

Table 29-146: PCIE_LNK_CTLSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	SLTCLKCFG	Slot Clock Configuration. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].SLTCLKCFG` bit, see the PCI Express Base Specification 3.0. The access attributes of this field are as follows: • Wire: No access. • Dbi: R/W • Dbi2: HWINIT
27 (R/NW)	TRAIN	Link Training. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].TRAIN` bit, see the PCI Express Base Specification 3.0.
24:20 (R/NW)	NEGOWID	Negotiated Link Width. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].NEGOWID` bit, see the PCI Express Base Specification 3.0.
19:16 (R/NW)	SPD	Current Link Speed. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].SPD` bit, see the PCI Express Base Specification 3.0.
11 (R/W)	AWIDDIS	Hardware Autonomous Width Disable. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].AWIDDIS` bit, see the PCI Express Base Specification 3.0.
10 (R/W)	BWMIEN	Link Bandwidth Management Interrupt Enable. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].BWMIEN` bit, see the PCI Express Base Specification 3.0.
8 (R/W)	CLKPMTEN	Enable Clock Power Management. For a description of this standard PCIe register field, see the <i>PCI Express Base Specification 3.0</i>.
7 (R/W)	EXT_SYNC	Extended Synchronization. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].EXT_SYNC` bit, see the PCI Express Base Specification 3.0.
6 (R/W)	COMCLKCFG	Common Clock Configuration. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].COMCLKCFG` bit, see the PCI Express Base Specification 3.0.
5 (R/W)	RETRAIN	Retrain Link. For a description of the PCIe standard `PCIE_LNK_CTLSTAT_[n].RETRAIN` bit, see the PCI Express Base Specification 3.0.

Table 29-146: PCIE_LNK_CTLSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/NW)	DIS	Link Disable. The `PCIE_LNK_CTLSTAT_[n].DIS` bit disables the link by directing the LTSSM to the disabled state. This bit is reserved on Endpoints, PCI Express to PCI/PCI-X bridges, and upstream ports of switches. Writes to this bit are immediately reflected in the value read from the bit, regardless of actual link state.	
3 (R/W)	RCB	Read Completion Boundary. The `PCIE_LNK_CTLSTAT_[n].RCB` bit indicates the RCB value for the Root Port.	
		0	64-byte
		1	128-byte
1:0 (R/W)	ASPM	Active State Power Management (ASPM) Control. The `PCIE_LNK_CTLSTAT_[n].ASPM` bit field controls the level of ASPM enabled on the given PCI Express Link.	
		0	Disabled
		1	L0s entry enabled
		2	L1 entry enabled
		3	L0s and L1 entry enabled

DBI Read-Only Write Enable Register

The `PCIE_MISC_CTL1_[n]` register allows programs to write applicable hardware init and read-only bits from the local application through the DBI.

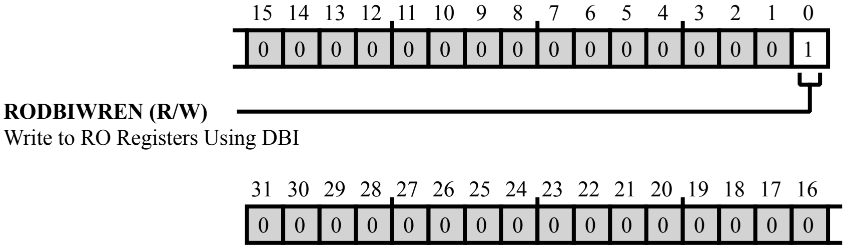

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
RODBIWREN (R/W)
Write to RO Registers Using DBI
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-138: PCIE_MISC_CTL1_[n] Register Diagram

Table 29-147: PCIE_MISC_CTL1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	RODBIWREN	Write to RO Registers Using DBI. The `PCIE_MISC_CTL1_[n].RODBIWREN` bit, when set (=1), allows programs to write applicable hardware init and read-only bits from the local application through the DBI.

MSI Capability ID, Next Pointer and Control Register

The `PCIE_MSI_CAPB_NPTR_[n]` register controls MSI capability.

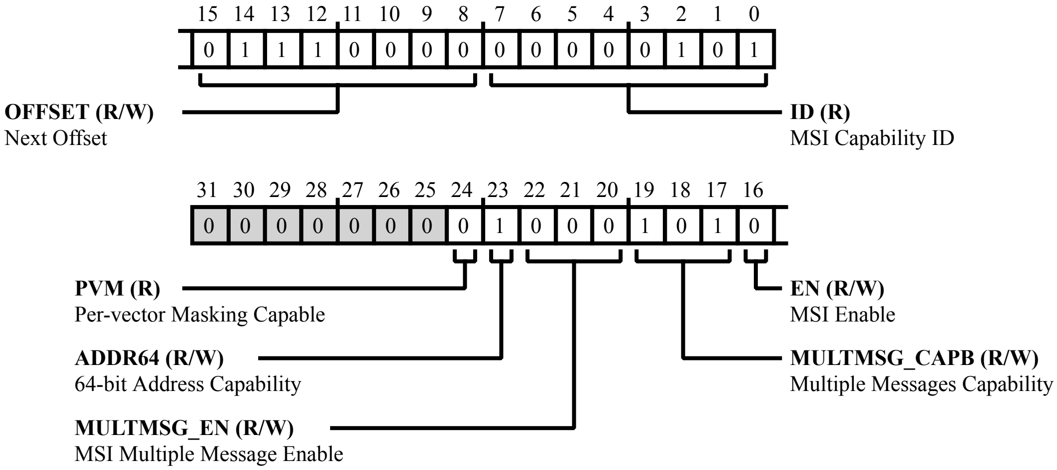

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1
OFFSET (R/W)
Next Offset
ID (R)
MSI Capability ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0
PVM (R)
Per-vector Masking Capable
ADDR64 (R/W)
64-bit Address Capability
MULTMSG_EN (R/W)
MSI Multiple Message Enable
EN (R/W)
MSI Enable
MULTMSG_CAPB (R/W)
Multiple Messages Capability

Figure 29-139: PCIE_MSI_CAPB_NPTR_[n] Register Diagram

Table 29-148: PCIE_MSI_CAPB_NPTR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/NW)	PVM	Per-vector Masking Capable. The `PCIE_MSI_CAPB_NPTR_[n].PVM` bit determines whether software has the ability to mask individual interrupts with the PVM capability structure (=1) or does not implement PVM (=0).	
23 (R/W)	ADDR64	64-bit Address Capability. The `PCIE_MSI_CAPB_NPTR_[n].ADDR64` bit indicates if the function is capable (=1) or is not capable (=0) of generating a 64-bit message address. This bit is read only.	
22:20 (R/W)	MULTMSG_EN	MSI Multiple Message Enable. System software writes to the `PCIE_MSI_CAPB_NPTR_[n].MULTMSG_EN` bit field to indicate the number of allocated messages (equal to or less than the number of requested messages). The number of allocated messages is aligned to a power of two. If a function requests four messages (indicated by a Multiple Message Capable encoding of 010), system software can allocate either four, two, or one message by writing a 010, 001, or 000 to this field, respectively. When MSI is enabled, a device is allocated at least 1 message. This fields state after reset is 000. This field is read/write.	
		0	1
		1	2
		2	4
		3	8

Table 29-148: PCIE_MSI_CAPB_NPTR_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	16
		5	32
		6	Reserved
		7	Reserved
19:17 (R/W)	MULTMSG_CAPB	Multiple Messages Capability. System software reads the `PCIE_MSI_CAPB_NPTR_[n].MULTMSG_CAPB` bit field to determine the number of requested messages. The number of requested messages must be aligned to a power of two (if a function requires three messages, it requests four by initializing this field to 010). This field is read only.	
		0	1
		1	2
		2	4
		3	8
		4	16
		5	32
		6	Reserved
		7	Reserved
16 (R/W)	EN	MSI Enable. If the `PCIE_MSI_CAPB_NPTR_[n].EN` bit =1, the function is permitted to use MSI to request service and is prohibited from using its INTx# pin (if implemented). System configuration software sets this bit to enable MSI. A device driver is prohibited from writing this bit to mask a functions service request. This bits state after reset is 0 (MSI is disabled). This bit is read/write.	
		0	The function is prohibited from using MSI to request service
		1	The function can use MSI to request service
15:8 (R/W)	OFFSET	Next Offset.	
7:0 (R/NW)	ID	MSI Capability ID. The value of 0x5 in the `PCIE_MSI_CAPB_NPTR_[n].ID` bit field identifies the function as message signaled interrupt capable. This field is read only.	

MSI Capability Offset Register

PCI MSI Capability Offset

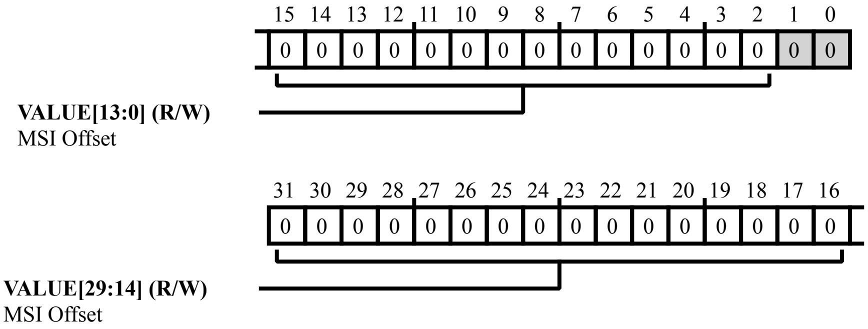

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[13:0] (R/W)
MSI Offset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:14] (R/W)
MSI Offset

Figure 29-140: PCIE_MSI_CAPB_OFF4_[n] Register Diagram

Table 29-149: PCIE_MSI_CAPB_OFF4_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/W)	VALUE	MSI Offset. Message Address [31:0]

MSI Capability Offset Register

PCI MSI Capability DW2 (64-bit Address).

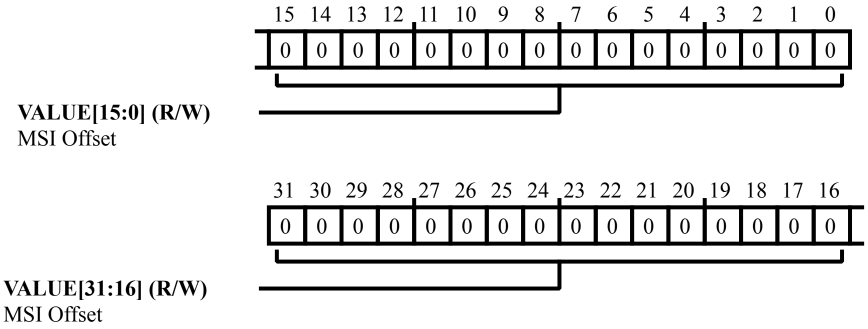

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Offset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Offset

Figure 29-141: PCIE_MSI_CAPB_OFF8_[n] Register Diagram

Table 29-150: PCIE_MSI_CAPB_OFF8_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Offset. Message Address [63:32]

MSI Capability Offset Register

PCI MSI Capability DW3 (64-bit Address).

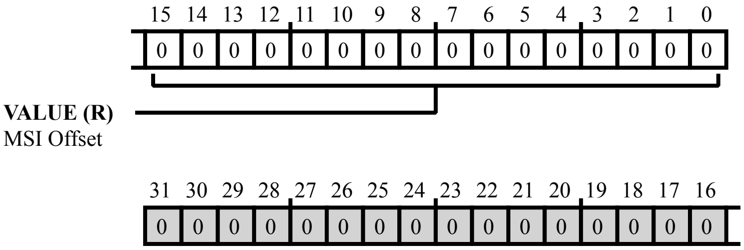

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
MSI Offset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-142: PCIE_MSI_CAPB_OFFC_[n] Register Diagram

Table 29-151: PCIE_MSI_CAPB_OFFC_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	MSI Offset. For a description of this standard PCIe register field, see the PCI Express Base Specification 3.0

MSI Controller Lower Address Register

The `PCIE_MSI_CTL_LADDR_[n]` register contains the system specified lower address for MSI memory write transaction termination. Within the SCB bridge, every received memory write request is examined to see if it targets the MSI address that has been specified in the MSI Controller Address Register and also to see if it satisfies the definition of an MSI interrupt request. When these conditions are satisfied the memory write request is marked as an MSI request.

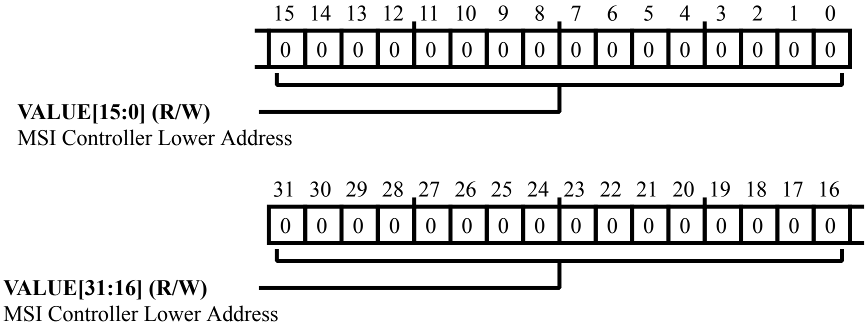

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Controller Lower Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Controller Lower Address

Figure 29-143: PCIE_MSI_CTL_LADDR_[n] Register Diagram

Table 29-152: PCIE_MSI_CTL_LADDR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Controller Lower Address. The `PCIE_MSI_CTL_LADDR_[n].VALUE` bit field contains the system specified address for MSI memory write transaction termination.

MSI Controller Upper Address Register

The `PCIE_MSI_CTL_UADDR_[n]` register contains the system specified upper address for MSI memory write transaction termination. Allows functions to support a 64-bit MSI address.

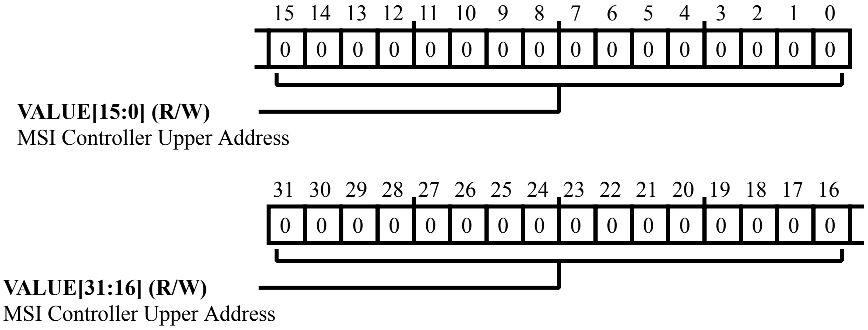

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Controller Upper Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Controller Upper Address

Figure 29-144: PCIE_MSI_CTL_UADDR_[n] Register Diagram

Table 29-153: PCIE_MSI_CTL_UADDR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Controller Upper Address. The `PCIE_MSI_CTL_UADDR_[n].VALUE` bit field contains the system specified upper address for MSI memory write transaction termination.

MSI Controller General-Purpose IO Register

The contents of the `PCIE_MSI_GPIO_IO_[n]` register drives the top-level GPIO msi_ctrl_io[31:0] pins.

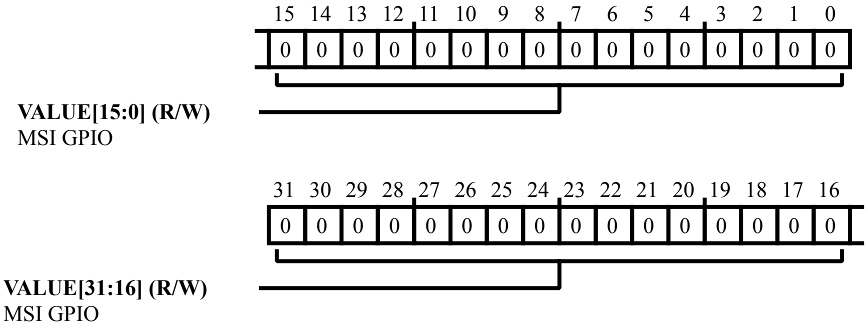

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI GPIO
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI GPIO

Figure 29-145: PCIE_MSI_GPIO_IO_[n] Register Diagram

Table 29-154: PCIE_MSI_GPIO_IO_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI GPIO. The contents of the `PCIE_MSI_GPIO_IO_[n].VALUE` bit field drives the top-level GPIO msi_ctrl_io[31:0] pins.

MSI Controller Interrupt 0 Enable Register

The `PCIE_MSI_IEN0_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

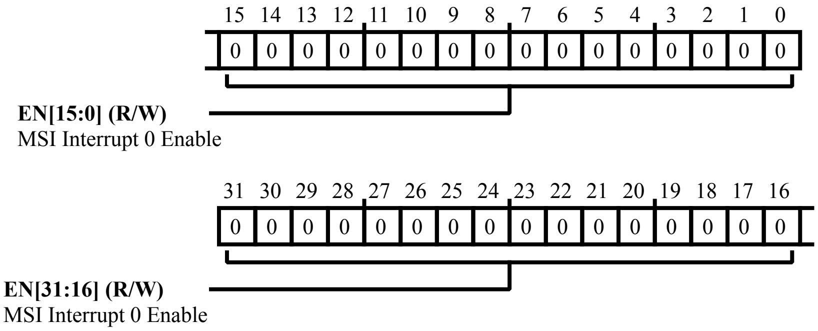

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 0 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 0 Enable

Figure 29-146: PCIE_MSI_IEN0_[n] Register Diagram

Table 29-155: PCIE_MSI_IEN0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 0 Enable. The `PCIE_MSI_IEN0_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 1 Enable Register

The `PCIE_MSI_IEN1_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

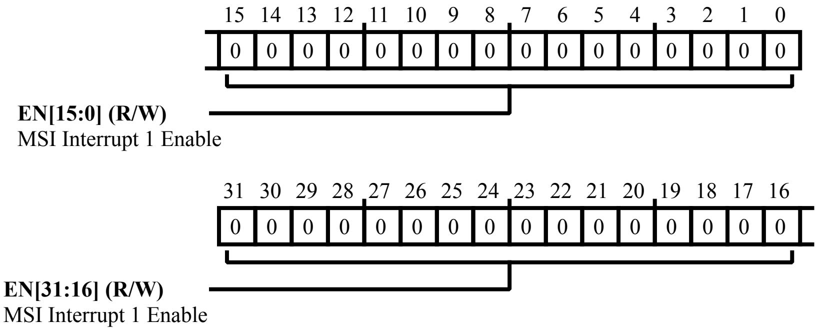

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 1 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 1 Enable

Figure 29-147: PCIE_MSI_IEN1_[n] Register Diagram

Table 29-156: PCIE_MSI_IEN1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 1 Enable. The `PCIE_MSI_IEN1_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 2 Enable Register

The `PCIE_MSI_IEN2_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

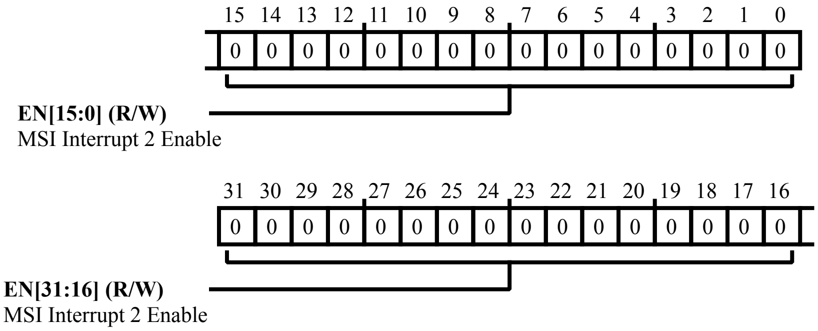

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 2 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 2 Enable

Figure 29-148: PCIE_MSI_IEN2_[n] Register Diagram

Table 29-157: PCIE_MSI_IEN2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 2 Enable. The `PCIE_MSI_IEN2_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 3 Enable Register

The `PCIE_MSI_IEN3_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

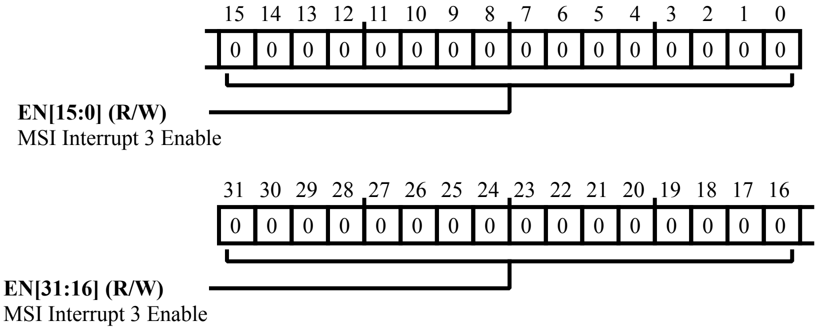

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 3 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 3 Enable

Figure 29-149: PCIE_MSI_IEN3_[n] Register Diagram

Table 29-158: PCIE_MSI_IEN3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 3 Enable. The `PCIE_MSI_IEN3_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 4 Enable Register

The `PCIE_MSI_IEN4_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

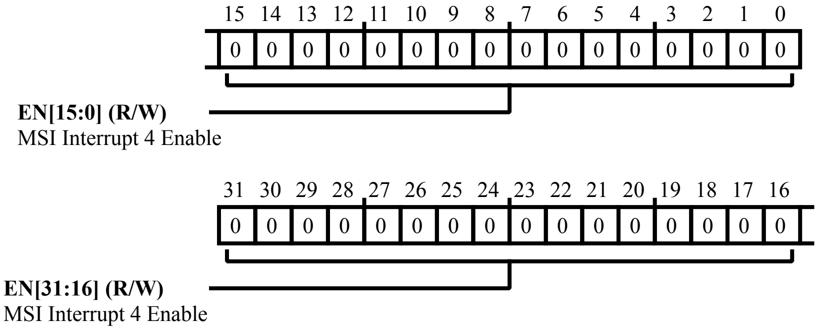

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 4 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 4 Enable

Figure 29-150: PCIE_MSI_IEN4_[n] Register Diagram

Table 29-159: PCIE_MSI_IEN4_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 4 Enable. The `PCIE_MSI_IEN4_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 5 Enable Register

The `PCIE_MSI_IEN5_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

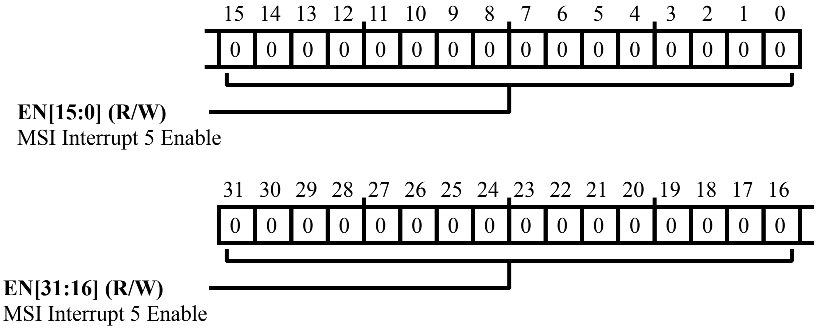

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 5 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 5 Enable

Figure 29-151: PCIE_MSI_IEN5_[n] Register Diagram

Table 29-160: PCIE_MSI_IEN5_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 5 Enable. The `PCIE_MSI_IEN5_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 6 Enable Register

The `PCIE_MSI_IEN6_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

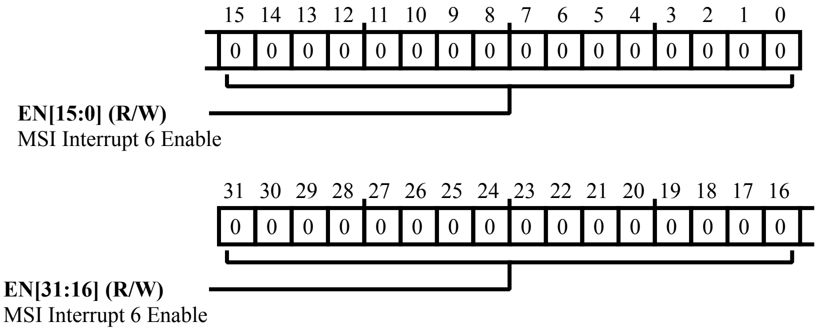

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 6 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 6 Enable

Figure 29-152: PCIE_MSI_IEN6_[n] Register Diagram

Table 29-161: PCIE_MSI_IEN6_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 6 Enable. The `PCIE_MSI_IEN6_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 7 Enable Register

The `PCIE_MSI_IEN7_[n]` register specifies which interrupts are enabled. When an MSI is received from a disabled interrupt, no status bit gets set in MSI controller interrupt status register. Each bit corresponds to a single MSI interrupt vector.

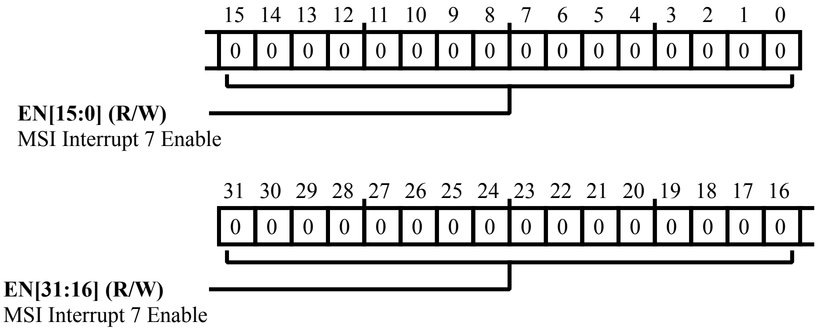

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[15:0] (R/W)
MSI Interrupt 7 Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN[31:16] (R/W)
MSI Interrupt 7 Enable

Figure 29-153: PCIE_MSI_IEN7_[n] Register Diagram

Table 29-162: PCIE_MSI_IEN7_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	EN	MSI Interrupt 7 Enable. The `PCIE_MSI_IEN7_[n].EN` bit field specifies which interrupts are enabled.

MSI Controller Interrupt 0 Mask Register

The `PCIE_MSI_IMSK0_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

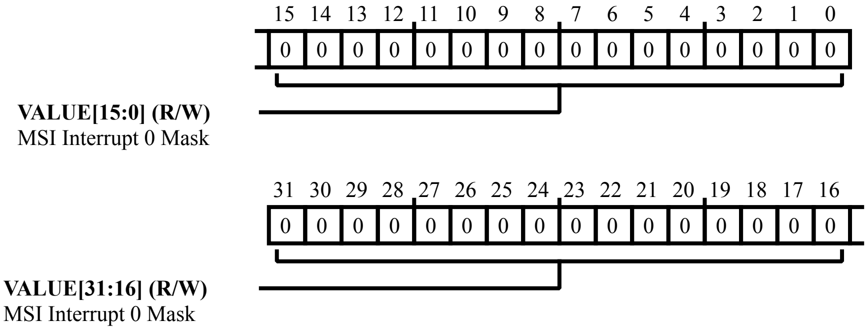

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 0 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 0 Mask

Figure 29-154: PCIE_MSI_IMSK0_[n] Register Diagram

Table 29-163: PCIE_MSI_IMSK0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 0 Mask. The `PCIE_MSI_IMSK0_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 1 Mask Register

The `PCIE_MSI_IMSK1_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

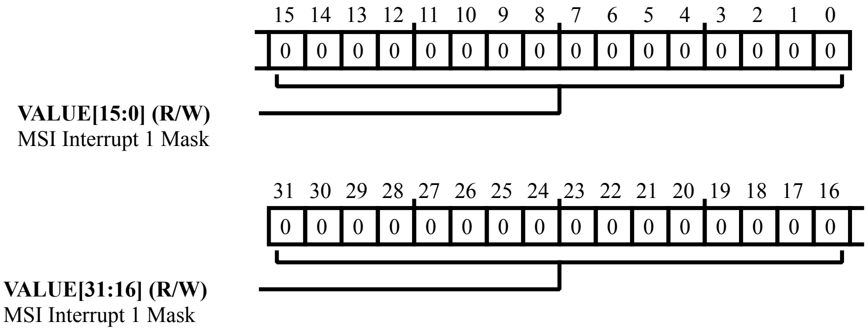

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 1 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 1 Mask

Figure 29-155: PCIE_MSI_IMSK1_[n] Register Diagram

Table 29-164: PCIE_MSI_IMSK1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 1 Mask. The `PCIE_MSI_IMSK1_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 2 Mask Register

The `PCIE_MSI_IMSK2_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

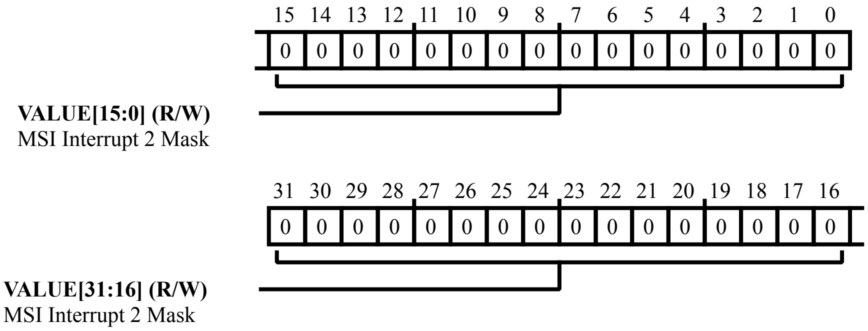

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 2 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 2 Mask

Figure 29-156: PCIE_MSI_IMSK2_[n] Register Diagram

Table 29-165: PCIE_MSI_IMSK2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 2 Mask. The `PCIE_MSI_IMSK2_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 3 Mask Register

The `PCIE_MSI_IMSK3_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

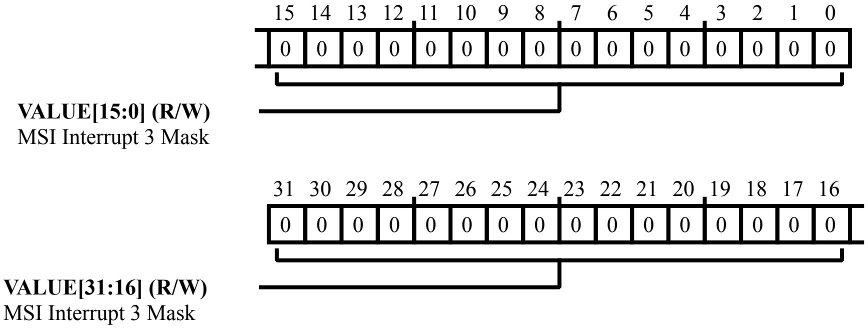

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 3 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 3 Mask

Figure 29-157: PCIE_MSI_IMSK3_[n] Register Diagram

Table 29-166: PCIE_MSI_IMSK3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 3 Mask. The `PCIE_MSI_IMSK3_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 4 Mask Register

The `PCIE_MSI_IMSK4_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

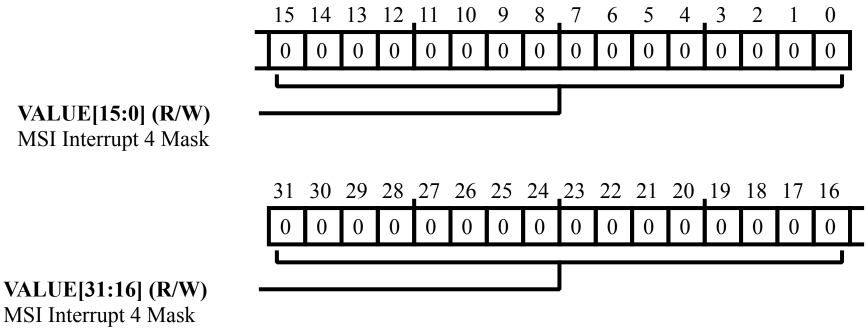

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 4 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 4 Mask

Figure 29-158: PCIE_MSI_IMSK4_[n] Register Diagram

Table 29-167: PCIE_MSI_IMSK4_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 4 Mask. The `PCIE_MSI_IMSK4_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 5 Mask Register

The `PCIE_MSI_IMSK5_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

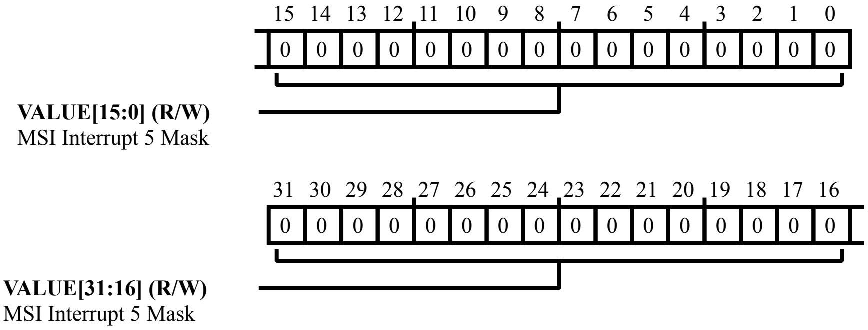

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 5 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 5 Mask

Figure 29-159: PCIE_MSI_IMSK5_[n] Register Diagram

Table 29-168: PCIE_MSI_IMSK5_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 5 Mask. The `PCIE_MSI_IMSK5_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 6 Mask Register

The `PCIE_MSI_IMSK6_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

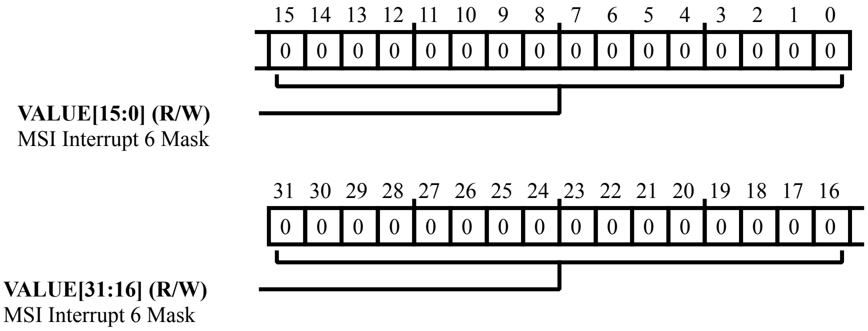

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 6 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 6 Mask

Figure 29-160: PCIE_MSI_IMSK6_[n] Register Diagram

Table 29-169: PCIE_MSI_IMSK6_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 6 Mask. The `PCIE_MSI_IMSK6_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 7 Mask Register

The `PCIE_MSI_IMSK7_[n]` register allows enabled interrupts to be masked. When an MSI is received for a masked interrupt, the corresponding status bit gets set in the interrupt status register but the msi_ctrl_int output is not set high. Each bit corresponds to a single MSI interrupt vector.

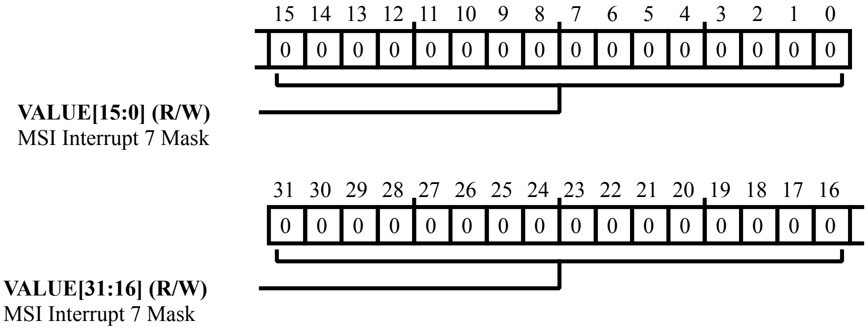

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 7 Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 7 Mask

Figure 29-161: PCIE_MSI_IMSK7_[n] Register Diagram

Table 29-170: PCIE_MSI_IMSK7_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 7 Mask. The `PCIE_MSI_IMSK7_[n].VALUE` bit field masks enabled interrupts.

MSI Controller Interrupt 0 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT0_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

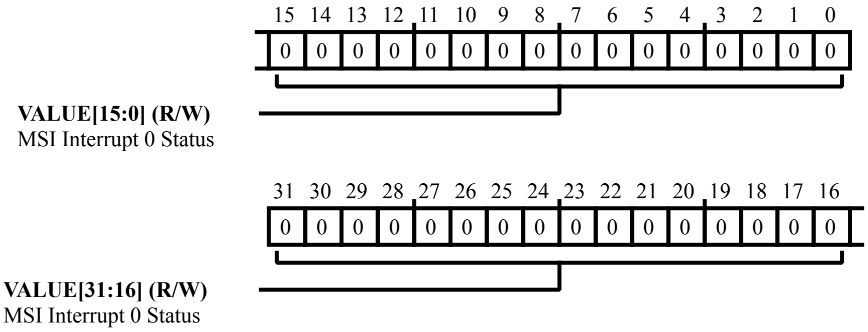

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 0 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 0 Status

Figure 29-162: PCIE_MSI_ISTAT0_[n] Register Diagram

Table 29-171: PCIE_MSI_ISTAT0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 0 Status. The `PCIE_MSI_ISTAT0_[n].VALUE` bit field indicates when an MSI is detected for EP0.

MSI Controller Interrupt 1 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT1_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

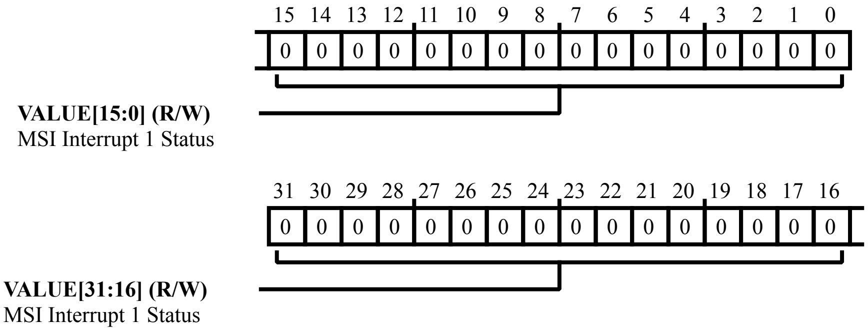

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 1 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 1 Status

Figure 29-163: PCIE_MSI_ISTAT1_[n] Register Diagram

Table 29-172: PCIE_MSI_ISTAT1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 1 Status. The `PCIE_MSI_ISTAT1_[n].VALUE` bit field indicates when an MSI is detected for EP1.

MSI Controller Interrupt 2 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT2_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

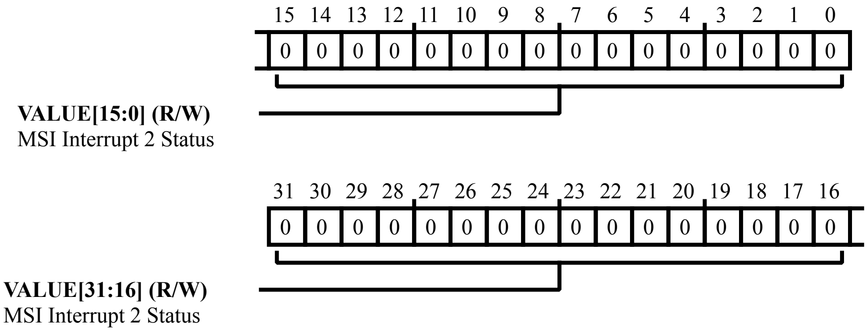

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 2 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 2 Status

Figure 29-164: PCIE_MSI_ISTAT2_[n] Register Diagram

Table 29-173: PCIE_MSI_ISTAT2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 2 Status. The `PCIE_MSI_ISTAT2_[n].VALUE` bit field indicates when an MSI is detected for EP2.

MSI Controller Interrupt 3 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT3_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

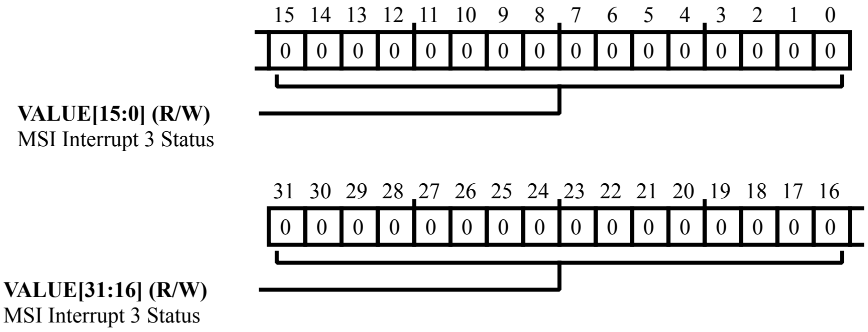

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 3 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 3 Status

Figure 29-165: PCIE_MSI_ISTAT3_[n] Register Diagram

Table 29-174: PCIE_MSI_ISTAT3_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 3 Status. The `PCIE_MSI_ISTAT3_[n].VALUE` bit field indicates when an MSI is detected for EP3.

MSI Controller Interrupt 4 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT4_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

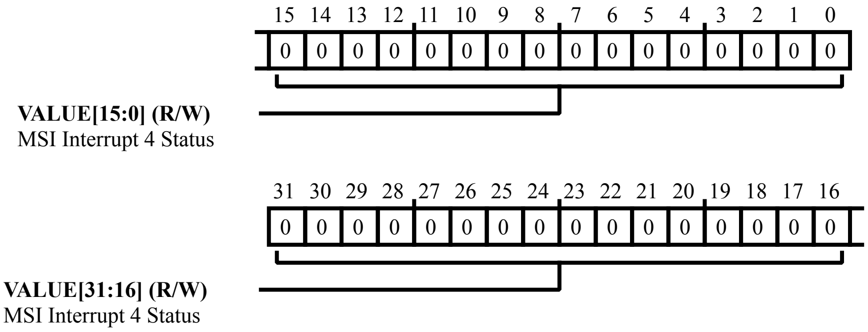

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 4 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 4 Status

Figure 29-166: PCIE_MSI_ISTAT4_[n] Register Diagram

Table 29-175: PCIE_MSI_ISTAT4_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 4 Status. The `PCIE_MSI_ISTAT4_[n].VALUE` bit field indicates when an MSI is detected for EP4.

MSI Controller Interrupt 5 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT5_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

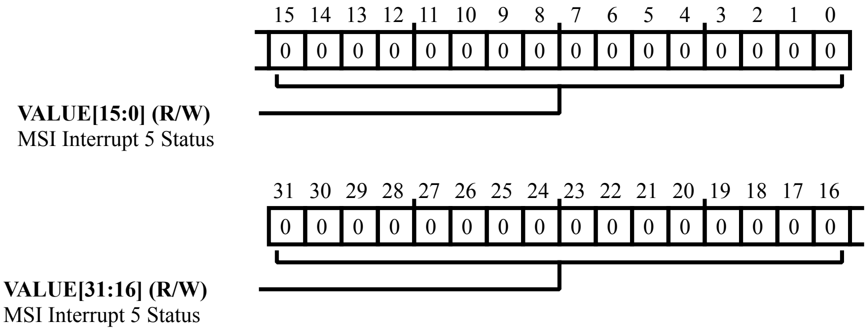

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 5 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 5 Status

Figure 29-167: PCIE_MSI_ISTAT5_[n] Register Diagram

Table 29-176: PCIE_MSI_ISTAT5_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 5 Status. The `PCIE_MSI_ISTAT5_[n].VALUE` bit field indicates when an MSI is detected for EP5.

MSI Controller Interrupt 6 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT6_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

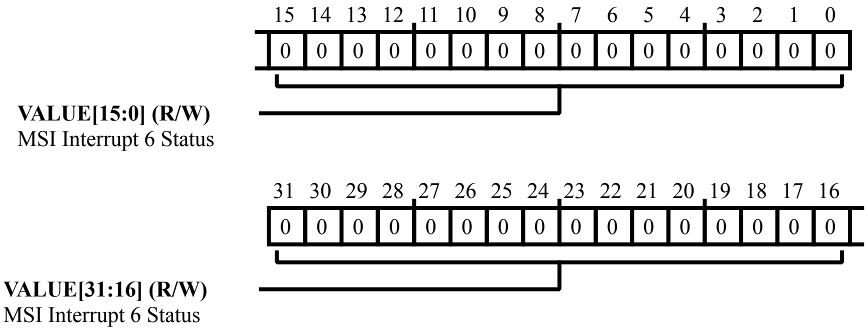

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 6 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 6 Status

Figure 29-168: PCIE_MSI_ISTAT6_[n] Register Diagram

Table 29-177: PCIE_MSI_ISTAT6_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 6 Status. The `PCIE_MSI_ISTAT6_[n].VALUE` bit field indicates when an MSI is detected for EP6.

MSI Controller Interrupt 7 Status Register

When an MSI is detected for End Point 0, one bit in the `PCIE_MSI_ISTAT7_[n]` register is set. The decoding of the data payload of the MSI memory write request determines which bit is set. A status bit is cleared by writing a 1 to the bit. Each bit corresponds to a single MSI interrupt vector.

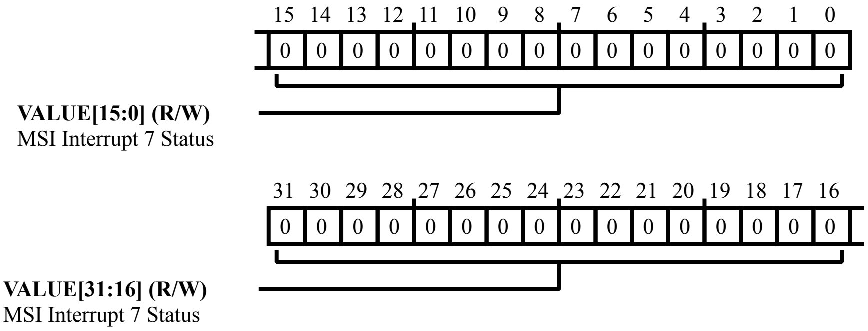

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
MSI Interrupt 7 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
MSI Interrupt 7 Status

Figure 29-169: PCIE_MSI_ISTAT7_[n] Register Diagram

Table 29-178: PCIE_MSI_ISTAT7_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MSI Interrupt 7 Status. The `PCIE_MSI_ISTAT7_[n].VALUE` bit field indicates when an MSI is detected for EP7.

TX De-emphasis Parameters Register

The `PCIE_PHY_TXDEEMPH` register can be used to modify the de-emphasis level of the transmitter.

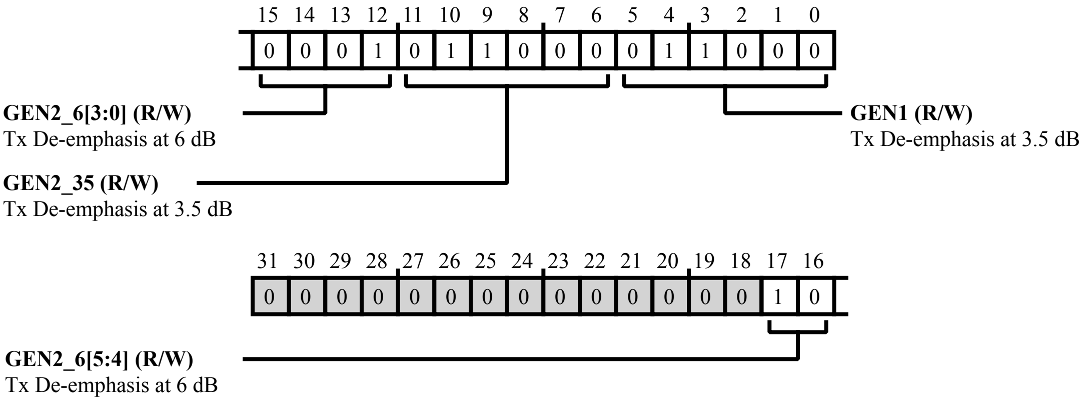

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0
GEN2_6[3:0] (R/W)
Tx De-emphasis at 6 dB
GEN2_35 (R/W)
Tx De-emphasis at 3.5 dB
GEN1 (R/W)
Tx De-emphasis at 3.5 dB
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
GEN2_6[5:4] (R/W)
Tx De-emphasis at 6 dB

Figure 29-170: PCIE_PHY_TXDEEMPH Register Diagram

Table 29-179: PCIE_PHY_TXDEEMPH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:12 (R/W)	GEN2_6	Tx De-emphasis at 6 dB. The `PCIE_PHY_TXDEEMPH.GEN2_6` bit field is a static value that sets the Tx driver de-emphasis value in the case where pipeP_tx_deemph is set to 1'b0 and the PHY is running at the Gen2 rate. This bus is provided for completeness and as a second potential launch amplitude.
11:6 (R/W)	GEN2_35	Tx De-emphasis at 3.5 dB. The `PCIE_PHY_TXDEEMPH.GEN2_35` bit field is a static value that sets the Tx driver de-emphasis value in the case where pipeP_tx_deemph is set to 1'b1 (default setting) and the PHY is running at the Gen2 rate. To enable tuning at the board level for Rx eye compliance, connect this signal to a register.
5:0 (R/W)	GEN1	Tx De-emphasis at 3.5 dB. The `PCIE_PHY_TXDEEMPH.GEN1` bit field is a static value that sets the Tx driver de-emphasis value in the case where pipeP_tx_deemph is set to 1'b1 (default setting) and the PHY is running at the Gen1 rate. To enable tuning at the board level for Rx eye compliance, connect this signal to a register.

TX Launch Amplitude Register

The `PCIE_PHY_TXSWING` register is used to modify the voltage swing level of the transmitter.

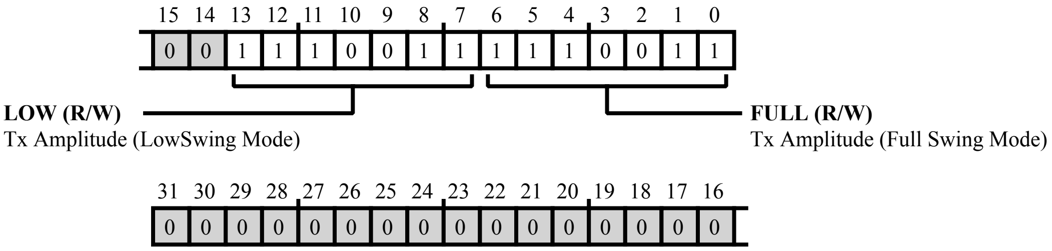

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 1 0 0 1 1 1 1 1 0 0 1 1
LOW (R/W)
Tx Amplitude (LowSwing Mode)
FULL (R/W)
Tx Amplitude (Full Swing Mode)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-171: PCIE_PHY_TXSWING Register Diagram

Table 29-180: PCIE_PHY_TXSWING Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:7 (R/W)	LOW	Tx Amplitude (LowSwing Mode). The `PCIE_PHY_TXSWING.LOW` bit field is a static value that sets the launch amplitude of the transmitter when pipeP_tx_swing is set to 1'b0 (default state). To enable tuning at the board level for Rx eye compliance, connect this signal to a register.
6:0 (R/W)	FULL	Tx Amplitude (Full Swing Mode). The `PCIE_PHY_TXSWING.FULL` bit field is a static value that sets the launch amplitude of the transmitter when pipeP_tx_swing is set to 1'b0 (default state). To enable tuning at the board level for Rx eye compliance, connect this signal to a register.

Port Logic Link Control Register

The `PCIE_PLCTL_[n]` register controls various functions of the port logic including DLL use, loopback control, and reset recovery.

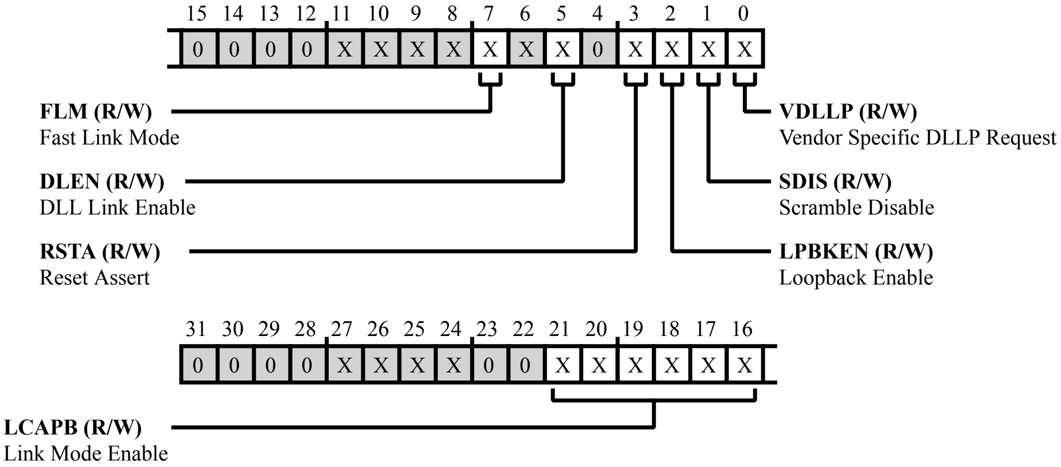

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 X X X X X X X 0 X X X X
FLM (R/W)
Fast Link Mode
DLEN (R/W)
DLL Link Enable
RSTA (R/W)
Reset Assert
VDLLP (R/W)
Vendor Specific DLLP Request
SDIS (R/W)
Scramble Disable
LPBKEN (R/W)
Loopback Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 X X X X 0 0 X X X X X X
LCAPB (R/W)
Link Mode Enable

Figure 29-172: PCIE_PLCTL_[n] Register Diagram

Table 29-181: PCIE_PLCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:16 (R/W)	LCAPB	Link Mode Enable. The `PCIE_PLCTL_[n].LCAPB` bit field sets the number of lanes in the link that are needed to connect to the link partner. When there are unused lanes in a system, then the value in this bit field must change to reflect the number of lanes. The value in the "Predetermined Number of Lanes" field of the "Link Width and Speed Change Control Register" also must change. For more information, see "How to Tie Off Unused Lanes". For information on upsizing and downsizing the link width, see "Link Establishment". This register field is sticky.	
		0	x1
		1	x2
		2	x4
		3	x8
		4	x16
		5	x32 (not supported)

Table 29-181: PCIE_PLCTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	FLM	Fast Link Mode. The `PCIE_PLCTL_[n].FLM` bit enables Fast Link mode. All internal timers are set to Fast Mode to speed up simulation. This mode forces the LTSSM training (link initialization) to use shorter time-outs and to link up faster. The scaling factor is 1024 for all internal timers. Fast Link Mode can also be activated by setting the diag_ctrl_bus[2] pin to 1. This register field is sticky.
5 (R/W)	DLEN	DLL Link Enable. The `PCIE_PLCTL_[n].DLEN` bit enables link initialization. When DLL Link Enable =0, the core does not transmit InitFC DLLPs and does not establish a link. This register field is sticky.
3 (R/W)	RSTA	Reset Assert. The `PCIE_PLCTL_[n].RSTA` bit triggers a recovery and forces the LTSSM to the hot reset state for a downstream port only. This bit is sticky.
2 (R/W)	LPBKEN	Loopback Enable. The `PCIE_PLCTL_[n].LPBKEN` bit turns on loopback. This register field is sticky.
1 (R/W)	SDIS	Scramble Disable. The `PCIE_PLCTL_[n].SDIS` bit turns off data scrambling. This register field is sticky.
0 (R/W)	VDLLP	Vendor Specific DLLP Request. When the `PCIE_PLCTL_[n].VDLLP` bit is set (=1), the core transmits the DLLP contained in the "Vendor Specific DLLP Register". Reading from this self-clearing register field always returns 0.

Port Logic Debug0 Register

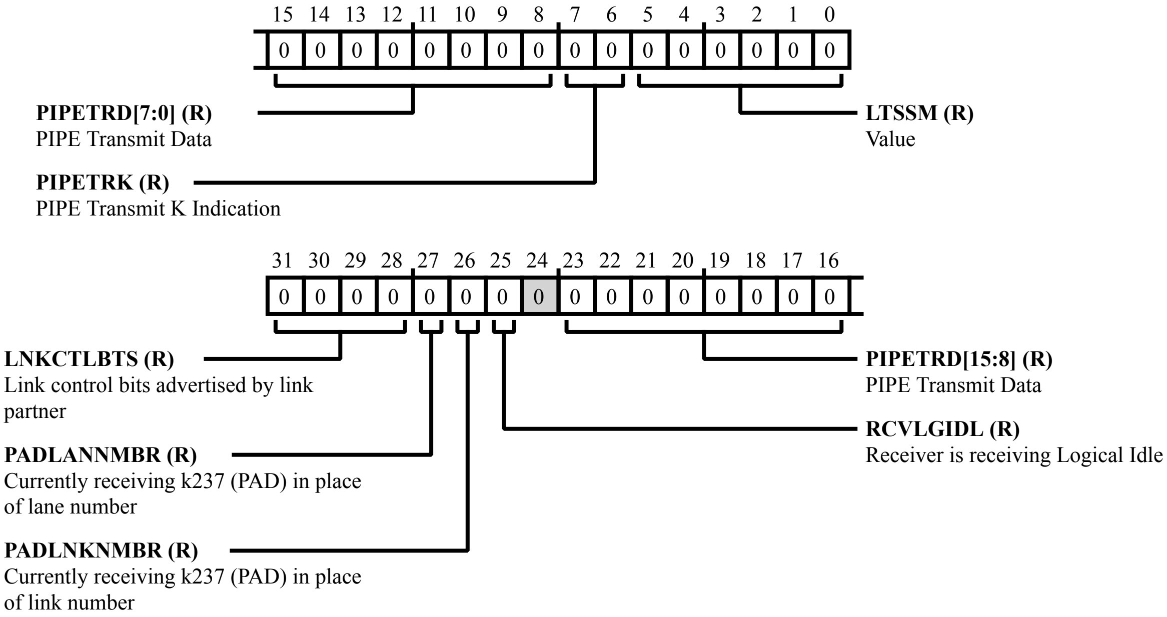

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PIPETRD[7:0] (R)
PIPE Transmit Data
PIPETRK (R)
PIPE Transmit K Indication
LTSSM (R)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LNKCTLBTS (R)
Link control bits advertised by link partner
PADLANNMBR (R)
Currently receiving k237 (PAD) in place of lane number
PADLNKNMBR (R)
Currently receiving k237 (PAD) in place of link number
PIPETRD[15:8] (R)
PIPE Transmit Data
RCVLGIDL (R)
Receiver is receiving Logical Idle

Figure 29-173: PCIE_PLDBG0_[n] Register Diagram

Table 29-182: PCIE_PLDBG0_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	LNKCTLBTS	Link control bits advertised by link partner.
27 (R/NW)	PADLANNMBR	Currently receiving k237 (PAD) in place of lane number.
26 (R/NW)	PADLNKNMBR	Currently receiving k237 (PAD) in place of link number.
25 (R/NW)	RCVLGIDL	Receiver is receiving Logical Idle.
23:8 (R/NW)	PIPETRD	PIPE Transmit Data. PIPETRD[i] is Data Byte [i] transmitted to the PHY over the PIPE interface.
7:6 (R/NW)	PIPETRK	PIPE Transmit K Indication. PIPETRK[i] is the K character indication for PIPETRD Byte [i]
5:0 (R/NW)	LTSSM	Value. Link Training and Status State Machine Current State

Table 29-182: PCIE_PLDBG0_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	S_DETECT_QUIET For more information see the PCI Express Base Specification 3.0
		1	S_DETECT_ACT For more information see the PCI Express Base Specification 3.0
		2	S_POLL_ACTIVE For more information see the PCI Express Base Specification 3.0
		3	S_POLL_COMPLIANCE For more information see the PCI Express Base Specification 3.0
		4	S_POLL_CONFIG For more information see the PCI Express Base Specification 3.0
		5	S_PRE_DETECT_QUIET For more information see the PCI Express Base Specification 3.0
		6	S_DETECT_WAIT For more information see the PCI Express Base Specification 3.0
		7	S_CFG_LINKWD_START For more information see the PCI Express Base Specification 3.0
		8	S_CFG_LINKWD_ACEPT For more information see the PCI Express Base Specification 3.0
		9	S_CFG_LANENUM_WAIT For more information see the PCI Express Base Specification 3.0
		10	S_CFG_LANENUM_ACEPT For more information see the PCI Express Base Specification 3.0
		11	S_CFG_COMPLETE For more information see the PCI Express Base Specification 3.0
		12	S_CFG_IDLE For more information see the PCI Express Base Specification 3.0
		13	S_RCVRY_LOCK For more information see the PCI Express Base Specification 3.0
		14	S_RCVRY_SPEED For more information see the PCI Express Base Specification 3.0
		15	S_RCVRY_RCVRCFG For more information see the PCI Express Base Specification 3.0
		16	S_RCVRY_IDLE For more information see the PCI Express Base Specification 3.0
		17	S_L0 For more information see the PCI Express Base Specification 3.0

Table 29-182: PCIE_PLDBG0_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		18	S_L0S For more information see the PCI Express Base Specification 3.0
		19	S_L123_SEND_EIDLE For more information see the PCI Express Base Specification 3.0
		20	S_L1_IDLE For more information see the PCI Express Base Specification 3.0
		21	S_L2_IDLE For more information see the PCI Express Base Specification 3.0
		22	S_L2_WAKE For more information see the PCI Express Base Specification 3.0
		23	S_DISABLED_ENTRY For more information see the PCI Express Base Specification 3.0
		24	S_DISABLED_IDLE For more information see the PCI Express Base Specification 3.0
		25	S_DISABLED For more information see the PCI Express Base Specification 3.0
		26	S_LPBK_ENTRY For more information see the PCI Express Base Specification 3.0
		27	S_LPBK_ACTIVE For more information see the PCI Express Base Specification 3.0
		28	S_LPBK_EXIT For more information see the PCI Express Base Specification 3.0
		29	S_LPBK_EXIT_TIMEOUT For more information see the PCI Express Base Specification 3.0
		30	S_HOT_RESET_ENTRY For more information see the PCI Express Base Specification 3.0
		31	S_HOT_RESET For more information see the PCI Express Base Specification 3.0
		32	S_RCVRY_EQ0 For more information see the PCI Express Base Specification 3.0
		33	S_RCVRY_EQ1 For more information see the PCI Express Base Specification 3.0
		34	S_RCVRY_EQ2 For more information see the PCI Express Base Specification 3.0
		35	S_RCVRY_EQ3 For more information see the PCI Express Base Specification 3.0

Port Logic Debug1 Register

The `PCIE_PLDBG1_[n]` register provides information regarding the state of the PCIe link.

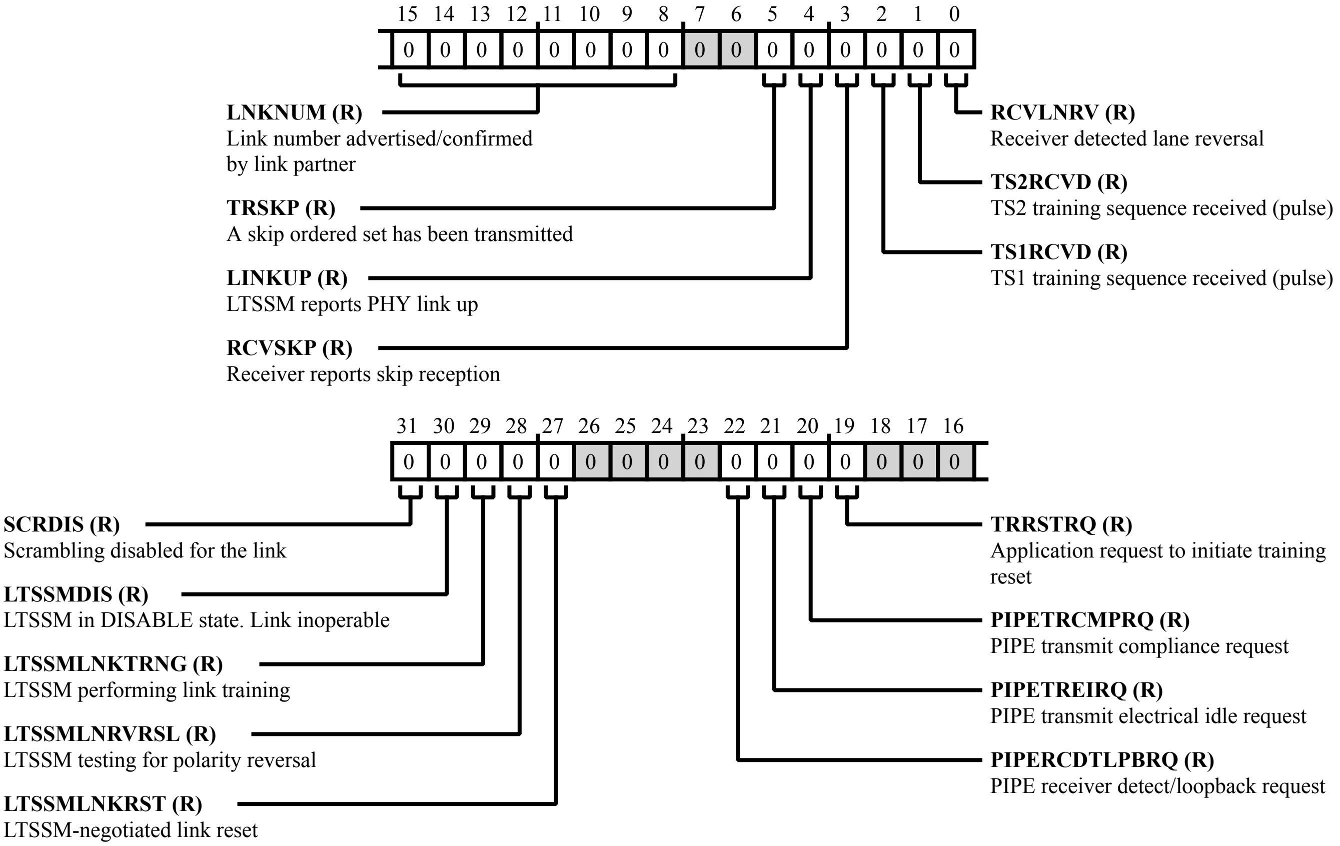

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LNKNUM (R)
Link number advertised/confirmed by link partner
TRSKP (R)
A skip ordered set has been transmitted
LINKUP (R)
LTSSM reports PHY link up
RCVSKP (R)
Receiver reports skip reception
RCVLNRV (R)
Receiver detected lane reversal
TS2RCVD (R)
TS2 training sequence received (pulse)
TS1RCVD (R)
TS1 training sequence received (pulse)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCRDIS (R)
Scrambling disabled for the link
LTSSMDIS (R)
LTSSM in DISABLE state. Link inoperable
LTSSMLNKTRNG (R)
LTSSM performing link training
LTSSMLNRVRSL (R)
LTSSM testing for polarity reversal
LTSSMLNKRST (R)
LTSSM-negotiated link reset
TRRSTRQ (R)
Application request to initiate training reset
PIPETRCMPRQ (R)
PIPE transmit compliance request
PIPETREIRQ (R)
PIPE transmit electrical idle request
PIPERCDTLPBRQ (R)
PIPE receiver detect/loopback request

Figure 29-174: PCIE_PLDBG1_[n] Register Diagram

Table 29-183: PCIE_PLDBG1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	SCRDIS	Scrambling disabled for the link.
30 (R/NW)	LTSSMDIS	LTSSM in DISABLE state. Link inoperable.
29 (R/NW)	LTSSMLNKTRNG	LTSSM performing link training.
28 (R/NW)	LTSSMLNRVRSL	LTSSM testing for polarity reversal.

Table 29-183: PCIE_PLDBG1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
27 (R/NW)	LTSSMLNKRST	LTSSM-negotiated link reset.
22 (R/NW)	PIPERCDTLPBRQ	PIPE receiver detect/loopback request.
21 (R/NW)	PIPETREIRQ	PIPE transmit electrical idle request.
20 (R/NW)	PIPETRCMPRQ	PIPE transmit compliance request.
19 (R/NW)	TRRSTRQ	Application request to initiate training reset.
15:8 (R/NW)	LNKNUM	Link number advertised/confirmed by link partner.
5 (R/NW)	TRSKP	A skip ordered set has been transmitted.
4 (R/NW)	LINKUP	LTSSM reports PHY link up.
3 (R/NW)	RCVSKP	Receiver reports skip reception.
2 (R/NW)	TS1RCVD	TS1 training sequence received (pulse).
1 (R/NW)	TS2RCVD	TS2 training sequence received (pulse).
0 (R/NW)	RCVLNRV	Receiver detected lane reversal.

Port Force Link Register

The `PCIE_PL_FRC_[n]` register implements core support for testing and debug that allows software to force the LTSSM state machine into a specific state, and to force the core to transmit a specific Link Command.

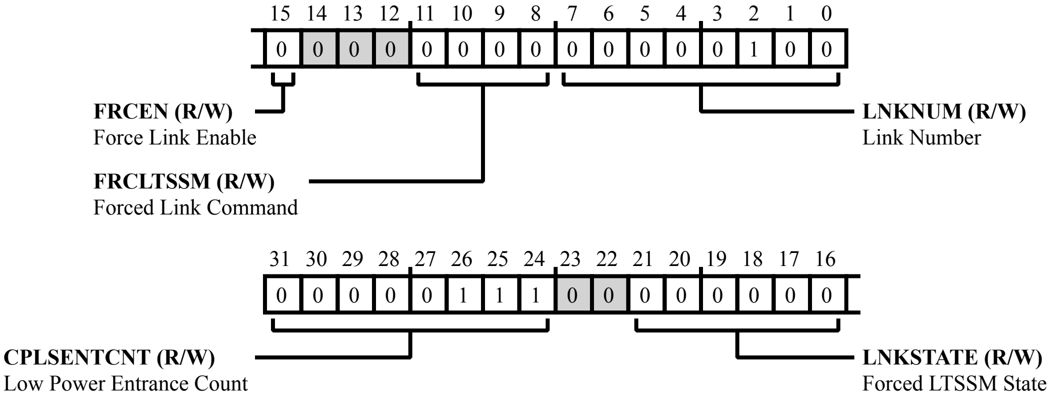

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
FRCEN (R/W)
Force Link Enable
FRCLTSSM (R/W)
Forced Link Command
LNKNUM (R/W)
Link Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
CPLSENTCNT (R/W)
Low Power Entrance Count
LNKSTATE (R/W)
Forced LTSSM State

Figure 29-175: PCIE_PL_FRC_[n] Register Diagram

Table 29-184: PCIE_PL_FRC_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	CPLSENTCNT	Low Power Entrance Count. The `PCIE_PL_FRC_[n].CPLSENTCNT` bit field reports the number of clock cycles for the associated completion of a CfgWr to D-state register to go low-power. This bit field is intended for applications that do not let the core handle a completion for configuration request to the PMCSCR register. Not used in downstream ports. <br><i>Note</i>: This register field is sticky.
21:16 (R/W)	LNKSTATE	Forced LTSSM State. The `PCIE_PL_FRC_[n].LNKSTATE` bit field reports the LTSSM state that the core is forced to when the `PCIE_PL_FRC_[n].FRCEN` bit is set. LTSSM state encoding is defined by the lts_state variable in workspace/src/Layer1/xmlh_ltssm.v. This register field is sticky.
15 (R/W)	FRCEN	Force Link Enable. The `PCIE_PL_FRC_[n].FRCEN` bit enables the force link functionality where the LTSSM is in the state specified by the `PCIE_PL_FRC_[n].LNKSTATE` bit field. Also the core is forced to transmit the command specified by the Forced Link Command field. This is a self-clearing bit. Reading from this bit always returns 0.
11:8 (R/W)	FRCLTSSM	Forced Link Command. The `PCIE_PL_FRC_[n].FRCLTSSM` bit field reports the link command that the core is forced to transmit when the `PCIE_PL_FRC_[n].FRCEN` bit is set. Link command encoding is defined by the ltssm_cmd variable in workspace/src/Layer1/xmlh_ltssm.v. This register field is sticky.

Table 29-184: PCIE_PL_FRC_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	LNKNUM	Link Number. The `PCIE_PL_FRC_[n].LNKNUM` bit field reports the link number. This bit field is not used for endpoint.

Port Control PHY Control Register

The `PCIE_PL_PHYCTL_[n]` register is a memory-mapped register that reflects the values on the the static phy_cfg_control output signals. Using this register does not in any way influence core function. The `PCIE_PL_PHYCTL_[n]` register can be used for any static sideband status signaling requirements for the PHY.

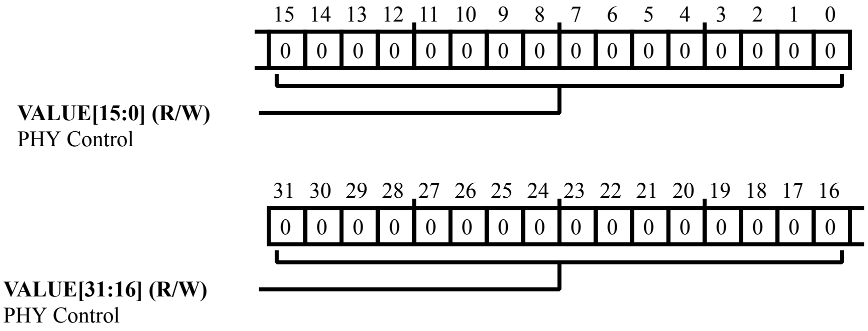

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
PHY Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
PHY Control

Figure 29-176: PCIE_PL_PHYCTL_[n] Register Diagram

Table 29-185: PCIE_PL_PHYCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	PHY Control. The `PCIE_PL_PHYCTL_[n].VALUE` bit field holds the data sent directly from the phy_cfg_control bus.

Port Control PHY Status Register

The `PCIE_PL_PHYSTAT_[n]` register is a memory-mapped register that reflects the values on the the static phy_cfg_status input signals. Using this register does not in any way influence core functionality. The `PCIE_PL_PHYSTAT_[n]` register can be used for any static sideband status signaling requirements for the PHY.

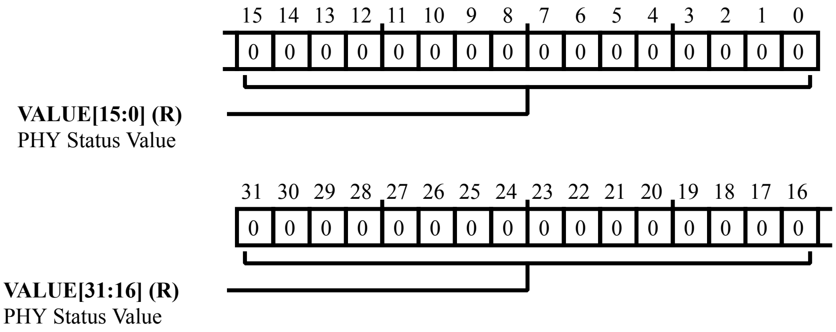

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
PHY Status Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
PHY Status Value

Figure 29-177: PCIE_PL_PHYSTAT_[n] Register Diagram

Table 29-186: PCIE_PL_PHYSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	PHY Status Value. The `PCIE_PL_PHYSTAT_[n].VALUE` bit field holds the data received directly from the phy_cfg_status bus.

Power Management Capabilities Register

The `PCIE_PMT_CAPB_[n]` register indicates various settings for the power management module.

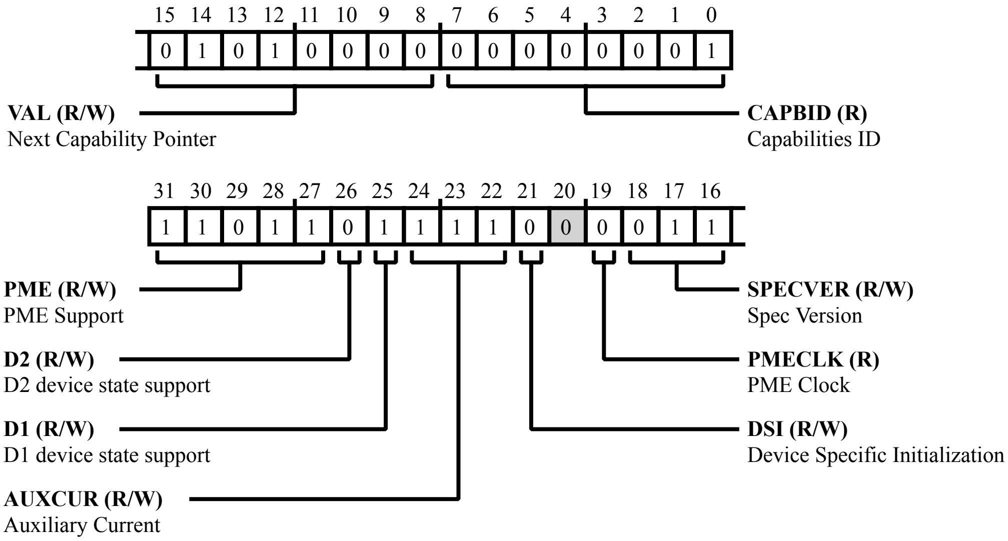

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1
VAL (R/W)
Next Capability Pointer
CAPBID (R)
Capabilities ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 1
PME (R/W)
PME Support
D2 (R/W)
D2 device state support
D1 (R/W)
D1 device state support
AUXCUR (R/W)
Auxiliary Current
SPECVER (R/W)
Spec Version
PMECLK (R)
PME Clock
DSI (R/W)
Device Specific Initialization

Figure 29-178: PCIE_PMT_CAPB_[n] Register Diagram

Table 29-187: PCIE_PMT_CAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:27 (R/W)	PME	PME Support. For a device Function, the `PCIE_PMT_CAPB_[n].PME` bit field indicates the power states in which the Function may generate a PME. Bits 31, 30, and 27 must be set for PCI-PCI Bridge structures representing ports on Root Complexes/Switches to indicate that the Bridge will forward PME Messages.
26 (R/W)	D2	D2 device state support. If the `PCIE_PMT_CAPB_[n].D2` bit =1, this function supports the D2 Power Management State. Functions that do not support D2 must always return a value of 0.
25 (R/W)	D1	D1 device state support. If the `PCIE_PMT_CAPB_[n].D1` bit =1, this function supports the D1 Power Management State. Functions that do not support D1 must always return a value of 0.
24:22 (R/W)	AUXCUR	Auxiliary Current. The `PCIE_PMT_CAPB_[n].AUXCUR` bit field reports the 3.3V auxiliary current requirements for the PCI function.
21 (R/W)	DSI	Device Specific Initialization. The `PCIE_PMT_CAPB_[n].DSI` bit indicates whether special initialization of this function is required (beyond the standard PCI configuration header) before the generic class device driver is able to use it

Table 29-187: PCIE_PMT_CAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/NW)	PMECLK	PME Clock. Does not apply to PCI Express. Must be set to 0
18:16 (R/W)	SPECVER	Spec Version. When the `PCIE_PMT_CAPB_[n].SPECVER` bit field = 011 it indicates that this function complies with revision 1.2 of the PCI Power Management Interface Specification.
15:8 (R/W)	VAL	Next Capability Pointer. The `PCIE_PMT_CAPB_[n].VAL` bit field provides an offset into the functions PCI Configuration Space pointing to the location of next item in the functions capability list. If there are no additional items in the Capabilities List, this bit field is set to 0.
7:0 (R/NW)	CAPBID	Capabilities ID. When the `PCIE_PMT_CAPB_[n].CAPBID` bit field = 0x1, it identifies the linked list item as being the PCI Power Management registers.

Power Management Control and Status Register

The `PCIE_PMT_CTLSTAT_[n]` register controls/indicates various settings for the power management module.

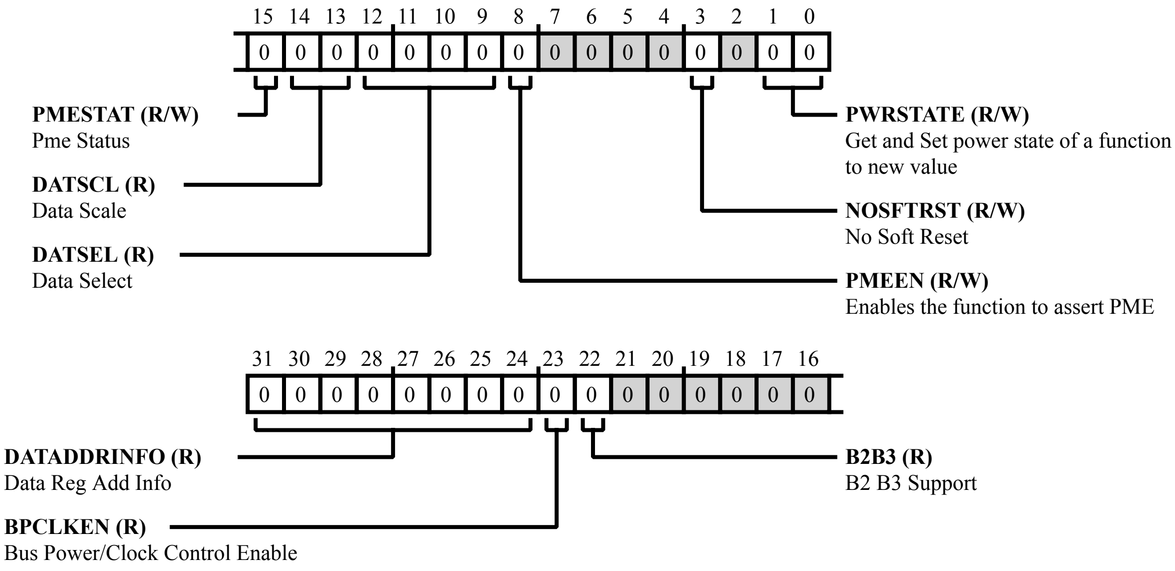

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PMESTAT (R/W)
Pme Status
DATSCL (R)
Data Scale
DATSEL (R)
Data Select
PWRSTATE (R/W)
Get and Set power state of a function to new value
NOSFTRST (R/W)
No Soft Reset
PMEEN (R/W)
Enables the function to assert PME
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATADDRINFO (R)
Data Reg Add Info
BPCLKEN (R)
Bus Power/Clock Control Enable
B2B3 (R)
B2 B3 Support

Figure 29-179: PCIE_PMT_CTLSTAT_[n] Register Diagram

Table 29-188: PCIE_PMT_CTLSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/NW)	DATADDRINFO	Data Reg Add Info. The `PCIE_PMT_CTLSTAT_[n].DATADDRINFO` bit field provides a mechanism for a function to report state dependent operating data such as power consumed or heat dissipation (read-only).	
23 (R/NW)	BPCLKEN	Bus Power/Clock Control Enable. The `PCIE_PMT_CTLSTAT_[n].BPCLKEN` bit determines whether the bridges PMCSR PowerState field can be used by system software to control the power or clock of the bridges secondary bus.	
		0	Indicates that the bus power/clock control policies have been disabled
		1	Indicates that the bus power/clock control mechanism is enabled

Table 29-188: PCIE_PMT_CTLSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/NW)	B2B3	B2 B3 Support. The state of the `PCIE_PMT_CTLSTAT_[n].B2B3` bit determines the action that is to occur as a direct result of programming the function to D3hot. This bit is only meaningful if the `PCIE_PMT_CTLSTAT_[n].BPCLKEN` bit is enabled.	
		0	Indicates that when the bridge function is programmed to D3hot, its secondary bus will have its power removed (B3)
		1	Indicates that when the bridge function is programmed to D3hot, its secondary buss PCI clock will be stopped (B2)
15 (R/W)	PMESTAT	Pme Status. The `PCIE_PMT_CTLSTAT_[n].PMESTAT` bit is set when the function would normally assert the PME# signal independent of the state of the PME_En bit. Writing a 1 to this bit clears it and causes the function to stop asserting a PME# (if enabled). Writing a 0 has no effect. This bit defaults to 0 if the function does not support PME# generation from D3cold. If the function supports PME# from D3cold, then this bit is sticky and must be explicitly cleared by the operating system each time the operating system is initially loaded.	
14:13 (R/NW)	DATSCL	Data Scale. The `PCIE_PMT_CTLSTAT_[n].DATSCL` bit field indicates the scaling factor to be used when interpreting the value of the Data register. The value and meaning of this field varies depending on which data value has been selected by the Data_Select field.	
12:9 (R/NW)	DATSEL	Data Select. The `PCIE_PMT_CTLSTAT_[n].DATSEL` bit field is used to select which data is to be reported through the Data register and Data_Scale field.	
8 (R/W)	PMEEN	Enables the function to assert PME. The `PCIE_PMT_CTLSTAT_[n].PMEEN` bit enables the function to assert PME.	
3 (R/W)	NOSFTRST	No Soft Reset. The `PCIE_PMT_CTLSTAT_[n].NOSFTRST` bit enables or disables No_soft_Reset.	
		0	Functional context is not required to be maintained by the Function in the D3hot state
		1	Functional context should be maintained by the Function in the D3hot state

Table 29-188: PCIE_PMT_CTLSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	PWRSTATE	Get and Set power state of a function to new value. The `PCIE_PMT_CTLSTAT_[n].PWRSTATE` bit field is used both to determine the current power state of a function and to set the function into a new power state. If software attempts to write an unsupported, optional state to this field, the write operation must complete normally on the bus. However, the data is discarded and no state change occurs.	
		0	D0
		1	D1
		2	D2
		3	D3 hot

Queue Status Register

The `PCIE_QSTAT_[n]` register reports the status of data transfers and the FC Latency Timer.

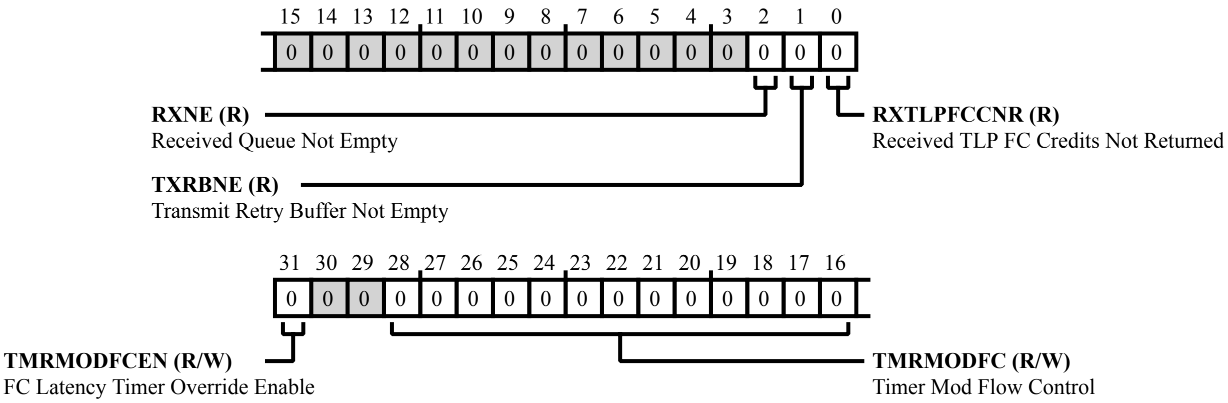

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNE (R)
Received Queue Not Empty
TXRBNE (R)
Transmit Retry Buffer Not Empty
RXTLPFCCNR (R)
Received TLP FC Credits Not Returned
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRMODFCEN (R/W)
FC Latency Timer Override Enable
TMRMODFC (R/W)
Timer Mod Flow Control

Figure 29-180: PCIE_QSTAT_[n] Register Diagram

Table 29-189: PCIE_QSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TMRMODFCEN	FC Latency Timer Override Enable. The `PCIE_QSTAT_[n].TMRMODFCEN` bit indicates the FC Latency Timer Override Enable. When this bit is set, the value from the `PCIE_QSTAT_[n].TMRMODFC` bit field overrides the FC latency timer value that the core calculates according to the PCIe specification.
28:16 (R/W)	TMRMODFC	Timer Mod Flow Control. The `PCIE_QSTAT_[n].TMRMODFC` bit field indicates the FC Latency Timer Override value. When the `PCIE_QSTAT_[n].TMRMODFCEN` bit is set, the value in this field overrides the FC latency timer value that the core calculates according to the PCIe specification.
2 (R/NW)	RXNE	Received Queue Not Empty. The `PCIE_QSTAT_[n].RXNE` bit indicates there is data in one or more of the receive buffers.
1 (R/NW)	TXRBNE	Transmit Retry Buffer Not Empty. The `PCIE_QSTAT_[n].TXRBNE` bit indicates that there is data in the transmit retry buffer.
0 (R/NW)	RXTLPFCCNR	Received TLP FC Credits Not Returned. The `PCIE_QSTAT_[n].RXTLPFCCNR` bit indicates that the core has sent a TLP but has not yet received an UpdateFC DLLP indicating that the credits for that TLP have been restored by the receiver at the other end of the link.

Interrupt Bridge Line and Pin Control Register

Interrupt Bridge, Line and Pin Control Register

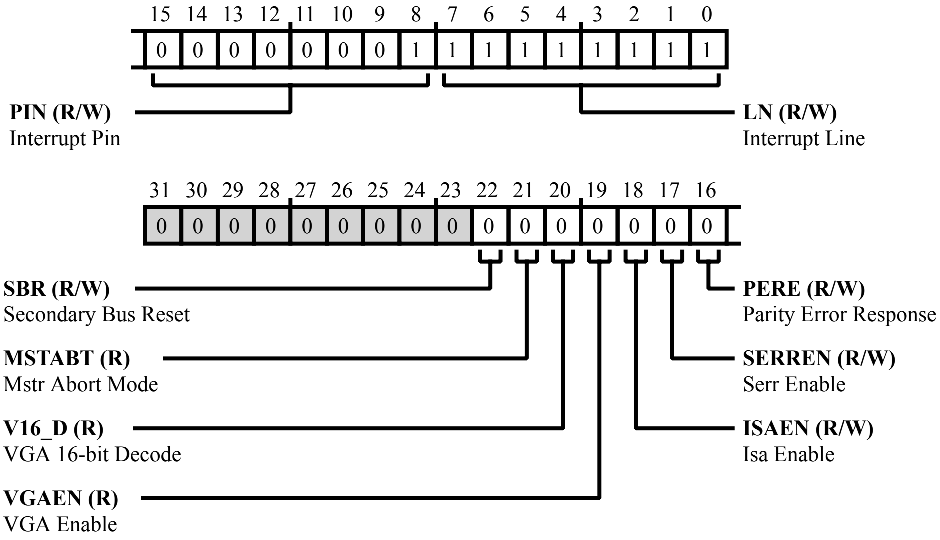

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
PIN (R/W)
Interrupt Pin
LN (R/W)
Interrupt Line
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SBR (R/W)
Secondary Bus Reset
MSTABT (R)
Mstr Abort Mode
V16_D (R)
VGA 16-bit Decode
VGAEN (R)
VGA Enable
PERE (R/W)
Parity Error Response
SERREN (R/W)
Serr Enable
ISAEN (R/W)
Isa Enable

Figure 29-181: PCIE_RC_BRDG_ILPCTL_[n] Register Diagram

Table 29-190: PCIE_RC_BRDG_ILPCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	SBR	Secondary Bus Reset. Setting the `PCIE_RC_BRDG_ILPCTL_[n].SBR` bit triggers a hot reset on the corresponding PCIe port.
21 (R/NW)	MSTABT	Mstr Abort Mode. The `PCIE_RC_BRDG_ILPCTL_[n].MSTABT` bit controls the behavior of a bridge when a Master-Abort termination occurs on either interface while the bridge is the master of the transaction. Does not apply to PCIe and must be set to 0.
20 (R/NW)	V16_D	VGA 16-bit Decode. The `PCIE_RC_BRDG_ILPCTL_[n].V16_D` bit only has meaning if either bit 3 (VGA Enable) of this register or bit 5 (VGA Palette Snoop Enable) of the Command Register, is also set to 1, enabling VGA I/O decoding and forwarding by the bridge. This read/write bit enables system configuration software to select between 10-bit (=0) and 16-bit (=1) I/O address decoding for all VGA I/O register accesses that are forwarded from primary to secondary.

Table 29-190: PCIE_RC_BRDG_ILPCTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/NW)	VGAEN	VGA Enable. The `PCIE_RC_BRDG_ILPCTL_[n].VGAEN` bit modifies the response by the bridge to VGA compatible addresses.	
		0	Do not forward VGA compatible memory and I/O addresses from the primary to the secondary interface (addresses defined above) unless they are enabled for forwarding by the defined I/O and memory address ranges
		1	Forward VGA compatible memory and I/O addresses (addresses defined above) from the primary interface to the secondary interface (if the I/O Enable and Memory Enable bits are set) independent of the I/O and memory address ranges and independent of the ISA Enable bit
18 (R/W)	ISAEN	Isa Enable. The `PCIE_RC_BRDG_ILPCTL_[n].ISAEN` bit modifies the response by the bridge to ISA I/O addresses. This applies only to I/O addresses that are enabled by the I/O Base and I/O Limit registers and are in the first 64 KB of PCI I/O address space (0000 0000h to 0000 FFFFh). The default state of this bit after reset must be 0.	
		0	Forward downstream all I/O addresses in the address range defined by the I/O Base and I/O Limit registers
		1	Forward upstream ISA I/O addresses in the address range defined by the I/O Base and I/O Limit registers that are in the first 64 KB of PCI I/O address space (top 768 bytes of each 1-KB block)
17 (R/W)	SERREN	Serr Enable. The `PCIE_RC_BRDG_ILPCTL_[n].SERREN` bit controls forwarding of correctable, nonfatal and fatal error from secondary to primary.	
16 (R/W)	PERE	Parity Error Response. The `PCIE_RC_BRDG_ILPCTL_[n].PERE` bit controls the logging of poisoned TLPs in the Master Data Parity Error bit in the Secondary Status register.	
15:8 (R/W)	PIN	Interrupt Pin. The `PCIE_RC_BRDG_ILPCTL_[n].PIN` bit field identifies the legacy interrupt Message(s) the Function uses. Valid values are 0x1, 0x2, 0x3, and 0x4 that map to legacy interrupt Messages for INTA, INTB, INTC, and INTD respectively. A value of 0x0 indicates that the Function uses no legacy interrupt Message(s).	

Table 29-190: PCIE_RC_BRDG_ILPCTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	LN	Interrupt Line. The `PCIE_RC_BRDG_ILPCTL_[n].LN` bit field communicates interrupt line routing information. This bit field is read/write and must be implemented by any Function that uses an interrupt pin. Values in this register are programmed by system software and are system architecture specific. The Function itself does not use this value; rather the value in this register is used by device drivers and operating systems

Capability Pointer Register.

The PCIE_RC_CAPBPTR_[n] register points to the first item in the list of capabilities.

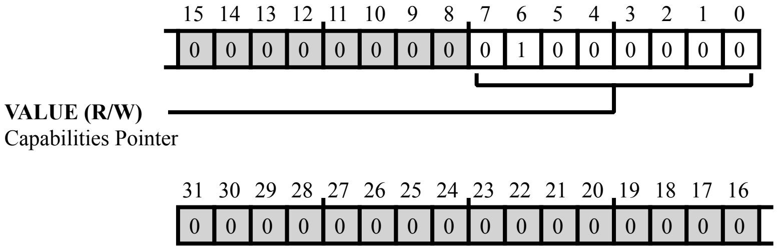

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
VALUE (R/W)
Capabilities Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-182: PCIE_RC_CAPBPTR_[n] Register Diagram

Table 29-191: PCIE_RC_CAPBPTR_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Capabilities Pointer. The `PCIE_RC_CAPBPTR_[n].VALUE` bit field points to the first item in the list of capabilities. Each capability in the list consists of an 8-bit ID field assigned by the PCI SIG, an 8 bit pointer in configuration space to the next capability, and some number of additional registers immediately following the pointer to implement that capability. Each capability must be DWORD aligned. The bottom two bits of all pointers (including the initial pointer at 0x34) are reserved and must be implemented as 00b although software must mask them to allow for future uses of these bits. A pointer value of 0x00 is used to indicate the last capability in the list

Class Code and Revision ID Register

The `PCIE_RC_CCRID_[n]` register provides the Class Code and Revision ID.

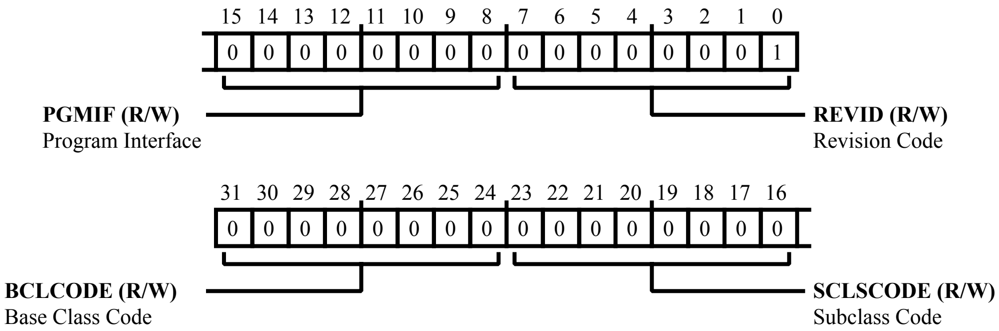

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
PGMIF (R/W)
Program Interface
REVID (R/W)
Revision Code
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BCLCODE (R/W)
Base Class Code
SCLSCODE (R/W)
Subclass Code

Figure 29-183: PCIE_RC_CCRID_[n] Register Diagram

Table 29-192: PCIE_RC_CCRID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BCLCODE	Base Class Code. The `PCIE_RC_CCRID_[n].BCLCODE` bit field broadly classifies the type of function the device performs.
23:16 (R/W)	SCLSCODE	Subclass Code. The `PCIE_RC_CCRID_[n].SCLSCODE` bit field identifies more specifically the function of the device.
15:8 (R/W)	PGMIF	Program Interface. The `PCIE_RC_CCRID_[n].PGMIF` bit field identifies a specific register-level programming interface (if any) so that device independent software can interact with the device.
7:0 (R/W)	REVID	Revision Code. The `PCIE_RC_CCRID_[n].REVID` bit field is the device-specific revision identifier.

Root Complex Configuration Register

The `PCIE_RC_CFG_[n]` register provides bits to configure the header type, cache line size, and master latency timer.

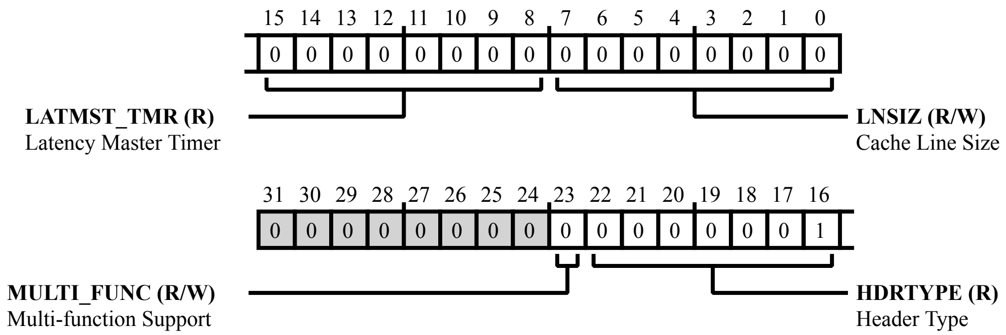

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LATMST_TMR (R)
Latency Master Timer
LNSIZ (R/W)
Cache Line Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
MULTI_FUNC (R/W)
Multi-function Support
HDRTYPE (R)
Header Type

Figure 29-184: PCIE_RC_CFG_[n] Register Diagram

Table 29-193: PCIE_RC_CFG_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	MULTI_FUNC	Multi-function Support. This bit must be set to 0.
22:16 (R/NW)	HDRTYPE	Header Type. The `PCIE_RC_CFG_[n].HDRTYPE` bit field identifies the layout of the second part of the predefined header and also whether or not the device contains multiple functions. Bit 7 in this register is used to identify a multifunction device. If the bit is 0, then the device is single function. If the bit is 1, then the device has multiple functions. Bits 6 through 0 identify the layout of the second part of the predefined header.
15:8 (R/NW)	LATMST_TMR	Latency Master Timer. The `PCIE_RC_CFG_[n].LATMST_TMR` bit field specifies the value of the latency timer in units of PCI bus clocks.
7:0 (R/W)	LNSIZ	Cache Line Size. The `PCIE_RC_CFG_[n].LNSIZ` bit field specifies the system cache line size in units of DWORDS.

Root Control and Capabilities Register

The `PCIE_RC_CTLCAPB_[n]` register identifies PCI Express Root Port specific control and capabilities.

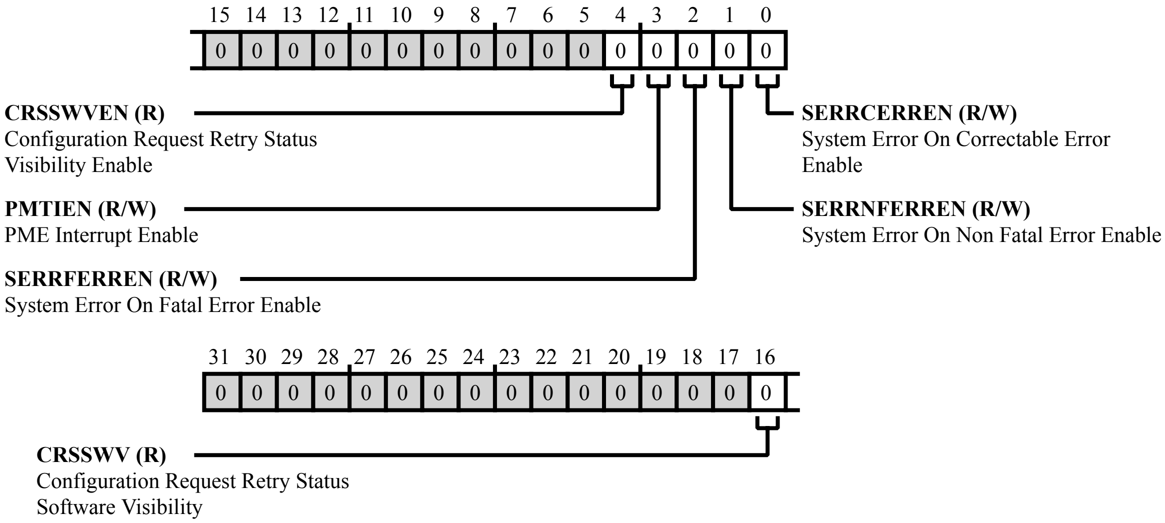

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CRSSWVEN (R)
Configuration Request Retry Status Visibility Enable
PMTIEN (R/W)
PME Interrupt Enable
SERRFERREN (R/W)
System Error On Fatal Error Enable
SERRCERREN (R/W)
System Error On Correctable Error Enable
SERRNFERREN (R/W)
System Error On Non Fatal Error Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CRSSWV (R)
Configuration Request Retry Status Software Visibility

Figure 29-185: PCIE_RC_CTLCAPB_[n] Register Diagram

Table 29-194: PCIE_RC_CTLCAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/NW)	CRSSWV	Configuration Request Retry Status Software Visibility. When Set, this bit indicates that the Root Port is capable of returning Configuration Request Retry Status (CRS) Completion Status to software.
4 (R/NW)	CRSSWVEN	Configuration Request Retry Status Visibility Enable. The `PCIE_RC_CTLCAPB_[n].CRSSWVEN` bit enables the Root Port to return Configuration Request Retry Status (CRS) Completion Status to software. Root Ports that do not implement this capability must set this bit to 0. Default value of this bit is 0.
3 (R/W)	PMTIEN	PME Interrupt Enable. The `PCIE_RC_CTLCAPB_[n].PMTIEN` bit enables PME interrupt generation upon receipt of a PME Message as reflected in the PME Status bit. A PME interrupt is also generated if the PME Status bit is set when this bit is changed from clear to set. Default value of this bit is 0.

Table 29-194: PCIE_RC_CTLCAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	SERRFERREN	System Error On Fatal Error Enable. The `PCIE_RC_CTLCAPB_[n].SERRFERREN` bit indicates that a System Error should be generated if a Fatal error is reported by any of the devices in the hierarchy associated with this Root Port, or by the Root Port itself. The mechanism for signaling a System Error to the system is system specific. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints. Default value of this bit is 0.
1 (R/W)	SERRNFERREN	System Error On Non Fatal Error Enable. The `PCIE_RC_CTLCAPB_[n].SERRNFERREN` bit indicates that a System Error should be generated if a Non-fatal error is reported by any of the devices in the hierarchy associated with this Root Port, or by the Root Port itself. The mechanism for signaling a System Error to the system is system specific. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints. Default value of this bit is 0.
0 (R/W)	SERRCERREN	System Error On Correctable Error Enable. The `PCIE_RC_CTLCAPB_[n].SERRCERREN` bit indicates that a System Error should be generated if a correctable error is reported by any of the devices in the hierarchy associated with this Root Port, or by the Root Port itself. The mechanism for signaling a System Error to the system is system specific. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints. Default value of this bit is 0.

Device Capabilities Register

The `PCIE_RC_DEVCAPB_[n]` register identifies PCI Express device Function specific capabilities.

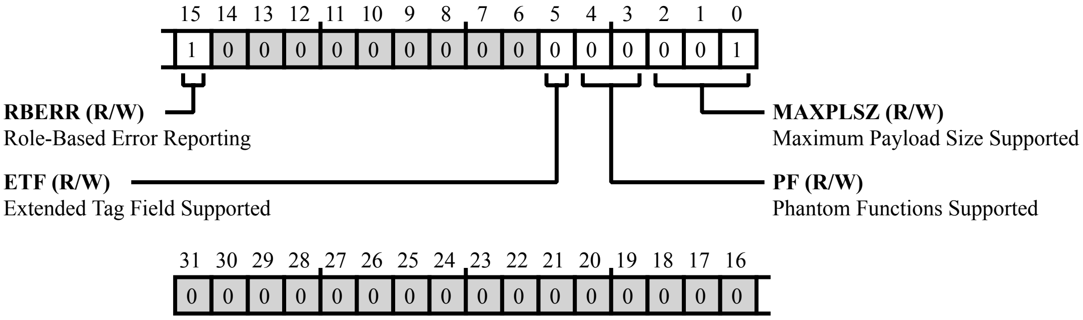

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
RBERR (R/W)
Role-Based Error Reporting
ETF (R/W)
Extended Tag Field Supported
MAXPLSZ (R/W)
Maximum Payload Size Supported
PF (R/W)
Phantom Functions Supported
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-186: PCIE_RC_DEVCAPB_[n] Register Diagram

Table 29-195: PCIE_RC_DEVCAPB_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	RBERR	Role-Based Error Reporting. When the `PCIE_RC_DEVCAPB_[n].RBERR` bit is set, (=1) it indicates that the Function implements the functionality originally defined in the PCI Express Base Specification. This bit is read-only and always set to 1.	
5 (R/W)	ETF	Extended Tag Field Supported. The `PCIE_RC_DEVCAPB_[n].ETF` bit indicates the maximum supported size of the Tag field as a Requester. Note that 8-bit Tag field generation must be enabled by the Extended Tag Field Enable bit in the Device Control register of the Requester Function before 8-bit Tags can be generated by the Requester.	
		0	5-bit Tag field supported
		1	8-bit Tag field supported
4:3 (R/W)	PF	Phantom Functions Supported. The `PCIE_RC_DEVCAPB_[n].PF` bit field indicates the support for use of unclaimed Function Numbers to extend the number of outstanding transactions allowed by logically combining unclaimed Function Numbers (called Phantom Functions) with the Tag identifier.	
2:0 (R/W)	MAXPLSZ	Maximum Payload Size Supported. The `PCIE_RC_DEVCAPB_[n].MAXPLSZ` bit field indicates the maximum payload size that the Function can support for TLPs. The Functions of a multi-Function device are permitted to report different values for this field.	
		0	128 bytes max payload size
		1	256 bytes max payload size
		2	512 bytes max payload size

Table 29-195: PCIE_RC_DEVCAPB_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	1024 bytes max payload size
		4	2048 bytes max payload size
		5	4096 bytes max payload size
		6	Reserved
		7	Reserved

Root Complex Error Command Register

The PCIE_RC_ERRCMD_[n] register allows further control of Root Complex response to Correctable, Non-Fatal, and Fatal error Messages than the basic Root Complex capability to generate system errors in response to error Messages (either received or internally generated). Bit fields enable or disable generation of interrupts (claimed by the Root Port or Root Complex Event Collector) in addition to system error Messages.

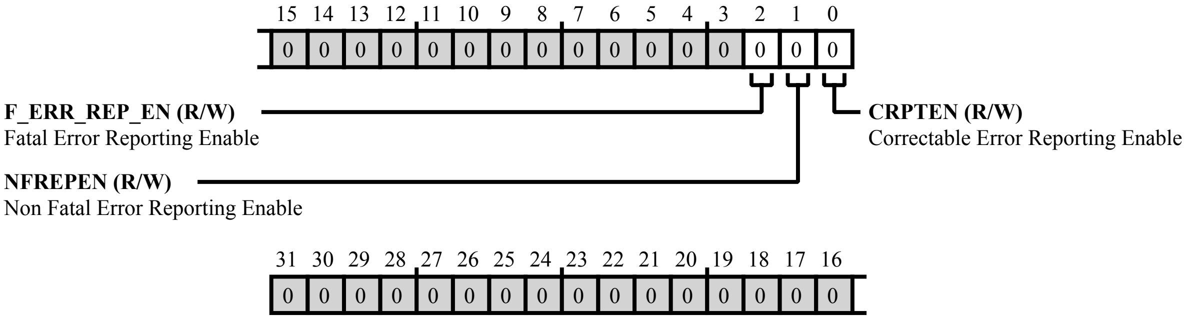

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F_ERR_REP_EN (R/W)
Fatal Error Reporting Enable
NFREPEN (R/W)
Non Fatal Error Reporting Enable
CRPTEN (R/W)
Correctable Error Reporting Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 29-187: PCIE_RC_ERRCMD_[n] Register Diagram

Table 29-196: PCIE_RC_ERRCMD_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	F_ERR_REP_EN	Fatal Error Reporting Enable. The PCIE_RC_ERRCMD_[n].F_ERR_REP_EN bit enables the generation of an interrupt when a Fatal error is reported by any of the Functions in the hierarchy associated with this Root Complex Port. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints.
1 (R/W)	NFREPEN	Non Fatal Error Reporting Enable. The PCIE_RC_ERRCMD_[n].NFREPEN bit enables the generation of an interrupt when a Non-fatal error is reported by any of the Functions in the hierarchy associated with this Root Complex Port. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints.
0 (R/W)	CRPTEN	Correctable Error Reporting Enable. When Set, this bit enables the generation of an interrupt when a correctable error is reported by any of the Functions in the hierarchy associated with this Root Complex Port. Root Complex Event Collectors provide support for the above described functionality for Root Complex Integrated Endpoints.

Root Error Status Register

The `PCIE_RC_ERRSTAT_[n]` register reports status of error Messages (ERR_COR, ERR_NONFATAL, and ERR_FATAL) received by the Root Port, and of errors detected by the Root Port itself (which are treated conceptually as if the Root Port had sent an error Message to itself).

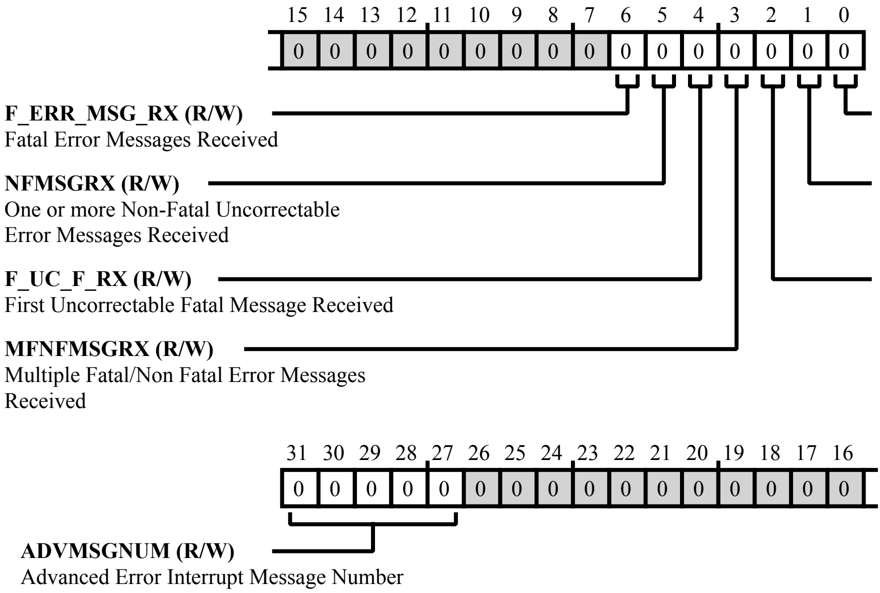

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F_ERR_MSG_RX (R/W)
Fatal Error Messages Received
NFMSGRX (R/W)
One or more Non-Fatal Uncorrectable Error Messages Received
F_UC_F_RX (R/W)
First Uncorrectable Fatal Message Received
MFNFMSGRX (R/W)
Multiple Fatal/Non Fatal Error Messages Received
CORMSGRX (R/W)
Correctable Error Message Received
MCORMSGRX (R/W)
Multiple Correctable Error Messages Received
FNFMSGRX (R/W)
Fatal/Non Fatal Error Received
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADVMSGNUM (R/W)
Advanced Error Interrupt Message Number

Figure 29-188: PCIE_RC_ERRSTAT_[n] Register Diagram

Table 29-197: PCIE_RC_ERRSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:27 (R/W)	ADVMSGNUM	Advanced Error Interrupt Message Number. The `PCIE_RC_ERRSTAT_[n].ADVMSGNUM` bit field indicates which MSI/MSI-X vector is used for the interrupt message generated in association with any of the status bits of this Capability. For MSI, the value in this register indicates the offset between the base Message Data and the interrupt message that is generated. Hardware is required to update this field so that it is correct if the number of MSI Messages assigned to the Function changes when software writes to the Multiple Message Enable field in the MSI Message Control register. For MSI-X, the value in this register indicates which MSI-X Table entry is used to generate the interrupt message. The entry must be one of the first 32 entries even if the Function implements more than 32 entries. For a given MSI-X implementation, the entry must remain constant. If both MSI and MSI-X are implemented, they are permitted to use different vectors, though software is permitted to enable only one mechanism at a time. If MSI-X is enabled, the value in this register must indicate the vector for MSI-X. If MSI is enabled or neither is enabled, the value in this register must indicate the vector for MSI. If software enables both MSI and MSI-X at the same time, the value in this register is undefined.
6 (R/W)	F_ERR_MSG_RX	Fatal Error Messages Received. The `PCIE_RC_ERRSTAT_[n].F_ERR_MSG_RX` bit is set when one or more Fatal Uncorrectable error Messages have been received. Default value of this bit is 0.
5 (R/W)	NFMSGRX	One or more Non-Fatal Uncorrectable Error Messages Received. The `PCIE_RC_ERRSTAT_[n].NFMSGRX` bit is set when one or more Non-Fatal Uncorrectable error Messages have been received. Default value of this bit is 0.
4 (R/W)	F_UC_F_RX	First Uncorrectable Fatal Message Received. The `PCIE_RC_ERRSTAT_[n].F_UC_F_RX` bit is set when the first Uncorrectable error Message received is for a Fatal error. Default value of this field is 0.
3 (R/W)	MFNFMSGRX	Multiple Fatal/Non Fatal Error Messages Received. The `PCIE_RC_ERRSTAT_[n].MFNFMSGRX` bit is set when either a Fatal or a Non-fatal error is received and ERR_FATAL/NONFATAL Received is already Set. Default value of this bit is 0.
2 (R/W)	FNFMSGRX	Fatal/Non Fatal Error Received. The `PCIE_RC_ERRSTAT_[n].FNFMSGRX` bit is set when either a Fatal or a Non-fatal error Message is received and this bit is not already Set. Default value of this bit is 0.

Table 29-197: PCIE_RC_ERRSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	MCORMSGRX	Multiple Correctable Error Messages Received. The `PCIE_RC_ERRSTAT_[n].MCORMSGRX` bit is set when a Correctable error Message is received and ERR_COR Received is already Set. Default value of this bit is 0.
0 (R/W)	CORMSGRX	Correctable Error Message Received. The `PCIE_RC_ERRSTAT_[n].CORMSGRX` bit is set when a Correctable error Message is received and this bit is not already set. Default value of this bit is 0.

Device ID and Vendor ID Register

The `PCIE_RC_ID_[n]` register provides the Device ID and Vendor ID.

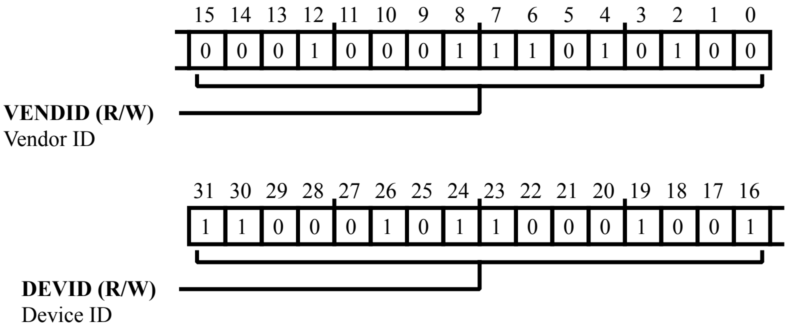

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0
VENDID (R/W)
Vendor ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1
DEVID (R/W)
Device ID

Figure 29-189: PCIE_RC_ID_[n] Register Diagram

Table 29-198: PCIE_RC_ID_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DEVID	Device ID. The `PCIE_RC_ID_[n].DEVID` bit field is the Vendor assigned field that provides a means for a vendor to classify a particular RCRB.
15:0 (R/W)	VENDID	Vendor ID. The `PCIE_RC_ID_[n].VENDID` bit field is the PCI-SIG assigned field that provides a means to associate an RCRB (root complex register block) with a particular vendor.

Root Complex I/O Base and Limit Upper 16 bits Register

The `PCIE_RC_IOBL_[n]` register provides the root complex I/O base and limit upper address bits (16 bits).

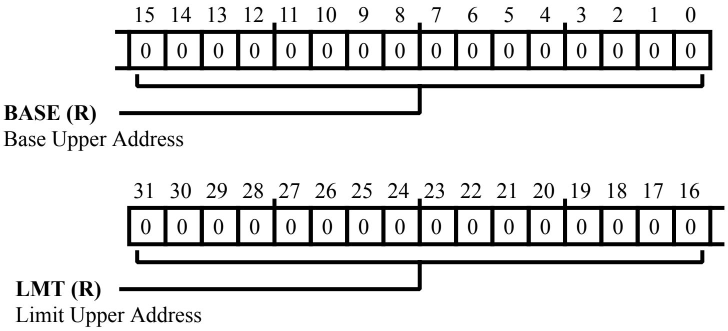

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE (R)
Base Upper Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMT (R)
Limit Upper Address

Figure 29-190: PCIE_RC_IOBL_[n] Register Diagram

Table 29-199: PCIE_RC_IOBL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	LMT	Limit Upper Address. The `PCIE_RC_IOBL_[n].LMT` bit field contains the I/O Limit upper 16 bits.
15:0 (R/NW)	BASE	Base Upper Address. The `PCIE_RC_IOBL_[n].BASE` bit field contains the I/O base upper 16 bits.

Memory Base and Memory Limit Register

The `PCIE_RC_MBL_[n]` register contains the non-prefetchable memory base address and the non-prefetchable memory limit.

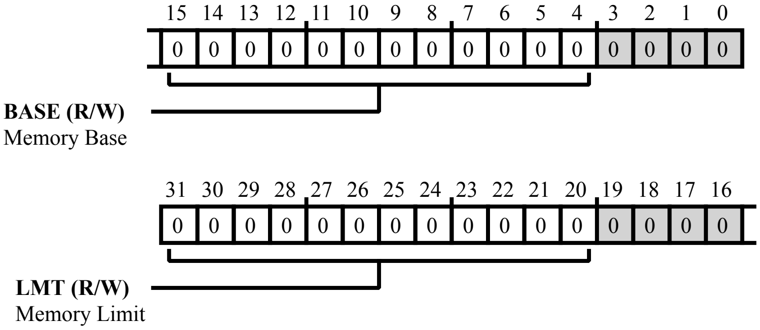

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE (R/W)
Memory Base
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMT (R/W)
Memory Limit

Figure 29-191: PCIE_RC_MBL_[n] Register Diagram

Table 29-200: PCIE_RC_MBL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:20 (R/W)	LMT	Memory Limit. The `PCIE_RC_MBL_[n].LMT` bit field contains the non-prefetchable memory limit.
15:4 (R/W)	BASE	Memory Base. The `PCIE_RC_MBL_[n].BASE` bit field contains the non-prefetchable memory base address.

Prefetchable Memory Base and Limit Register

The `PCIE_RC_PREFMBL_[n]` register contains the prefetchable memory base and limit addresses.

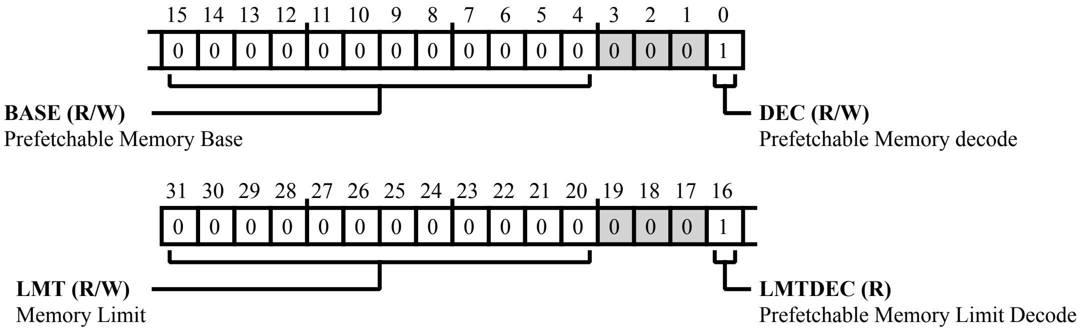

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
BASE (R/W)
Prefetchable Memory Base
DEC (R/W)
Prefetchable Memory decode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
LMT (R/W)
Memory Limit
LMTDEC (R)
Prefetchable Memory Limit Decode

Figure 29-192: PCIE_RC_PREFMBL_[n] Register Diagram

Table 29-201: PCIE_RC_PREFMBL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:20 (R/W)	LMT	Memory Limit. Precethable memory limit
16 (R/NW)	LMTDEC	Prefetchable Memory Limit Decode. Prefetchable memory limit decode
15:4 (R/W)	BASE	Prefetchable Memory Base. Prefetchable memory base address
0 (R/W)	DEC	Prefetchable Memory decode. Prefetchable memory decode

Prefetchable Base Upper 32 Bits Register

The `PCIE_RC_PREF_BUPP_[n]` register contains the Prefetchable base upper 32 bits.

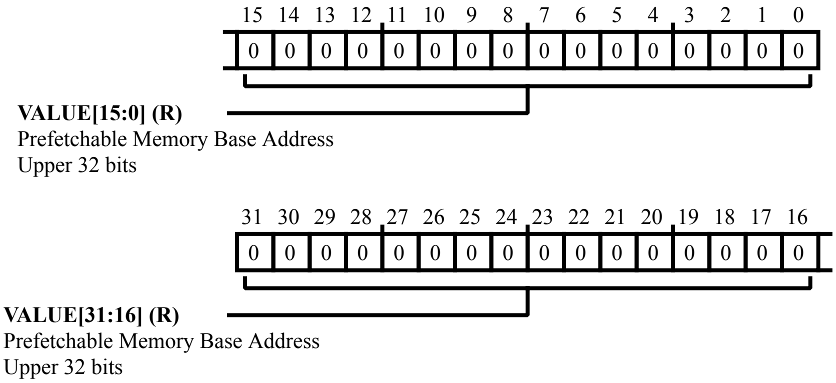

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Prefetchable Memory Base Address Upper 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Prefetchable Memory Base Address Upper 32 bits

Figure 29-193: PCIE_RC_PREF_BUPP_[n] Register Diagram

Table 29-202: PCIE_RC_PREF_BUPP_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Prefetchable Memory Base Address Upper 32 bits. The `PCIE_RC_PREF_BUPP_[n].VALUE` bit field contains the prefetchable base address upper 32 bits.

Prefetchable Limit Upper 32 Bits Register

The PCIE_RC_PREF_LMT_UP_[n] register contains the prefetchable limit upper 32 bits.

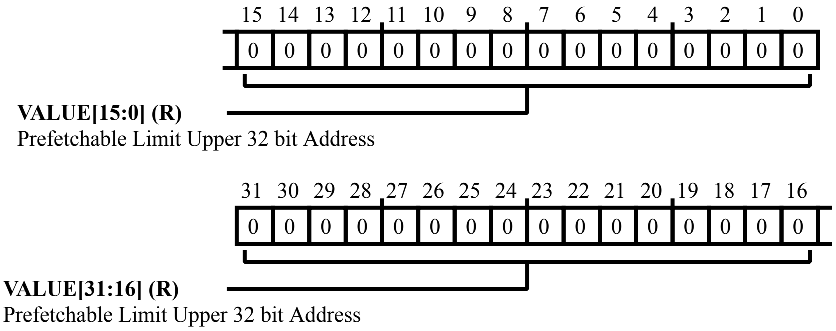

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Prefetchable Limit Upper 32 bit Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Prefetchable Limit Upper 32 bit Address

Figure 29-194: PCIE_RC_PREF_LMT_UP_[n] Register Diagram

Table 29-203: PCIE_RC_PREF_LMT_UP_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Prefetchable Limit Upper 32 bit Address. The `PCIE_RC_PREF_LMT_UP_[n].VALUE` bit field contains the prefetchable limit upper 32 bit address.

Root Complex Expansion ROM Base Address Register

The `PCIE_RC_ROMCFG_[n]` register controls whether or not the device accepts accesses to its expansion ROM and indicates the upper 21 bits of the Expansion ROM base address.

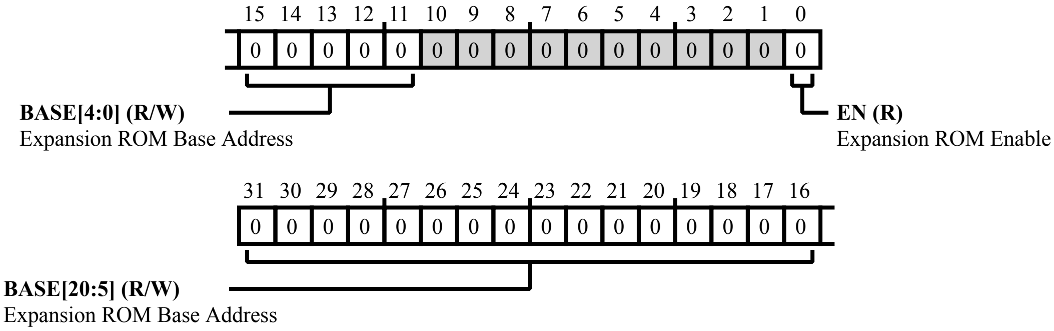

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE[4:0] (R/W)
Expansion ROM Base Address
EN (R)
Expansion ROM Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BASE[20:5] (R/W)
Expansion ROM Base Address

Figure 29-195: PCIE_RC_ROMCFG_[n] Register Diagram

Table 29-204: PCIE_RC_ROMCFG_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:11 (R/W)	BASE	Expansion ROM Base Address. The `PCIE_RC_ROMCFG_[n].BASE` bit field corresponds to the upper 21 bits of the Expansion ROM base address. The number of bits (out of these 21) that a device actually implements depends on how much address space the device requires. For instance, a device that requires a 64 KB area to map its expansion ROM would implement the top 16 bits in the register, leaving the bottom 5 (out of these 21) hardwired to 0. Devices that support an expansion ROM must implement this register
0 (R/NW)	EN	Expansion ROM Enable. The `PCIE_RC_ROMCFG_[n].EN` bit controls whether or not the device accepts accesses to its expansion ROM. When this bit is 0, the devices expansion ROM address space is disabled. When the bit is 1, address decoding is enabled using the parameters in the other part of the base register. This allows a device to be used with or without an expansion ROM depending on system configuration.

Secondary Status and I/O Base and Limit Register

The `PCIE_RC_SECSTAT_[n]` register provides error and status information for the secondary status and I/O base and limit. This register also provides the upper 4 bits of the 16-bit limit address (bits 15:12) and the upper 4 bits of the 16- bit base address (bits 15:12).

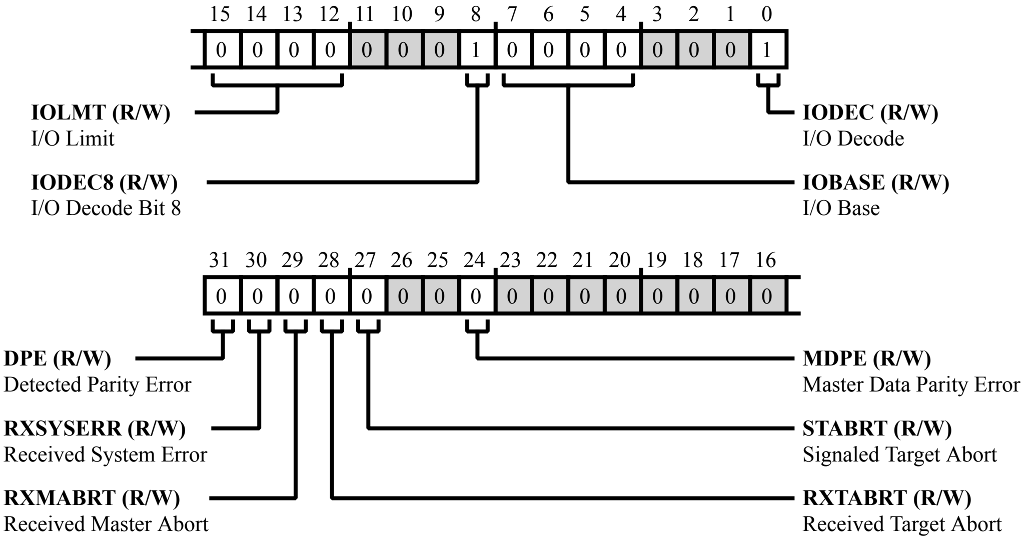

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
IOLMT (R/W)
I/O Limit
IODEC8 (R/W)
I/O Decode Bit 8
IODEC (R/W)
I/O Decode
IOBASE (R/W)
I/O Base
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DPE (R/W)
Detected Parity Error
RXSYSERR (R/W)
Received System Error
RXMABRT (R/W)
Received Master Abort
MDPE (R/W)
Master Data Parity Error
STABRT (R/W)
Signaled Target Abort
RXTABRT (R/W)
Received Target Abort

Figure 29-196: PCIE_RC_SECSTAT_[n] Register Diagram

Table 29-205: PCIE_RC_SECSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	DPE	Detected Parity Error. The `PCIE_RC_SECSTAT_[n].DPE` bit reports the detection of an address or data parity error by the bridge on its secondary interface. This bit must be set when any of the following conditions is true: • Detects an address parity error as a potential target • Detects a data parity error when the target of a write transaction • Detects a data parity error when the master of a read transaction The bit is set regardless of the state of the PERREN bit (bit 0) in the Bridge Control register. Once set, this bit remains set until it is reset by writing a 1 to this bit location. A bridge must implement this bit, and the default state must be 0 after reset.	
		0	Address or data parity error not detected by the bridge on its secondary interface
		1	Address or data parity error detected by the bridge on its secondary interface

Table 29-205: PCIE_RC_SECSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RXSYSERR	Received System Error. The `PCIE_RC_SECSTAT_[n].RXSYSERR` bit reports the detection of an SERR# assertion on the secondary interface of the bridge. Once set, this bit remains set until it is reset by writing a 1 to this bit location. The default state must be 0 after reset.	
		0	SERR# assertion on the secondary interface has not been detected
		1	SERR# assertion on the secondary interface has been detected
29 (R/W)	RXMABRT	Received Master Abort. The `PCIE_RC_SECSTAT_[n].RXMABRT` bit reports the detection of a Master-Abort termination by the bridge when it is the master of a transaction on its secondary interface. Once set, this bit remains set until it is reset by writing a 1 to this bit location. A bridge must implement this bit and the default state must be 0 after reset.	
		0	Master-Abort not detected by the bridge on its secondary interface
		1	Master-Abort detected by the bridge on its secondary interface
28 (R/W)	RXTABRT	Received Target Abort. The `PCIE_RC_SECSTAT_[n].RXTABRT` bit reports the detection of a Target-Abort termination by the bridge when it is the master of a transaction on its secondary interface. Once set, this bit remains set until it is reset by writing a 1 to this bit location. A bridge must implement this bit. The default state of this bit must be 0 after reset.	
		0	Target-Abort not detected by the bridge on its secondary interface
		1	Target-Abort detected by the bridge on its secondary interface
27 (R/W)	STABRT	Signaled Target Abort. The `PCIE_RC_SECSTAT_[n].STABRT` bit reports the signaling of a Target-Abort termination by the bridge when it responds as the target of a transaction on its secondary interface. Once set, this bit remains set until it is reset by writing a 1 to this bit location. A bridge must implement this bit and the default state must be 0 after reset.	
		0	Target-Abort not signaled by the bridge on its secondary interface
		1	Target-Abort signaled by the bridge on its secondary interface

Table 29-205: PCIE_RC_SECSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	MDPE	Master Data Parity Error. The `PCIE_RC_SECSTAT_[n].MDPE` bit is used to report the detection of a parity error by the bridge when it is the master of a transaction. This bit is set if all the following conditions are true: • The bridge is the bus master of the transaction on the secondary interface. • The bridge asserted PERR# (read transaction) or detected PERR# asserted (write transaction). • The Parity Error Response bit in the Bridge Control register is set. Once set, this bit remains set until it is reset by writing a 1 to this bit location. A bridge must implement this bit and the default state must be 0 after reset.	
		0	No parity error detected on the secondary interface
		1	Parity error detected on the secondary interface
15:12 (R/W)	IOLMT	I/O Limit. The `PCIE_RC_SECSTAT_[n].IOLMT` bit field contains the upper 4 bits of the 16-bit Limit address (bits 15:12). The lower 12 bits of the limit address are implied to be all F's.	
8 (R/W)	IODEC8	I/O Decode Bit 8.	
7:4 (R/W)	IOBASE	I/O Base. The `PCIE_RC_SECSTAT_[n].IOBASE` bit field contains the upper 4 bits of the 16-bit Base address (bits 15:12). The lower 12 bits of the base address are implied to be all 0's, meaning that the base address is always aligned on a 4KB boundary.	
0 (R/W)	IODEC	I/O Decode.	

Command and Status Register

The `PCIE_RC_STATCMD_[n]` register contains bits that enable various memory spaces. The register also reports several errors.

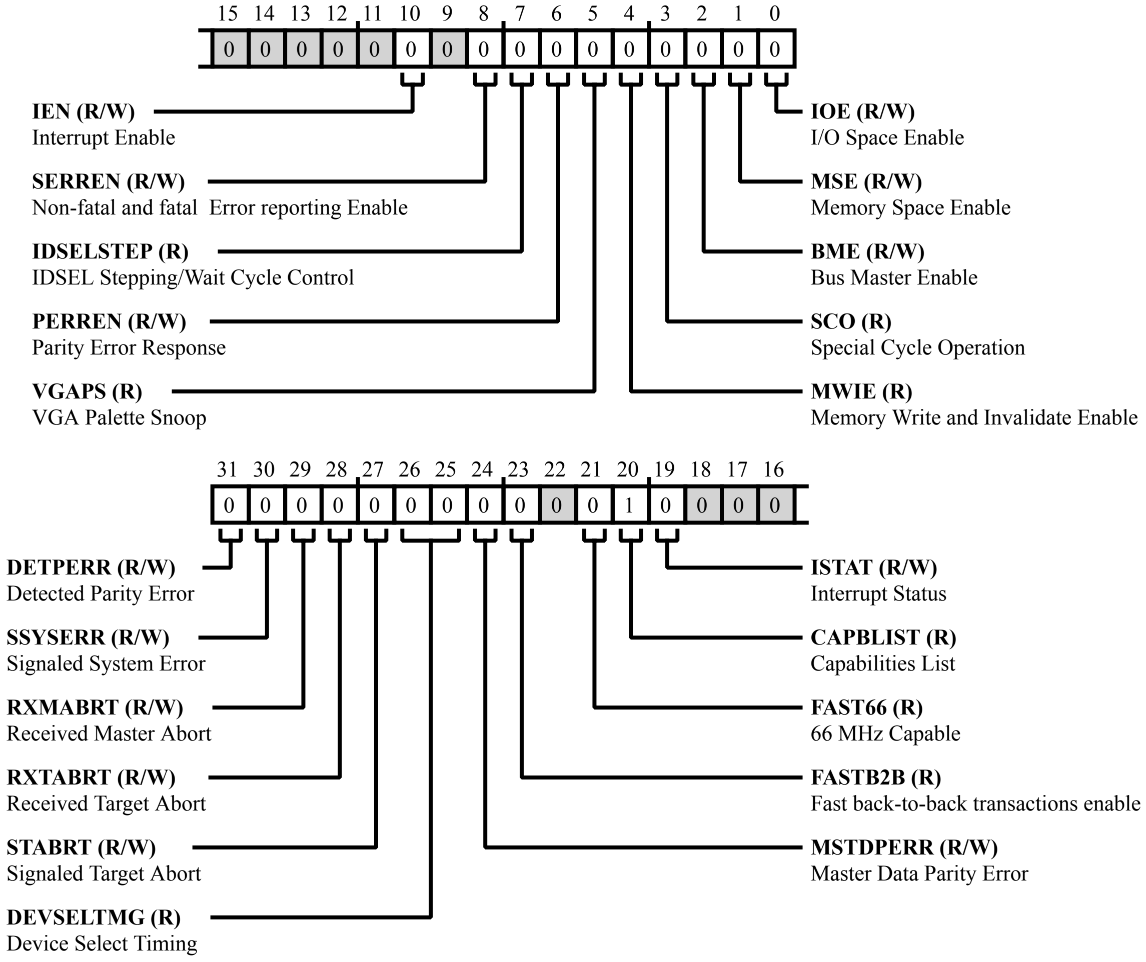

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IEN (R/W)
Interrupt Enable
SERREN (R/W)
Non-fatal and fatal Error reporting Enable
IDSELSTEP (R)
IDSEL Stepping/Wait Cycle Control
PERREN (R/W)
Parity Error Response
VGAPS (R)
VGA Palette Snoop
IOE (R/W)
I/O Space Enable
MSE (R/W)
Memory Space Enable
BME (R/W)
Bus Master Enable
SCO (R)
Special Cycle Operation
MWIE (R)
Memory Write and Invalidate Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
DETPERR (R/W)
Detected Parity Error
SSYSERR (R/W)
Signaled System Error
RXMABRT (R/W)
Received Master Abort
RXTABRT (R/W)
Received Target Abort
STABRT (R/W)
Signaled Target Abort
DEVSELTMG (R)
Device Select Timing
ISTAT (R/W)
Interrupt Status
CAPBLIST (R)
Capabilities List
FAST66 (R)
66 MHz Capable
FASTB2B (R)
Fast back-to-back transactions enable
MSTDPERR (R/W)
Master Data Parity Error

Figure 29-197: PCIE_RC_STATCMD_[n] Register Diagram

Table 29-206: PCIE_RC_STATCMD_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	DETPERR	Detected Parity Error. The `PCIE_RC_STATCMD_[n].DETPERR` bit is set by a Function whenever it receives a Poisoned TLP, regardless of the state of the `PCIE_RC_STATCMD_[n].PERREN` bit. On a Function with a Type 1 Configuration header, the bit is Set when the Poisoned TLP is received by its Primary Side. Default value of this bit is 0.

Table 29-206: PCIE_RC_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W)	SSYSERR	Signaled System Error. The `PCIE_RC_STATCMD_[n].SSYSERR` bit is Set when a Function sends an ERR_FATAL or ERR_NONFATAL Message, and the `PCIE_RC_STATCMD_[n].SERREN` bit is set to 1. Default value of this bit is 0.
29 (R/W)	RXMABRT	Received Master Abort. The `PCIE_RC_STATCMD_[n].RXMABRT` bit is set when a Requester receives a Completion with Unsupported Request Completion Status. On a Function with a Type 1 Configuration header, the bit is Set when the Unsupported Request is received by its Primary Side. Default value of this bit is 0.
28 (R/W)	RXTABRT	Received Target Abort. The `PCIE_RC_STATCMD_[n].RXTABRT` bit is Set when a Requester receives a Completion with Completer Abort Completion Status. On a Function with a Type 1 Configuration header, the bit is Set when the Completer Abort is received by its Primary Side. Default value of this bit is 0.
27 (R/W)	STABRT	Signaled Target Abort. The `PCIE_RC_STATCMD_[n].STABRT` bit is set when a Function completes a Posted or Non-Posted Request as a Completer Abort error. This applies to a Function with a Type 1 Configuration header when the Completer Abort was generated by its Primary Side. Default value of this bit is 0.
26:25 (R/NW)	DEVSELTMG	Device Select Timing. Does not apply to PCI express and must be set to 0.
24 (R/W)	MSTDPERR	Master Data Parity Error. The `PCIE_RC_STATCMD_[n].MSTDPERR` bit is set by a Root Port, Switch Upstream Port, or Switch Downstream Port if the `PCIE_RC_STATCMD_[n].PERREN` bit is set to 1 and either of the following two conditions occurs: • Port receives a Poisoned Completion going Downstream • Port transmits a Poisoned Request Upstream If the `PCIE_RC_STATCMD_[n].PERREN` bit is set to 0, this bit is never set. Default value of this bit is 0.
23 (R/NW)	FASTB2B	Fast back-to-back transactions enable. Does not apply to PCI express and must be set to 0.
21 (R/NW)	FAST66	66 MHz Capable. Does not apply to PCI express and must be set to 0.

Table 29-206: PCIE_RC_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	CAPBLIST	Capabilities List. The `PCIE_RC_STATCMD_[n].CAPBLIST` bit indicates the presence of an Extended Capability list item. Since all PCI Express device Functions are required to implement the PCI Express Capability structure, this bit is always set to 1.
19 (R/W)	ISTAT	Interrupt Status. The `PCIE_RC_STATCMD_[n].ISTAT` bit indicates that an INTx emulation interrupt is pending internally in the function. Note that INTx emulation interrupts forwarded by Root and Switch Ports from devices Downstream of the Root or Switch Port are not reflected in this bit. Default value of this bit is 0.
10 (R/W)	IEN	Interrupt Enable. The `PCIE_RC_STATCMD_[n].IEN` bit controls the ability of a PCI Express Function to generate INTx interrupts. When set, functions are prevented from asserting INTx interrupts. Any INTx emulation interrupts already asserted by the function must be de-asserted when this bit is set. Note that INTx interrupts use virtual wires that must, if asserted, be de-asserted using the appropriate Deassert_INTx message(s) when this bit is Set. Only the INTx virtual wire interrupt(s) associated with the function(s) for which this bit is Set are affected. For Endpoints that generate INTx interrupts, this bit is required. The `PCIE_RC_STATCMD_[n].IEN` bit has no effect on interrupts that pass through the port from the secondary side. For root ports, switch ports, and bridges that do not generate INTx interrupts on their own behalf this bit is optional. If not implemented, this bit must be hardwired to 0. Default value of this bit is 0.
8 (R/W)	SERREN	Non-fatal and fatal Error reporting Enable. When the `PCIE_RC_STATCMD_[n].SERREN` bit is set, the PCIE reports non-fatal and fatal errors detected by the function to the root complex. Note that errors are reported if enabled either through this bit or through the PCI Express specific bits in the Device Control register. In addition, for functions with Type 1 configuration space headers, this bit controls transmission by the primary interface of ERR_NONFATAL and ERR_FATAL error Messages forwarded from the secondary interface. This bit does not affect the transmission of forwarded ERR_COR messages. A root complex integrated endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. The default value of this bit is 0.
7 (R/NW)	IDSELSTEP	IDSEL Stepping/Wait Cycle Control. Does not apply to PCI Express and must be set to 0.

Table 29-206: PCIE_RC_STATCMD_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	PERREN	Parity Error Response. The `PCIE_RC_STATCMD_[n].PERREN` bit controls the logging of poisoned TLPs in the `PCIE_EP_STATCMD_[n].MSTDPERR` bit. A Root Complex Integrated Endpoint that is not associated with a Root Complex Event Collector is permitted to hardwire this bit to 0. Default value of this bit is 0.
5 (R/NW)	VGAPS	VGA Palette Snoop. Does not apply to PCI Express and must be set to 0.
4 (R/NW)	MWIE	Memory Write and Invalidate Enable. Does not apply to PCI Express and must be set to 0.
3 (R/NW)	SCO	Special Cycle Operation. Does not apply to PCI Express and must be set to 0.
2 (R/W)	BME	Bus Master Enable. The `PCIE_RC_STATCMD_[n].BME` bit controls the ability of a PCI Express Endpoint to issue Memory and I/O Read/Write Requests, and the ability of a Root or Switch Port to forward Memory and I/O Read/Write Requests in the Upstream direction. For root complex, the `PCIE_RC_STATCMD_[n].BME` bit controls forwarding of memory or I/O requests by a switch or root port in the upstream direction. When this bit is 0, memory and I/O requests received at a root port or the downstream side of a switch port must be handled as unsupported requests (UR), and for non-posted requests a completion with UR completion status must be returned. This bit does not affect forwarding of completions in either the upstream or downstream direction. The forwarding of requests other than memory or I/O requests is not controlled by this bit. The default value of this bit is 0.
1 (R/W)	MSE	Memory Space Enable. The `PCIE_RC_STATCMD_[n].MSE` bit enables memory space.
0 (R/W)	IOE	I/O Space Enable. The `PCIE_RC_STATCMD_[n].IOE` bit enables I/O space.

Root Status Register

The `PCIE_RC_STAT_[n]` register provides information about PCI Express device specific parameters.

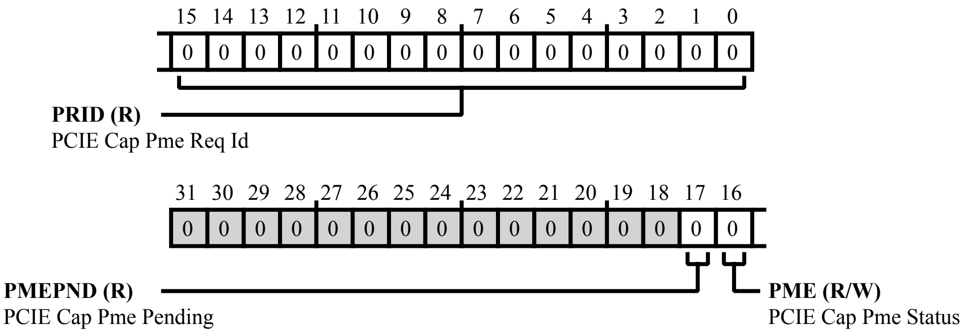

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PRID (R)
PCIE Cap Pme Req Id
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PMEPND (R)
PCIE Cap Pme Pending
PME (R/W)
PCIE Cap Pme Status

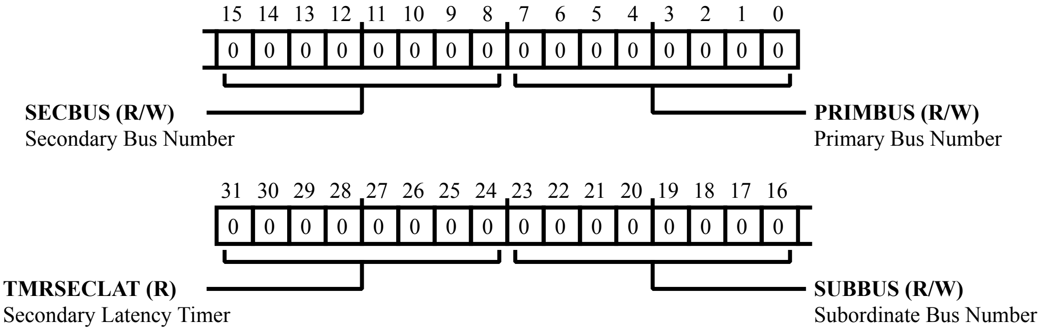
Figure 29-198: PCIE_RC_STAT_[n] Register Diagram

Table 29-207: PCIE_RC_STAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	PMEPND	PCIE Cap Pme Pending. The `PCIE_RC_STAT_[n].PMEPND` bit indicates that another PME is pending when the PME Status bit is Set. When the PME Status bit is cleared by software; the PME is delivered by hardware by setting the PME Status bit again and updating the PME Requester ID field appropriately. The PME Pending bit is cleared by hardware if no more PMEs are pending.
16 (R/W)	PME	PCIE Cap Pme Status. The `PCIE_RC_STAT_[n].PME` bit indicates that PME was asserted by the PME Requester indicated in the PME Requester ID field. Subsequent PMEs are kept pending until the status register is cleared by software by writing a 1. Default value of this bit is 0.
15:0 (R/NW)	PRID	PCIE Cap Pme Req Id. The `PCIE_RC_STAT_[n].PRID` bit field indicates the PCI Requester ID of the last PME Requester. This field is only valid when the PME Status bit is Set.

Latency Timer Register

The `PCIE_RC_TMRLAT_[n]` register provides the primary, secondary and subordinate bus numbers and the latency timer value.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SECBUS (R/W)
Secondary Bus Number
PRIMBUS (R/W)
Primary Bus Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TMRSECLAT (R)
Secondary Latency Timer
SUBBUS (R/W)
Subordinate Bus Number

Figure 29-199: PCIE_RC_TMRLAT_[n] Register Diagram

Table 29-208: PCIE_RC_TMRLAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	TMRSECLAT	Secondary Latency Timer.
23:16 (R/W)	SUBBUS	Subordinate Bus Number.
15:8 (R/W)	SECBUS	Secondary Bus Number.
7:0 (R/W)	PRIMBUS	Primary Bus Number.

Reset and Clock Control Register

The `PCIE_RSCK_CTL` register contains bits that control various reset and clocking options.

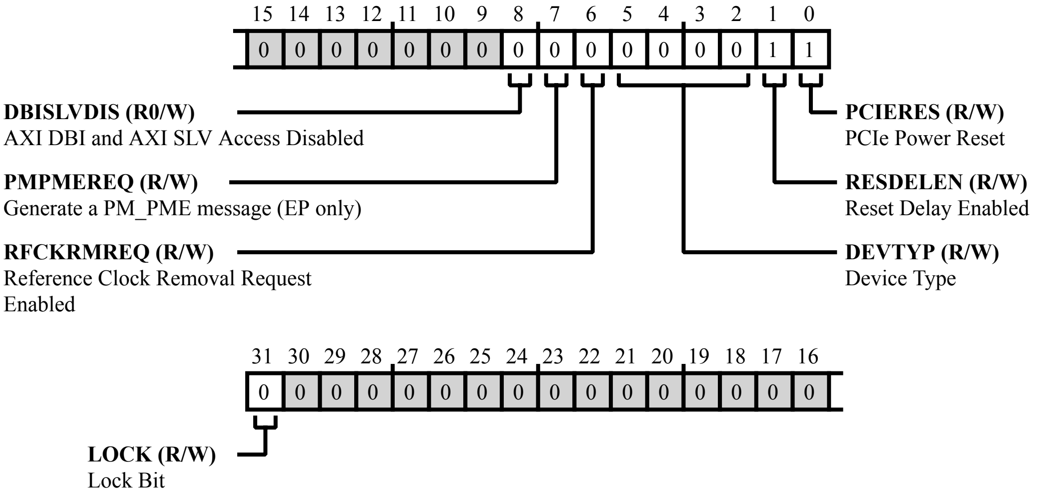

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
DBISLVDIS (R0/W)
AXI DBI and AXI SLV Access Disabled
PMPMEREQ (R/W)
Generate a PM_PME message (EP only)
RFCKRMREQ (R/W)
Reference Clock Removal Request Enabled
PCIERES (R/W)
PCIe Power Reset
RESDELEN (R/W)
Reset Delay Enabled
DEVTYP (R/W)
Device Type
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LOCK (R/W)
Lock Bit

Figure 29-200: PCIE_RSCK_CTL Register Diagram

Table 29-209: PCIE_RSCK_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit.	
		0	Register is not locked
		1	Register is locked
8 (R0/W)	DBISLVDIS	AXI DBI and AXI SLV Access Disabled.	
7 (R/W)	PMPMEREQ	Generate a PM_PME message (EP only).	
6 (R/W)	RFCKRMREQ	Reference Clock Removal Request Enabled.	
		0	Request to remove the reference clock disabled
		1	Request to remove the reference clock enabled
5:2 (R/W)	DEVTYP	Device Type.	
		0	PCIe Endpoint
		4	PCIe Root Complex
1 (R/W)	RESDELEN	Reset Delay Enabled.	
		0	RSCKPHY_SYSRESb de-assertion is not delayed
		1	RSCKPHY_SYSRESb de-assertion is delayed

Table 29-209: PCIE_RSCK_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PCIERES	PCIe Power Reset.	
		0	PCIe Power Reset is not asserted
		1	PCIe Power Reset is asserted

Reset and Clock Status

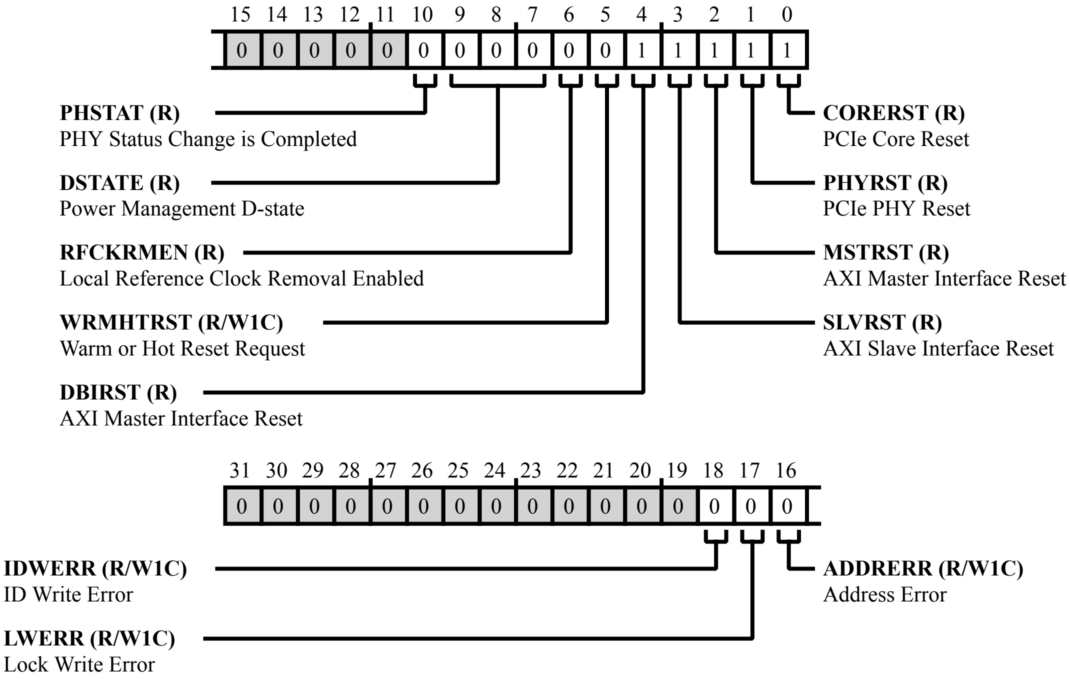

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
PHSTAT (R)
PHY Status Change is Completed
DSTATE (R)
Power Management D-state
RFCKRMEN (R)
Local Reference Clock Removal Enabled
WRMHTRST (R/W1C)
Warm or Hot Reset Request
DBIRST (R)
AXI Master Interface Reset
CORERST (R)
PCIe Core Reset
PHYRST (R)
PCIe PHY Reset
MSTRST (R)
AXI Master Interface Reset
SLVRST (R)
AXI Slave Interface Reset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IDWERR (R/W1C)
ID Write Error
LWERR (R/W1C)
Lock Write Error
ADDRERR (R/W1C)
Address Error

Figure 29-201: PCIE_RSCK_STAT Register Diagram

Table 29-210: PCIE_RSCK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	IDWERR	ID Write Error.	
		0	No Error
		1	Address Write ID and Write ID are different
17 (R/W1C)	LWERR	Lock Write Error.	
		0	No Lock Write Error
		1	Lock Write Error
16 (R/W1C)	ADDRERR	Address Error.	
		0	No Error
		1	o Accesses to an unimplemented address, writes to Read Only registers, different than single 32-bit aligned
10 (R/NW)	PHSTAT	PHY Status Change is Completed.	
9:7 (R/NW)	DSTATE	Power Management D-state.	

Table 29-210: PCIE_RSCK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	D0 Power State
		1	D1 Power State
		2	D2 Power State
		3	D3 Power State
		4	D0 Uninitialized
6 (R/NW)	RFCKRMEN	Local Reference Clock Removal Enabled.	
		0	Local Reference Clock is needed
		1	Local Reference Clock can be removed
5 (R/W1C)	WRMHTRST	Warm or Hot Reset Request.	
		0	No Reset Request
		1	PCIe_DM_CORE issued a Reset Request
4 (R/NW)	DBIRST	AXI Master Interface Reset.	
		0	The AXI DBI IF is out of Reset
		1	The AXI DBI IF is Reset
3 (R/NW)	SLVRST	AXI Slave Interface Reset.	
		0	The AXI Slave IF is out of Reset
		1	The AXI Slave IF is Reset
2 (R/NW)	MSTRST	AXI Master Interface Reset.	
		0	The AXI Master IF is out of Reset
		1	The AXI Master IF is Reset
1 (R/NW)	PHYRST	PCIe PHY Reset.	
		0	The PCIe PHY is out of Reset
		1	The PCIe PHY is Reset
0 (R/NW)	CORERST	PCIe Core Reset.	
		0	The PCIe Core is out of reset
		1	The PCIe Core is Reset

Symbol Timer Filter 1 Off

Bits [31:16] of the `PCIE_SYM_TMRFILT1_[n]` register are the Mask RADM Filtering and Error Handling Rules for Mask 1. There are several mask bits to turn off the filtering and error handling rules (For more details, see "Receive Filtering". In each case, 0 applies the associated filtering rule and 1 masks the associated filtering rule. Default is DEFAULT_FILTER_MSK_1.

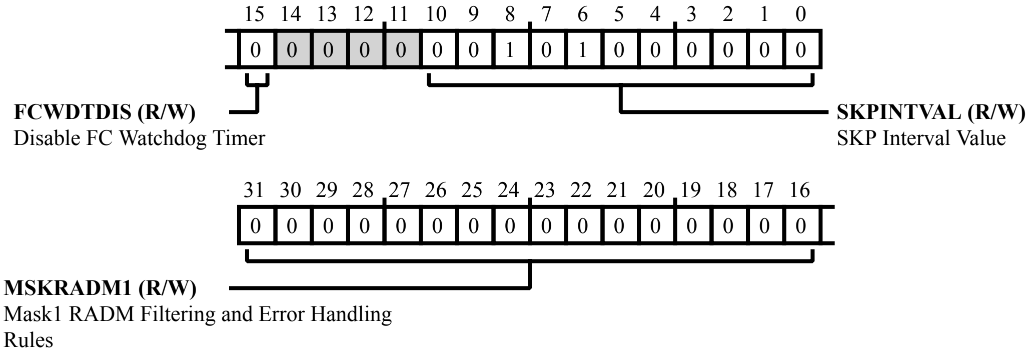

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
FCWDTDIS (R/W)
Disable FC Watchdog Timer
SKPINTVAL (R/W)
SKP Interval Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MSKRADM1 (R/W)
Mask1 RADM Filtering and Error Handling Rules

Figure 29-202: PCIE_SYM_TMRFILT1_[n] Register Diagram

Table 29-211: PCIE_SYM_TMRFILT1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	MSKRADM1	Mask1 RADM Filtering and Error Handling Rules. The `PCIE_SYM_TMRFILT1_[n].MSKRADM1` bit field mask bits turn off the filtering and error handling rules. In each case, 0 applies the associated filtering rule and 1 masks the associated filtering rule.
15 (R/W)	FCWDTDIS	Disable FC Watchdog Timer. The `PCIE_SYM_TMRFILT1_[n].FCWDTDIS` bit disables the FC Watchdog Timer. This register field is sticky.
10:0 (R/W)	SKPINTVAL	SKP Interval Value. The `PCIE_SYM_TMRFILT1_[n].SKPINTVAL` bit field sets the number of symbol times to wait between transmitting SKP ordered sets. Note that the core actually waits the number of symbol times in this register plus 1 between transmitting SKP ordered sets. The application must program this register accordingly. For example, if 1536 is programmed into this bit field (in a 250 MHz core), then the core actually transmits SKP ordered sets once every 1537 symbol times. The value programmed to this register is actually clock ticks and not symbol times. In a 125 MHz core, programming the value programmed to this register should be scaled down by a factor of 2 (because 1 clock tick = 2 symbol times in this case). This bit field is sticky.

Timer Control and Max Function Number Register

The `PCIE_TMRCTL_MFN_[n]` register provides bits that configure and control the timer and requests targeted at function numbers.

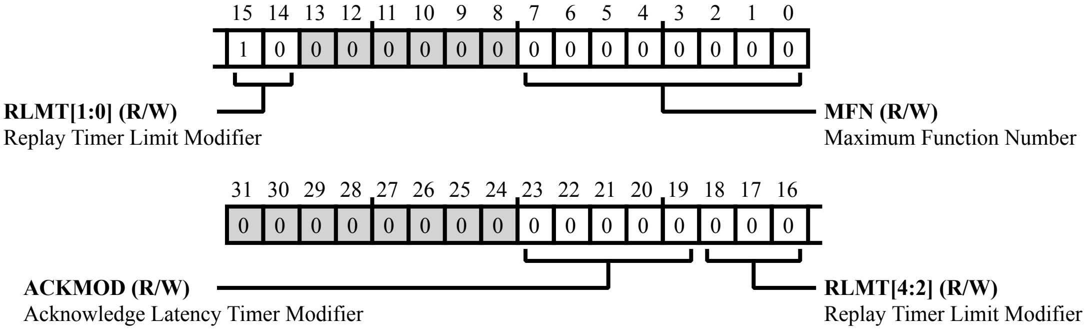

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RLMT[1:0] (R/W)
Replay Timer Limit Modifier
MFN (R/W)
Maximum Function Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ACKMOD (R/W)
Acknowledge Latency Timer Modifier
RLMT[4:2] (R/W)
Replay Timer Limit Modifier

Figure 29-203: PCIE_TMRCTL_MFN_[n] Register Diagram

Table 29-212: PCIE_TMRCTL_MFN_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:19 (R/W)	ACKMOD	Acknowledge Latency Timer Modifier. The `PCIE_TMRCTL_MFN_[n].ACKMOD` bit field increases the timer value for the Ack latency timer in increments of 64 clock cycles. A value of 0 represents no modification to the timer value. <br><i>Note</i>: This register field is sticky.
18:14 (R/W)	RLMT	Replay Timer Limit Modifier. The `PCIE_TMRCTL_MFN_[n].RLMT` bit field increases the time-out value for the replay timer in increments of 64 clock cycles at Gen1 or Gen2 speed, and in increments of 256 clock cycles at Gen3 speed. A value of 0 represents no modification to the timer limit. At Gen3 speed, the core automatically changes the value of this field to DEFAULT_GEN3_REPLAY_ADJ. <br><i>Note</i>: This register field is sticky.
7:0 (R/W)	MFN	Maximum Function Number. The `PCIE_TMRCTL_MFN_[n].MFN` bit field configures requests targeted at function numbers. Value higher than this are returned with UR (unsupported request). This register field is sticky.

Transmit Completion FC Credit Status Register

The `PCIE_TXCOMP_FCCSTAT_[n]` register reports the completion data credits and the completion header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP.

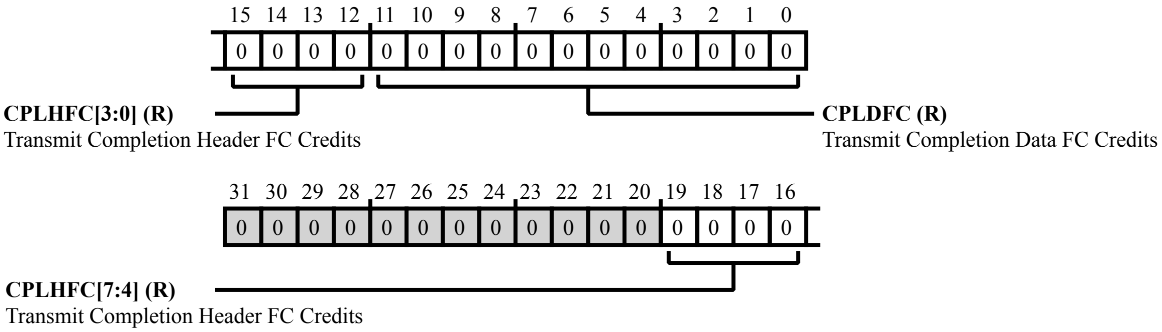

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPLHFC[3:0] (R)
Transmit Completion Header FC Credits
CPLDFC (R)
Transmit Completion Data FC Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPLHFC[7:4] (R)
Transmit Completion Header FC Credits

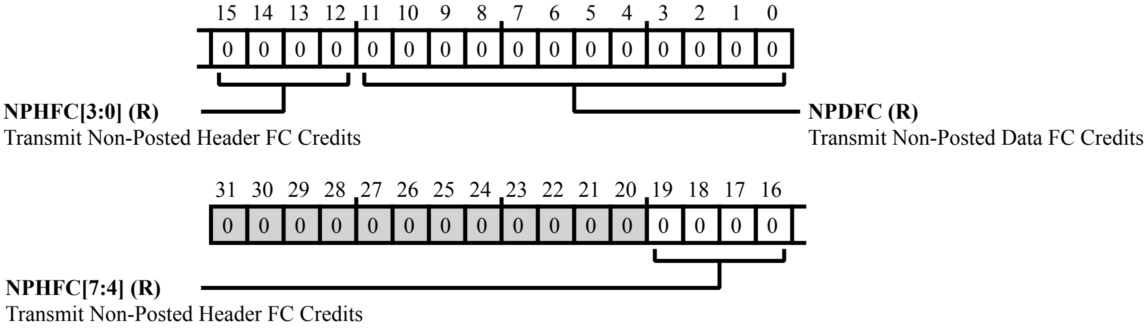
Figure 29-204: PCIE_TXCOMP_FCCSTAT_[n] Register Diagram

Table 29-213: PCIE_TXCOMP_FCCSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	CPLHFC	Transmit Completion Header FC Credits. The `PCIE_TXCOMP_FCCSTAT_[n].CPLHFC` bit field reports the Completion Header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. Default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_cplh_cdts, xtlh_xadm_cpld_cdts]; When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].
11:0 (R/NW)	CPLDFC	Transmit Completion Data FC Credits. The `PCIE_TXCOMP_FCCSTAT_[n].CPLDFC` bit field reports the completion data credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. Default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_cplh_cdts, xtlh_xadm_cpld_cdts]; When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].

Transmit Non-Posted FC Credit Status Register

The `PCIE_TX_NPST_FCCSTAT_[n]` register reports the non-posted data credits and the non-posted header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NPHFC[3:0] (R)
Transmit Non-Posted Header FC Credits
NPDFC (R)
Transmit Non-Posted Data FC Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NPHFC[7:4] (R)
Transmit Non-Posted Header FC Credits

Figure 29-205: PCIE_TX_NPST_FCCSTAT_[n] Register Diagram

Table 29-214: PCIE_TX_NPST_FCCSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	NPHFC	Transmit Non-Posted Header FC Credits. The `PCIE_TX_NPST_FCCSTAT_[n].NPHFC` bit field reports the non-posted header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. Default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_nph_cdts, xtlh_xadm_npd_cdts]; When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].
11:0 (R/NW)	NPDFC	Transmit Non-Posted Data FC Credits. The `PCIE_TX_NPST_FCCSTAT_[n].NPDFC` bit field reports the non-posted data credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. Default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_nph_cdts, xtlh_xadm_npd_cdts]; When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].

Transmit Posted FC Credit Status Register

The `PCIE_TX_PST_FCCSTAT_[n]` register reports the posted data credits and the posted header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP.

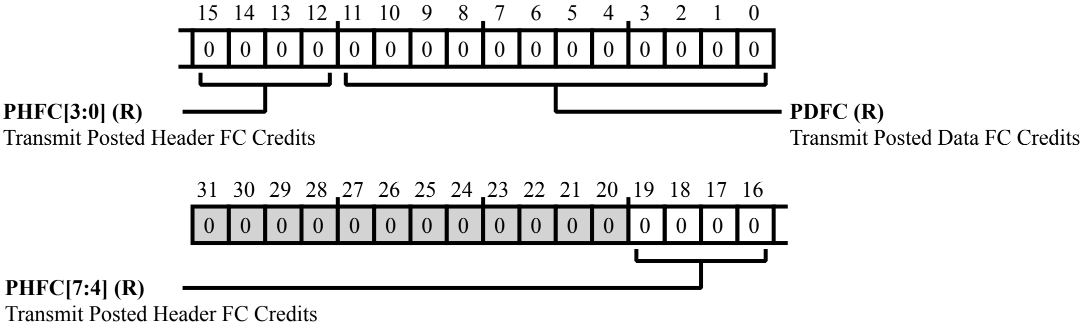

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHFC[3:0] (R)
Transmit Posted Header FC Credits
PDFC (R)
Transmit Posted Data FC Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHFC[7:4] (R)
Transmit Posted Header FC Credits

Figure 29-206: PCIE_TX_PST_FCCSTAT_[n] Register Diagram

Table 29-215: PCIE_TX_PST_FCCSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	PHFC	Transmit Posted Header FC Credits. The `PCIE_TX_PST_FCCSTAT_[n].PHFC` bit field reports the posted header credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. The default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_ph_cdts, xtlh_xadm_pd_cdts]; When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].
11:0 (R/NW)	PDFC	Transmit Posted Data FC Credits. The `PCIE_TX_PST_FCCSTAT_[n].PDFC` bit field reports the posted data credits advertised by the receiver at the other end of the link, updated with each UpdateFC DLLP. The default value depends on the number of advertised credits for header and data [12'b0, xtlh_xadm_ph_cdts, xtlh_xadm_pd_cdts]. When the number of advertised completion credits (both header and data) are infinite, then the default is [12'b0, 8'hFF, 12'hFFF].

Uncorrectable Error Mask Register

The `PCIE_UNCOR_ERRMSK_[n]` register controls reporting of individual errors by the device Function to the PCI Express Root Complex via a PCI Express error message. A masked error (indicated by a set bit) is not recorded or reported in the Header Log register. Also the First Error Pointer is not updated or reported to the PCI Express Root Complex by this function. There is a mask bit per error bit of the Uncorrectable Error Status register (`PCIE_UNCOR_ERRSTAT_[n]`). Register fields for bits not implemented by the Function are set to 0.

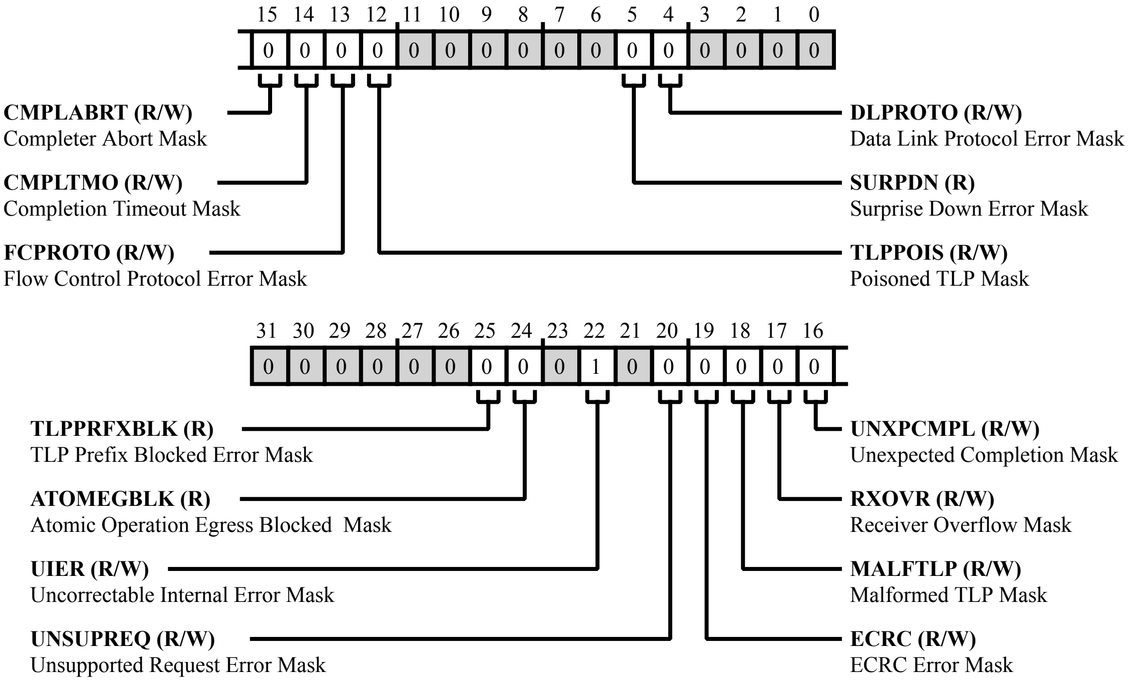

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CMPLABRT (R/W)
Completer Abort Mask
CMPLTMO (R/W)
Completion Timeout Mask
FCPROTO (R/W)
Flow Control Protocol Error Mask
DLPROTO (R/W)
Data Link Protocol Error Mask
SURPDN (R)
Surprise Down Error Mask
TLPPOIS (R/W)
Poisoned TLP Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
TLPPRFXBLK (R)
TLP Prefix Blocked Error Mask
ATOMEGBLK (R)
Atomic Operation Egress Blocked Mask
UIER (R/W)
Uncorrectable Internal Error Mask
UNSUPREQ (R/W)
Unsupported Request Error Mask
UNXPCMPL (R/W)
Unexpected Completion Mask
RXOVR (R/W)
Receiver Overflow Mask
MALFTLP (R/W)
Malformed TLP Mask
ECRC (R/W)
ECRC Error Mask

Figure 29-207: PCIE_UNCOR_ERRMSK_[n] Register Diagram

Table 29-216: PCIE_UNCOR_ERRMSK_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	TLPPRFXBLK	TLP Prefix Blocked Error Mask. The `PCIE_UNCOR_ERRMSK_[n].TLPPRFXBLK` bit indicates whether the LP Prefix Blocked Error Severity error is masked (=1) or unmasked (=0).
24 (R/NW)	ATOMEGBLK	Atomic Operation Egress Blocked Mask. The `PCIE_UNCOR_ERRMSK_[n].ATOMEGBLK` bit indicates whether the AtomicOp Egress Blocked Severity error is masked (=1) or unmasked (=0).
22 (R/W)	UIER	Uncorrectable Internal Error Mask. The `PCIE_UNCOR_ERRMSK_[n].UIER` bit indicates whether the Uncorrectable Internal Error Severity error is masked (=1) or unmasked (=0).

Table 29-216: PCIE_UNCOR_ERRMSK_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	UNSUPREQ	Unsupported Request Error Mask. The `PCIE_UNCOR_ERRMSK_[n].UNSUPREQ` bit indicates whether the Unsupported Request Error Severity error is masked (=1) or unmasked (=0).
19 (R/W)	ECRC	ECRC Error Mask. The `PCIE_UNCOR_ERRMSK_[n].ECRC` bit indicates whether the ECRC Error Severity error is masked (=1) or unmasked (=0).
18 (R/W)	MALFTLP	Malformed TLP Mask. The `PCIE_UNCOR_ERRMSK_[n].MALFTLP` bit indicates whether the Malformed TLP Severity error is masked (=1) or unmasked (=0).
17 (R/W)	RXOVR	Receiver Overflow Mask. The `PCIE_UNCOR_ERRMSK_[n].RXOVR` bit indicates whether the Receiver Overflow Error Severity error is masked (=1) or unmasked (=0).
16 (R/W)	UNXPCMPL	Unexpected Completion Mask. The `PCIE_UNCOR_ERRMSK_[n].UNXPCMPL` bit indicates whether the Unexpected Completion Error Severity error is masked (=1) or unmasked (=0).
15 (R/W)	CMPLABRT	Completer Abort Mask. The `PCIE_UNCOR_ERRMSK_[n].CMPLABRT` bit indicates whether the Completer Abort Error Severity error is masked (=1) or unmasked (=0).
14 (R/W)	CMPLTMO	Completion Timeout Mask. The `PCIE_UNCOR_ERRMSK_[n].CMPLTMO` bit indicates whether the Completion Timeout Error Severity error is masked (=1) or unmasked (=0).
13 (R/W)	FCPROTO	Flow Control Protocol Error Mask. The `PCIE_UNCOR_ERRMSK_[n].FCPROTO` bit indicates whether the Flow Control Protocol Error Severity error is masked (=1) or unmasked (=0).
12 (R/W)	TLPPOIS	Poisoned TLP Mask. The `PCIE_UNCOR_ERRMSK_[n].TLPPOIS` bit indicates whether the Poisoned TLP Severity error is masked (=1) or unmasked (=0).
5 (R/NW)	SURPDN	Surprise Down Error Mask. The `PCIE_UNCOR_ERRMSK_[n].SURPDN` bit indicates whether the Surprise Down Error Severity error is masked (=1) or unmasked (=0).
4 (R/W)	DLPROTO	Data Link Protocol Error Mask. The `PCIE_UNCOR_ERRMSK_[n].DLPROTO` bit indicates whether the Data Link Protocol Error Severity error is masked (=1) or unmasked (=0).

Uncorrectable Error Severity Register

The `PCIE_UNCOR_ERRSEV_[n]` register controls whether an individual error is reported as a Nonfatal or Fatal error. An error is reported as fatal when the corresponding error bit in the severity register is set (=1). If the bit is clear (=0), the corresponding error is considered non-fatal. Register fields for bits not implemented by the Function are hardwired to an implementation specific value.

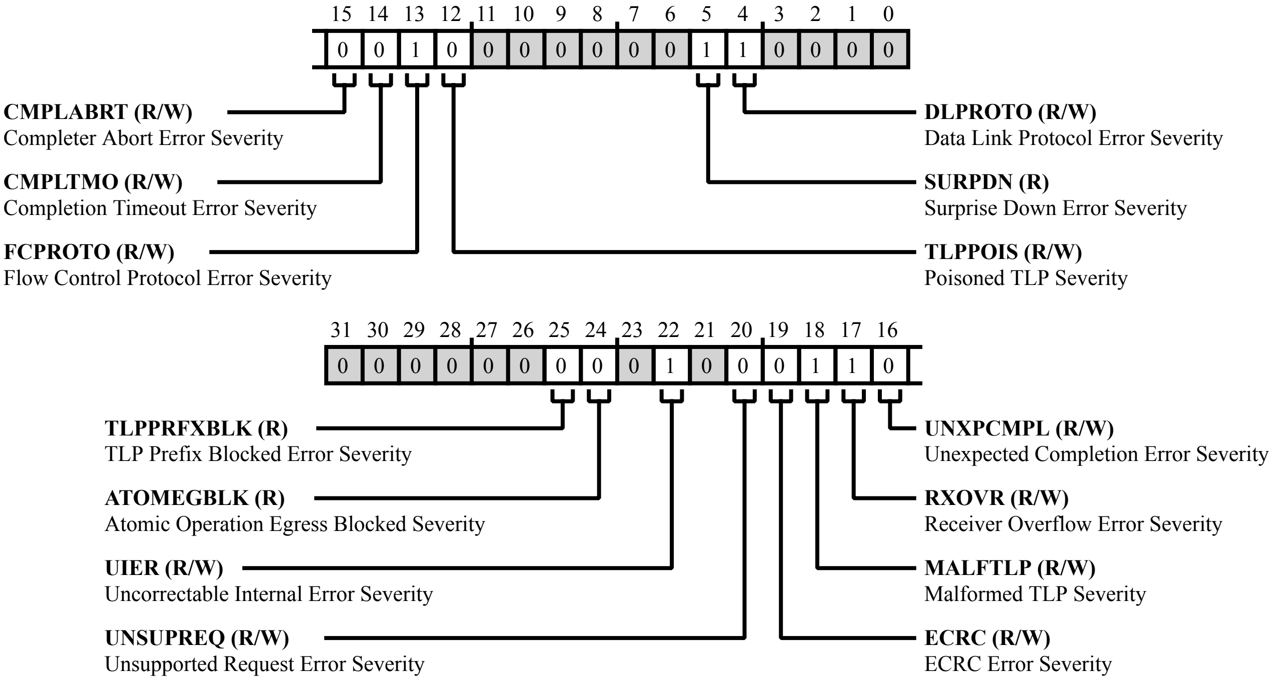

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0
CMPLABRT (R/W)
Completer Abort Error Severity
CMPLTMO (R/W)
Completion Timeout Error Severity
FCPROTO (R/W)
Flow Control Protocol Error Severity
DLPROTO (R/W)
Data Link Protocol Error Severity
SURPDN (R)
Surprise Down Error Severity
TLPPOIS (R/W)
Poisoned TLP Severity
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0
TLPPRFXBLK (R)
TLP Prefix Blocked Error Severity
ATOMEGBLK (R)
Atomic Operation Egress Blocked Severity
UIER (R/W)
Uncorrectable Internal Error Severity
UNSUPREQ (R/W)
Unsupported Request Error Severity
UNXPCMPL (R/W)
Unexpected Completion Error Severity
RXOVR (R/W)
Receiver Overflow Error Severity
MALFTLP (R/W)
Malformed TLP Severity
ECRC (R/W)
ECRC Error Severity

Figure 29-208: PCIE_UNCOR_ERRSEV_[n] Register Diagram

Table 29-217: PCIE_UNCOR_ERRSEV_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	TLPPRFXBLK	TLP Prefix Blocked Error Severity. The `PCIE_UNCOR_ERRSEV_[n].TLPPRFXBLK` bit indicates that the TLP Prefix Blocked Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
24 (R/NW)	ATOMEGBLK	Atomic Operation Egress Blocked Severity. The `PCIE_UNCOR_ERRSEV_[n].ATOMEGBLK` bit indicates that the AtomicOp Egress Blocked Severity error is fatal (=1) or non-fatal (=0) non-fatal.
22 (R/W)	UIER	Uncorrectable Internal Error Severity. The `PCIE_UNCOR_ERRSEV_[n].UIER` bit indicates that the Uncorrectable Internal Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.

Table 29-217: PCIE_UNCOR_ERRSEV_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	UNSUPREQ	Unsupported Request Error Severity. The `PCIE_UNCOR_ERRSEV_[n].UNSUPREQ` bit indicates that the Unsupported Request Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
19 (R/W)	ECRC	ECRC Error Severity. The `PCIE_UNCOR_ERRSEV_[n].ECRC` bit indicates that the ECRC Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
18 (R/W)	MALFTLP	Malformed TLP Severity. The `PCIE_UNCOR_ERRSEV_[n].MALFTLP` bit indicates that the Malformed TLP Severity error is fatal (=1) or non-fatal (=0) non-fatal.
17 (R/W)	RXOVR	Receiver Overflow Error Severity. The `PCIE_UNCOR_ERRSEV_[n].RXOVR` bit indicates that the Receiver Overflow Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
16 (R/W)	UNXPCMPL	Unexpected Completion Error Severity. The `PCIE_UNCOR_ERRSEV_[n].UNXPCMPL` bit indicates that the Unexpected Completion Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
15 (R/W)	CMPLABRT	Completer Abort Error Severity. The `PCIE_UNCOR_ERRSEV_[n].CMPLABRT` bit indicates that the Completer Abort Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
14 (R/W)	CMPLTMO	Completion Timeout Error Severity. The `PCIE_UNCOR_ERRSEV_[n].CMPLTMO` bit indicates that the Completion Timeout Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
13 (R/W)	FCPROTO	Flow Control Protocol Error Severity. The `PCIE_UNCOR_ERRSEV_[n].FCPROTO` bit indicates that the Flow Control Protocol Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
12 (R/W)	TLPPOIS	Poisoned TLP Severity. The `PCIE_UNCOR_ERRSEV_[n].TLPPOIS` bit indicates that the Poisoned TLP Severity error is fatal (=1) or non-fatal (=0) non-fatal.
5 (R/NW)	SURPDN	Surprise Down Error Severity. The `PCIE_UNCOR_ERRSEV_[n].SURPDN` bit indicates that the Surprise Down Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.
4 (R/W)	DLPROTO	Data Link Protocol Error Severity. The `PCIE_UNCOR_ERRSEV_[n].DLPROTO` bit indicates that the Data Link Protocol Error Severity error is fatal (=1) or non-fatal (=0) non-fatal.

Uncorrectable Error Status Register

The `PCIE_UNCOR_ERRSTAT_[n]` register indicates error detection status of individual errors on a PCI Express device Function. An individual error status bit that is set (=1) indicates that a particular error was detected. Software may clear an error status by writing a 1 to the respective bit. Register bits not implemented by the Function are set to 0.

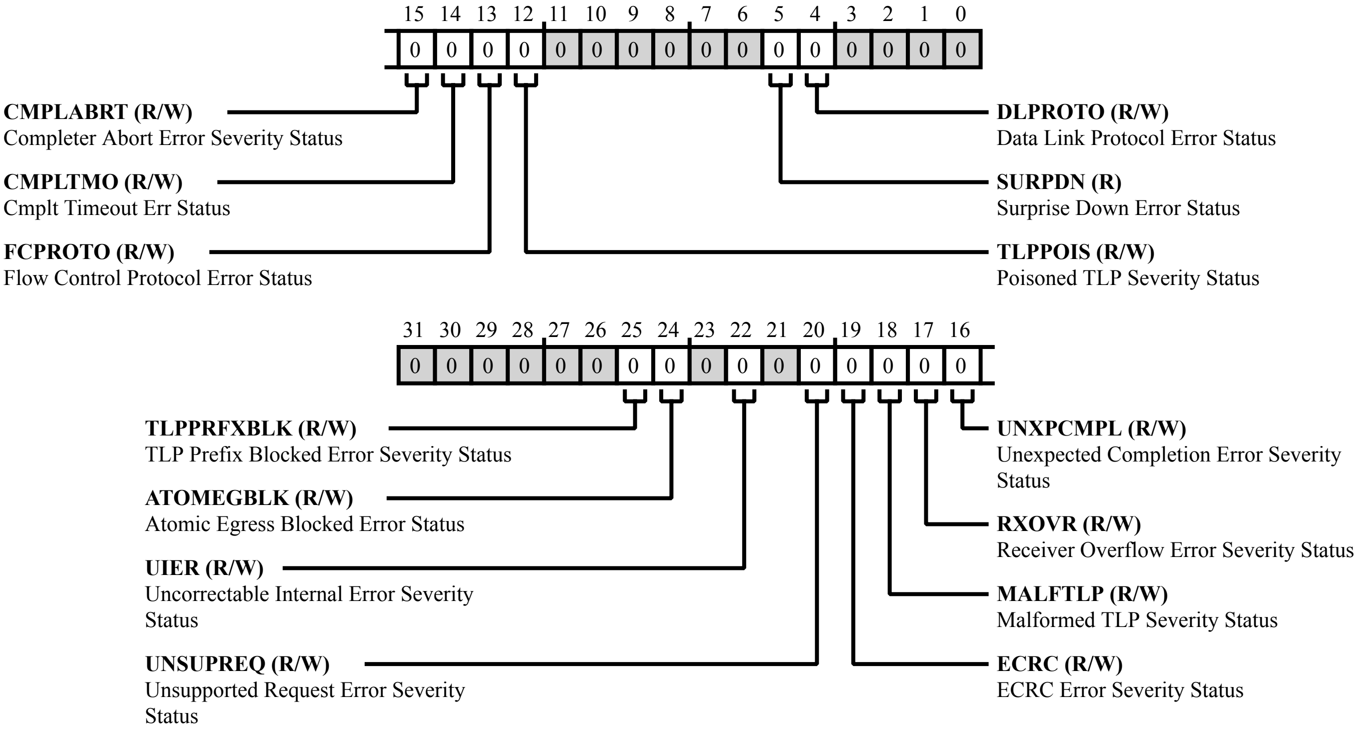

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CMPLABRT (R/W)
Completer Abort Error Severity Status
CMPLTMO (R/W)
Cmplt Timeout Err Status
FCPROTO (R/W)
Flow Control Protocol Error Status
DLPROTO (R/W)
Data Link Protocol Error Status
SURPDN (R)
Surprise Down Error Status
TLPPOIS (R/W)
Poisoned TLP Severity Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TLPPRFXBLK (R/W)
TLP Prefix Blocked Error Severity Status
ATOMEGBLK (R/W)
Atomic Egress Blocked Error Status
UIER (R/W)
Uncorrectable Internal Error Severity Status
UNSUPREQ (R/W)
Unsupported Request Error Severity Status
UNXPCMPL (R/W)
Unexpected Completion Error Severity Status
RXOVR (R/W)
Receiver Overflow Error Severity Status
MALFTLP (R/W)
Malformed TLP Severity Status
ECRC (R/W)
ECRC Error Severity Status

Figure 29-209: PCIE_UNCOR_ERRSTAT_[n] Register Diagram

Table 29-218: PCIE_UNCOR_ERRSTAT_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	TLPPRFXBLK	TLP Prefix Blocked Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].TLPPRFXBLK` bit indicates that a TLP Prefix Blocked Error Severity error occurred (=1) or did not occur (=0).
24 (R/W)	ATOMEGBLK	Atomic Egress Blocked Error Status. The `PCIE_UNCOR_ERRSTAT_[n].ATOMEGBLK` bit indicates that a AtomicOp Egress Blocked Severity error occurred (=1) or did not occur (=0).
22 (R/W)	UIER	Uncorrectable Internal Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].UIER` bit indicates that a Uncorrectable Internal Error Severity error occurred (=1) or did not occur (=0).

Table 29-218: PCIE_UNCOR_ERRSTAT_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	UNSUPREQ	Unsupported Request Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].UNSUPREQ` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
19 (R/W)	ECRC	ECRC Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].ECRC` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
18 (R/W)	MALFTLP	Malformed TLP Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].MALFTLP` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
17 (R/W)	RXOVR	Receiver Overflow Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].RXOVR` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
16 (R/W)	UNXPCMPL	Unexpected Completion Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].UNXPCMPL` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
15 (R/W)	CMPLABRT	Completer Abort Error Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].CMPLABRT` bit indicates that a Completer Abort Error Severity error occurred (=1) or did not occur (=0).
14 (R/W)	CMPLTMO	Cmplt Timeout Err Status. The `PCIE_UNCOR_ERRSTAT_[n].CMPLTMO` bit indicates that a Completion Timeout Error Severity error occurred (=1) or did not occur (=0).
13 (R/W)	FCPROTO	Flow Control Protocol Error Status. The `PCIE_UNCOR_ERRSTAT_[n].FCPROTO` bit indicates that a Flow Control Protocol Error Severity error occurred (=1) or did not occur (=0).
12 (R/W)	TLPPOIS	Poisoned TLP Severity Status. The `PCIE_UNCOR_ERRSTAT_[n].TLPPOIS` bit indicates that a Surprise Down Error Severity error occurred (=1) or did not occur (=0).
5 (R/NW)	SURPDN	Surprise Down Error Status. The `PCIE_UNCOR_ERRSTAT_[n].SURPDN` bit indicates that a Surprise Down Error Severity error occurred (=1) or did not occur (=0).
4 (R/W)	DLPROTO	Data Link Protocol Error Status. The `PCIE_UNCOR_ERRSTAT_[n].DLPROTO` bit indicates that a Data Link Protocol Error Severity error occurred (=1) or did not occur (=0).

Segmented-Buffer VC0 Completion Receive Queue Control Register

The `PCIE_VC0_COMP_RXQCTL_[n]` register reports VC0 completion data and header credits. Writing to the `PCIE_VC0_COMP_RXQCTL_[n]` register is only possible when CX_DYNAMIC_FC_CREDIT =1.

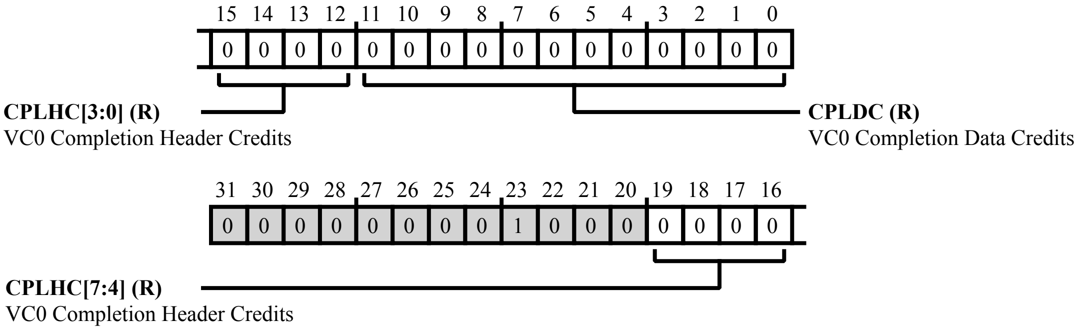

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPLHC[3:0] (R)
VC0 Completion Header Credits
CPLDC (R)
VC0 Completion Data Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
CPLHC[7:4] (R)
VC0 Completion Header Credits

Figure 29-210: PCIE_VC0_COMP_RXQCTL_[n] Register Diagram

Table 29-219: PCIE_VC0_COMP_RXQCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	CPLHC	VC0 Completion Header Credits. The `PCIE_VC0_COMP_RXQCTL_[n].CPLHC` bit field indicates the number of initial Completion header credits for VC0, used only in the segmented-buffer configuration. The access attributes of this field are - Wire: No access. - Dbi: R (sticky) - Dbi2: R (sticky)
11:0 (R/NW)	CPLDC	VC0 Completion Data Credits. The `PCIE_VC0_COMP_RXQCTL_[n].CPLDC` bit field indicates the number of initial Completion data credits for VC0, used only in the segmented-buffer configuration. he access attributes of this field are - Wire: No access. - Dbi: R (sticky) - Dbi2: R (sticky)

Segmented-Buffer VC0 Non-Posted Receive Queue Control Register

The `PCIE_VC0_NPST_RXQCTL_[n]` register reports VC0 non-posted data and header Credits. Writing to the `PCIE_VC0_NPST_RXQCTL_[n]` register is only possible when CX_DYNAMIC_FC_CREDIT =1.

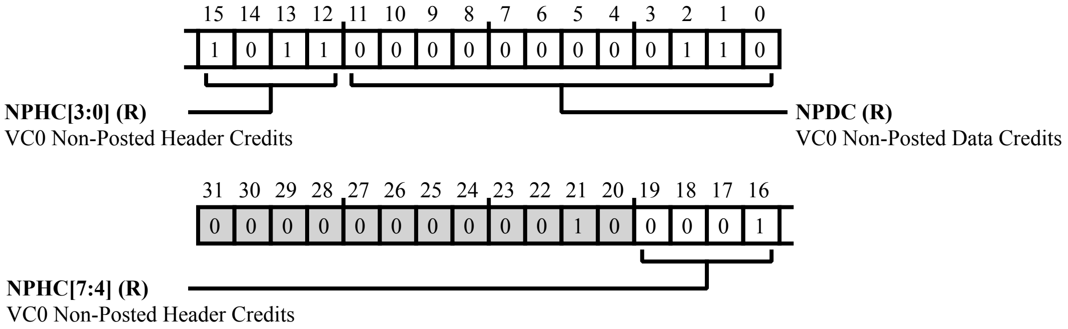

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0
NPHC[3:0] (R)
VC0 Non-Posted Header Credits
NPDC (R)
VC0 Non-Posted Data Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
NPHC[7:4] (R)
VC0 Non-Posted Header Credits

Figure 29-211: PCIE_VC0_NPST_RXQCTL_[n] Register Diagram

Table 29-220: PCIE_VC0_NPST_RXQCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	NPHC	VC0 Non-Posted Header Credits. The `PCIE_VC0_NPST_RXQCTL_[n].NPHC` bit field indicates the number of initial non-posted header credits for VC0, used only in the segmented-buffer configuration. The access attributes of this field are - Wire: R (sticky) - Dbi: R (sticky) - Dbi2: R (sticky)
11:0 (R/NW)	NPDC	VC0 Non-Posted Data Credits. The `PCIE_VC0_NPST_RXQCTL_[n].NPDC` bit field indicates the number of initial non-posted data credits for VC0, used only in the segmented-buffer configuration. The access attributes of this field are - Wire: R (sticky) - Dbi: R (sticky) - Dbi2: R (sticky)

Segmented-Buffer VC0 Posted Receive Queue Control Register

The `PCIE_VC0_PST_RXQCTL_[n]` register reports VC0 posted data and header credits. Writing to the `PCIE_VC0_PST_RXQCTL_[n]` register is only possible when CX_DYNAMIC_FC_CREDIT =1.

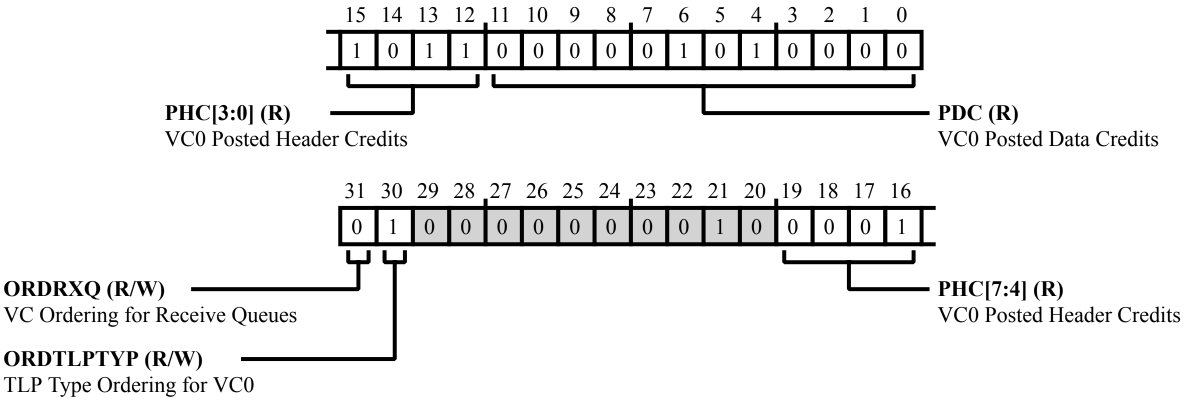

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0
PHC[3:0] (R)
VC0 Posted Header Credits
PDC (R)
VC0 Posted Data Credits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1
ORDRXQ (R/W)
VC Ordering for Receive Queues
ORDTLPTYP (R/W)
TLP Type Ordering for VC0
PHC[7:4] (R)
VC0 Posted Header Credits

Figure 29-212: PCIE_VC0_PST_RXQCTL_[n] Register Diagram

Table 29-221: PCIE_VC0_PST_RXQCTL_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	ORDRXQ	VC Ordering for Receive Queues. The `PCIE_VC0_PST_RXQCTL_[n].ORDRXQ` bit determines the VC ordering rule for the receive queues, used only in the segmented-buffer configuration. This bit is sticky.	
		0	Round robin
		1	Strict ordering, higher numbered VCs have higher priority
30 (R/W)	ORDTLPTYP	TLP Type Ordering for VC0. The `PCIE_VC0_PST_RXQCTL_[n].ORDTLPTYP` bit determines the TLP type ordering rule for VC0 receive queues, used only in the segmented-buffer configuration. This bit is sticky.	
		0	Strict ordering: posted, completion, then non-posted
		1	PCIe ordering rules

Table 29-221: PCIE_VC0_PST_RXQCTL_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:12 (R/NW)	PHC	VC0 Posted Header Credits. The `PCIE_VC0_PST_RXQCTL_[n].PHC` bit field indicates he number of initial posted header credits for VC0, used only in the segmented-buffer configuration. The access attributes of this field are as follows: • Wire: R (sticky) • Dbi: R (sticky) • Dbi2: R
11:0 (R/NW)	PDC	VC0 Posted Data Credits. The `PCIE_VC0_PST_RXQCTL_[n].PDC` bit field indicates the number of initial posted data credits for VC0, used only in the segmented-buffer configuration. The access attributes of this field are as follows: • Wire: R (sticky) • Dbi: R (sticky) • Dbi2: R

Virtual Channel Transmit Arbitration Register 1

The `PCIE_VC_TXARB1_[n]` register indicates the weighted round robin arbitration weights for the indicated virtual channels.

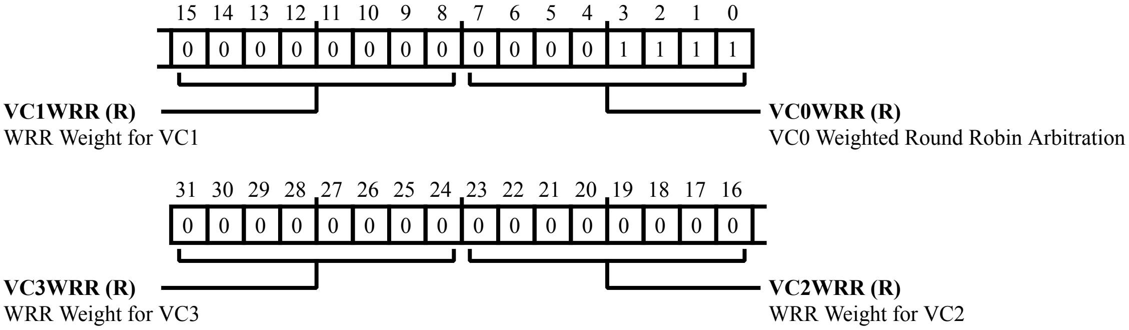

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
VC1WRR (R)
WRR Weight for VC1
VC0WRR (R)
VC0 Weighted Round Robin Arbitration
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VC3WRR (R)
WRR Weight for VC3
VC2WRR (R)
WRR Weight for VC2

Figure 29-213: PCIE_VC_TXARB1_[n] Register Diagram

Table 29-222: PCIE_VC_TXARB1_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	VC3WRR	WRR Weight for VC3. The `PCIE_VC_TXARB1_[n].VC3WRR` bit field indicates the WRR Weight for VC3. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R
23:16 (R/NW)	VC2WRR	WRR Weight for VC2. The `PCIE_VC_TXARB1_[n].VC2WRR` bit field indicates the WRR Weight for VC2. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R
15:8 (R/NW)	VC1WRR	WRR Weight for VC1. The `PCIE_VC_TXARB1_[n].VC1WRR` bit field indicates the WRR Weight for VC1. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R

Table 29-222: PCIE_VC_TXARB1_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VC0WRR	VC0 Weighted Round Robin Arbitration. The `PCIE_VC_TXARB1_[n].VC0WRR` bit field indicates the WRR Weight for VC0. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R

Virtual Channel Transmit Arbitration Register 2

The `PCIE_VC_TXARB2_[n]` register indicates the weighted round robin arbitration weights for the indicated virtual channels.

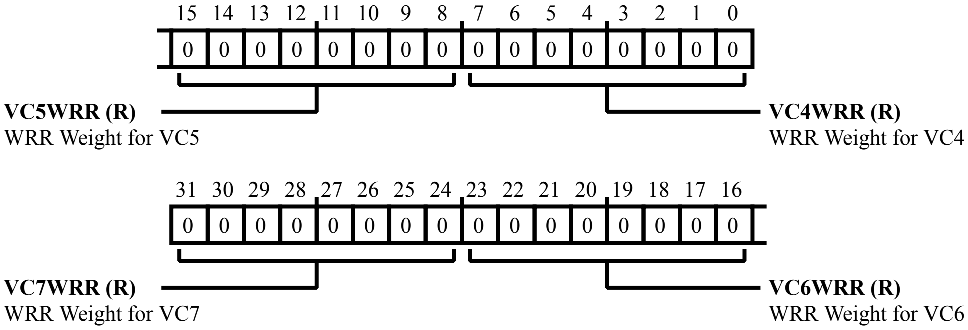

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VC5WRR (R)
WRR Weight for VC5
VC4WRR (R)
WRR Weight for VC4
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VC7WRR (R)
WRR Weight for VC7
VC6WRR (R)
WRR Weight for VC6

Figure 29-214: PCIE_VC_TXARB2_[n] Register Diagram

Table 29-223: PCIE_VC_TXARB2_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	VC7WRR	WRR Weight for VC7. The `PCIE_VC_TXARB2_[n].VC7WRR` bit field indicates the WRR Weight for VC7. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R
23:16 (R/NW)	VC6WRR	WRR Weight for VC6. The `PCIE_VC_TXARB2_[n].VC6WRR` bit field indicates the WRR Weight for VC6. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R
15:8 (R/NW)	VC5WRR	WRR Weight for VC5. The `PCIE_VC_TXARB2_[n].VC5WRR` bit field indicates the WRR Weight for VC5. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R

Table 29-223: PCIE_VC_TXARB2_[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VC4WRR	WRR Weight for VC4. The `PCIE_VC_TXARB2_[n].VC4WRR` bit field indicates the WRR Weight for VC4. The access attributes of this field are as follows: • Wire: No access • Dbi: R • Dbi2: R

Vendor Specific DLLP Register

The PCIE_VEND_DLLP_[n] register is used to send a specific PCI Express DLLP.

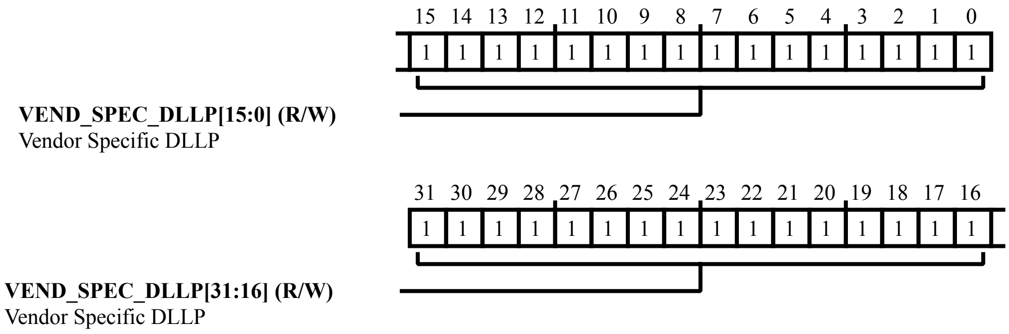

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
VEND_SPEC_DLLP[15:0] (R/W)
Vendor Specific DLLP
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
VEND_SPEC_DLLP[31:16] (R/W)
Vendor Specific DLLP

Figure 29-215: PCIE_VEND_DLLP_[n] Register Diagram

Table 29-224: PCIE_VEND_DLLP_[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VEND_SPEC_DLLP	Vendor Specific DLLP. The `PCIE_VEND_DLLP_[n].VEND_SPEC_DLLP` bit field contains a specific PCI Express DLLP. An application writes the 8-bit DLLP type and 24-bits of payload data into this bit field, then sets bit 0 of the Port Link Control Register (PCIE_PLCTL_[n]) to send the DLLP.

30 Two-Wire Interface (TWI)

The processor has a two-wire interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

The TWI module offers the capabilities of simultaneous master and slave operation and support for both 7-bit addressing and multimedia data arbitration. The TWI interface uses two pins for transferring clock (`TWI_SCL`) and data (`TWI_SDA`) and supports the protocol at speeds up to 400K bits/sec. The TWI interface pins are compatible with 5-V logic levels.

To preserve processor bandwidth, the TWI module can be set up with transfer-initiated interrupts to only service FIFO buffer data reads and writes. Protocol-related interrupts are optional. The TWI externally moves 8-bit data while maintaining compliance with the I^2C bus protocol.

TWI Features

The TWI is fully compatible with the widely used I^2C bus standard.

The TWI controller includes the following features.

- Simultaneous master and slave operation on multiple device systems
- Support for multi-master bus arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support
- Master clock synchronization and support for clock low extension
- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate
- Individual override control of data and clock lines in the event of bus lock-up
- Input filter for spike suppression

- Serial camera control bus support as specified in the *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*

TWI Functional Description

The TWI interface is a shift register that serially transmits and receives data bits. It moves data 1 bit at a time at the SCL rate, to and from other TWI devices. The SCL signal synchronizes the shifting and sampling of the data on the serial data pin.

ADSP-SC58x TWI Register List

The Two-Wire Interface controller TWI allows a device to interface to an inter-IC bus as specified by the Philips I^2C Bus Specification version 2.1, dated January 2000. A set of registers governs TWI operations. For more information on TWI functionality, see the TWI register descriptions.

Table 30-1: ADSP-SC58x TWI Register List

Name	Description
`TWI_CLKDIV`	SCL Clock Divider Register
`TWI_CTL`	Control Register
`TWI_FIFOCTL`	FIFO Control Register
`TWI_FIFOSTAT`	FIFO Status Register
`TWI_IMSK`	Interrupt Mask Register
`TWI_ISTAT`	Interrupt Status Register
`TWI_MSTRADDR`	Master Mode Address Register
`TWI_MSTRCTL`	Master Mode Control Registers
`TWI_MSTRSTAT`	Master Mode Status Register
`TWI_RXDATA16`	Rx Data Double-Byte Register
`TWI_RXDATA8`	Rx Data Single-Byte Register
`TWI_SLVADDR`	Slave Mode Address Register
`TWI_SLVCTL`	Slave Mode Control Register
`TWI_SLVSTAT`	Slave Mode Status Register
`TWI_TXDATA16`	Tx Data Double-Byte Register
`TWI_TXDATA8`	Tx Data Single-Byte Register

ADSP-SC58x TWI Interrupt List

Table 30-2: ADSP-SC58x TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
123	TWI0_DATA	TWI0 Data Interrupt	Level	
124	TWI1_DATA	TWI1 Data Interrupt	Level	
125	TWI2_DATA	TWI2 Data Interrupt	Level	

TWI Block Diagram

The *TWI Block Diagram* figure shows the basic blocks of the TWI interface.

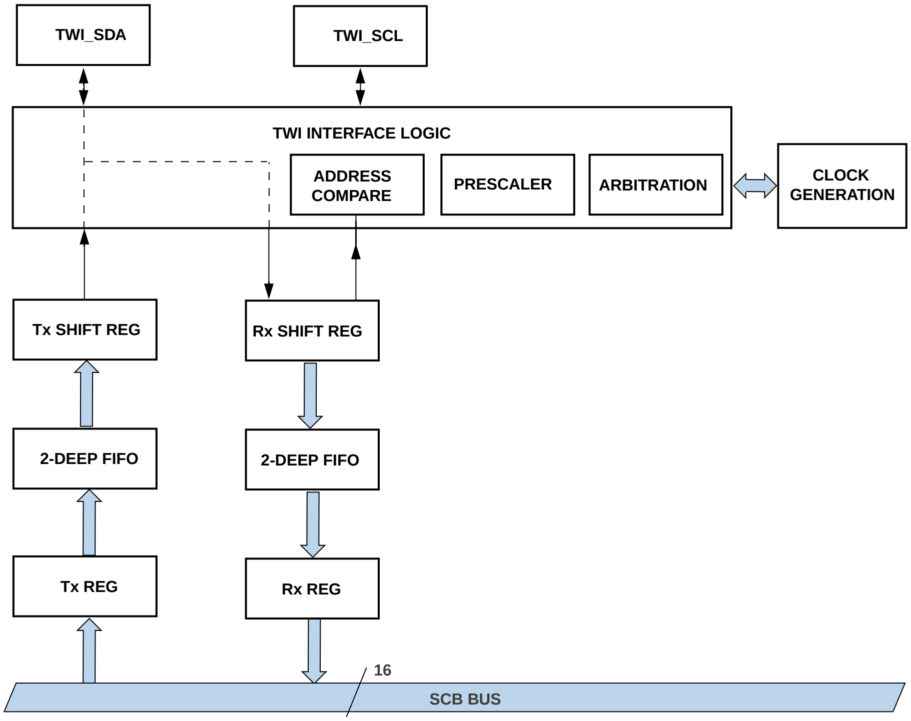

TWI_SDA
TWI_SCL
TWI INTERFACE LOGIC
ADDRESS COMPARE
PRESCALER
ARBITRATION
CLOCK GENERATION
Tx SHIFT REG
Rx SHIFT REG
2-DEEP FIFO
2-DEEP FIFO
Tx REG
Rx REG
16
SCB BUS

Figure 30-1: TWI Block Diagram

External Interface

The `TWI_SDA` (serial data) and `TWI_SCL` (serial clock) signals are open drain and require pull-up resistors. These bidirectional signals externally interface the TWI controller to the I^2C bus and no other external connections or logic are necessary.

Serial Clock Signal (SCL)

The serial clock signal (`TWI_SCL`) is an input in slave mode. In master mode, the TWI controller must set this signal to the desired frequency.

The TWI controller supports the standard mode of operation (up to 100 kHz) or fast mode (up to 400 kHz). The TWI control register (`TWI_CTL`) sets the `TWI_CTL.PRESCALE` value which sets the relationship between the system clock (SCLK0_0) and the internally timed events of the TWI controller. The internal time reference is derived from SCLK0_0 using a prescaled value. The prescale value is the number of SCLK0_0 periods used in the generation of one internal time reference. Set the value of prescale to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value as follows.

$\text{PRESCALE} = f_{\text{SCLK0\_0}}/10\text{MHz}$

NOTE: It is not always possible to achieve 10-MHz accuracy. In such cases, it is safe to round up the PRESCALE value to the next highest integer. For example, if SCLK0_0 is 100 MHz, the PRESCALE value is calculated as 100 MHz/10 MHz = 10. A prescale value of 14 in this case ensures that all timing requirements are met.

During master mode operation, the TWI module uses the `TWI_CLKDIV` register values to create the minimum `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` durations of the `TWI_SCL` signal. The `TWI_CLKDIV.CLKHI` field specifies the minimum number of 10-MHz time reference periods the `TWI_SCL` waits before a new clock low period begins, assuming a single master. (The 10-MHz time reference periods are represented as an 8-bit binary value). The TWI uses the `TWI_CLKDIV.CLKLO` field to specify the minimum number of internal time reference periods (represented as an 8-bit binary value). The `TWI_SCL` signal is held low.

Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the clock generated is 1/10 MHz or 100 ns. The following equation describes the frequency.

`TWI_CLKDIV` = `TWI_SCL` period/10 MHz time reference.

For example, for an `TWI_SCL` of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns), the following equation applies:

`TWI_CLKDIV` = 2500 ns/100 ns = 25

Therefore, a `TWI_SCL` signal with a 30% duty cycle has `TWI_CLKDIV.CLKLO`=17 and `TWI_CLKDIV.CLKHI`=8. Adding `TWI_CLKDIV.CLKLO` and `TWI_CLKDIV.CLKHI` equals `TWI_CLKDIV`.

NOTE: The `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` fields are not intended to guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for the `TWI_SCL` signal. Slew rate controls falling edges. The *RC* time constant governs the rising edges. The pull-up resistor and the `TWI_SCL` capacitance form the time constant. See the “Register Descriptions” section for more details.

Serial Data Signal (SDA)

The TWI transmits and receives serial data, depending on the direction of the transfer, on the bidirectional serial data signal (SDA).

Internal Interface

The peripheral bus interface supports the transfer of 16-bit wide data. The processor uses the interface in the support of register and FIFO buffer reads and writes. The TWI internal interface is comprised of the blocks described as follows.

Register block. Contains all control and status bits and reflects what can be written or read as outlined by the programming model. Each function block updates their corresponding status bits.

FIFO buffer. Configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

Transmit shift register. Serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgments or it can be manually overwritten.

Receive shift register. Receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

Address compare block. Supports address comparison in the event the TWI controller module is accessed as a slave.

Prescaler block. Must be programmed to generate a 10-MHz time reference relative to the system clock. The block uses this time base for filtering of data and timing events specified by the electrical data sheet (See the Philips specification). The block uses the time base to generate the `TWI_SCL` clock as well.

Clock generation module. Generates an external `TWI_SCL` clock when in master mode. It includes the logic necessary for synchronization in a multi-master clock configuration and clock stretching when configured in slave mode.

NOTE: The TWI does not support DMA based operation.

TWI Architectural Concepts

The TWI controller follows the transfer protocol of the Philips I^{2}C Bus specification version 2.1 dated January 2000.

NOTE: The TWI unit does not support DMA-based operation.

TWI Protocol

The ***Data Transfer*** figure shows a simple complete transfer.

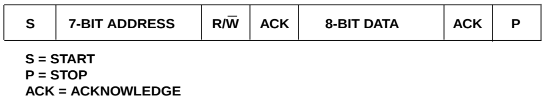


Figure 30-2: Data Transfer

The TWI controller register contents maps to a basic transfer. The ***Data Transfer with Bit Illustration*** figure details the same transfer from the ***Data Transfer*** figure noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits 1 byte of data. The slave has acknowledged both address and data.

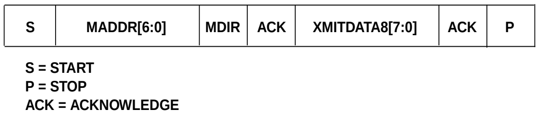


Figure 30-3: Data Transfer with Bit Illustration

Clock Generation and Synchronization

The TWI controller implementation only issues a clock during master mode operation and only at the time a transfer initiates. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. The *Clock Synchronization* figure shows this functionality.

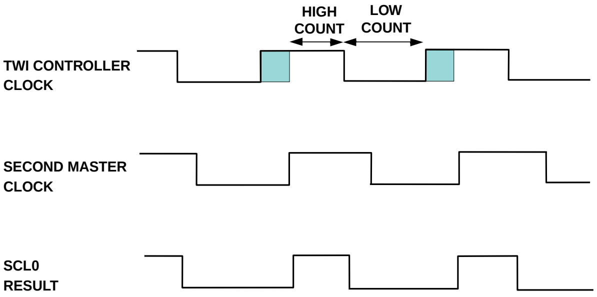


Figure 30-4: Clock Synchronization

The TWI controller serial clock (`TWI_SCL`) output follows these rules:

- Once the clock high (`TWI_CLKDIV.CLKHI`) count is complete, the serial clock output is driven low and the clock low (`TWI_CLKDIV.CLKLO`) count begins.
- Once the clock low count is complete, the serial clock line is three-stated. This state allows the external pull-up resistor to pull the `TWI_SCL` signal high. The clock synchronization logic enters into a delay mode (shaded area) until the `TWI_SCL` signal is detected at logic 1 level. Now, the clock high count begins.

Bus Arbitration

The TWI controller initiates a master mode transmission only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. The ***Bus Arbitration*** figure shows the arbitration.

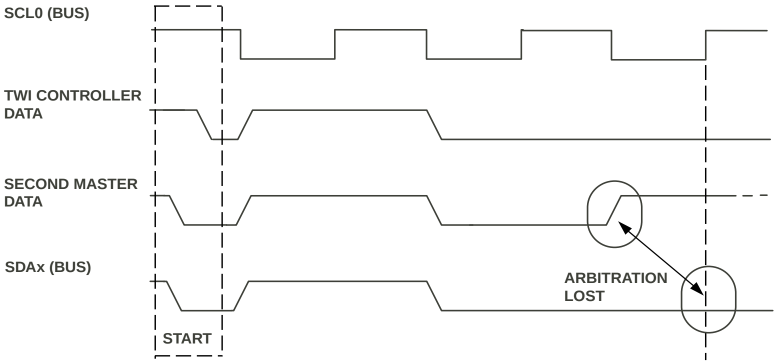


Figure 30-5: Bus Arbitration

The TWI controller monitors the serial data bus (SDA) while the `TWI_SCL` signal is high. If the `TWI_SDA` signal is determined to be an active logic 0 level while the data of the TWI controller is a logic 1 level, the TWI controller has lost arbitration. It stops generating the clock and data signals. Arbitration is not only performed at the serial clock edges, but also during the entire time the `TWI_SCL` signal is high.

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is a logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, except repeated start combined transfers. The ***Start and Stop Conditions*** figure shows the transitions.

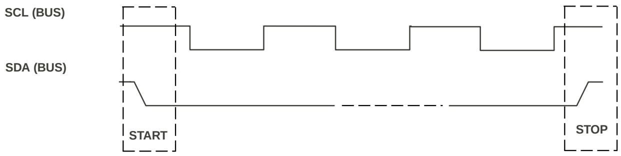


Figure 30-6: Start and Stop Conditions

The TWI special case start and stop conditions of the TWI controller include the following.

- Controller addressed as a slave-receiver. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_ISTAT.SCOMP`).
- Controller addressed as a slave-transmitter. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (`TWI_ISTAT.SCOMP`) and indicates a slave transfer error (`TWI_ISTAT.SERR`).
- Controller as a master-transmitter or master-receiver. If the stop bit (`TWI_MSTRCTL.STOP`) is set during an active master transfer, the TWI controller issues a stop condition as soon as possible avoiding any error conditions. The TWI controller operates as if data transfer count had been reached.

General Call Support

The TWI controller always decodes and acknowledges a general call address if:

- The TWI controller is enabled as a slave
- General call is enabled

The `TWI_SLVCTL.GEN` bit configures general call addressing (0x00) only when the TWI controller is a slave-receiver.

If the data associated with the transfer is (NAK) not acknowledged, the `TWI_SLVCTL.NAK` bit can be set. If the TWI controller issues a general call as a master-transmitter, set the appropriate address (`TWI_MSTRADDR` register) and transfer direction (`TWI_MSTRCTL.DIR` bit) and load the transmit FIFO data.

NOTE: The byte following the general call address usually defines the slaves response to the call. The interpretation of the command in the second byte is based on the value of its LSB. For a TWI slave device, the bytes received after the general call address are considered data.

Fast Mode

Fast mode essentially uses the same mechanics as the standard mode of operation. Fast mode affects electrical specifications and timing. When fast mode is enabled, (FAST) timing is modified to meet the following electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition set-up time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

TWI Operating Modes

The TWI has two modes of operation: repeated start and clock stretching. The following sections describe the operating modes.

Repeated Start

A repeated start condition is the absence of a stop condition between two transfers. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. The following sections guide the programmer in developing a service routine.

Transmit Receive Repeated Start

The *Repeated Start Followed by Data Receive* figure shows a repeated start followed by a data receive sequence. The shading in the figure indicates that the slave has control of the bus.

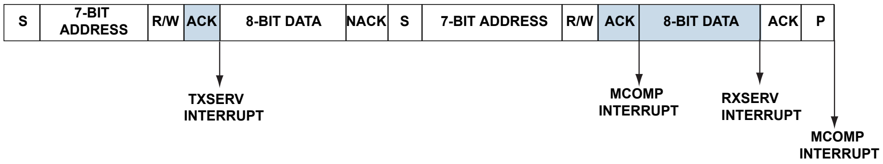


Figure 30-7: Repeated Start Followed by Data Receive

The tasks performed at each interrupt are:

- Transmit FIFO service (`TWI_ISTAT.TXSERV`) interrupt request. This interrupt is generated due to a FIFO access. Since this byte is the last of this transfer, the TWI uses the `TWI_FIFOSTAT` register to indicate that the transmit FIFO is empty. When read, `TWI_MSTRCTL.DCNT` bit field=0. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and set the `TWI_MSTRCTL.DIR` bit if the following transfer is a data receive.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt request is generated when all data transfers (`TWI_MSTRCTL.DCNT` bit field=0). If no errors occur, a start condition initiates. Clear the `TWI_MSTRCTL.RSTART` bit and program the `TWI_MSTRCTL.DCNT` bits with the desired number of bytes to receive.

- Receive FIFO service (`TWI_ISTAT.RXSERV`) interrupt. This interrupt request is generated due to the arrival of a byte in the receive FIFO. Simple data handling is the only requirement.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt request is generated when the transfer completes.

Receive Transmit Repeated Start

The ***Repeated Start Data Receive Followed by Data Transmit*** figure illustrates a repeated start data receive followed by a data transmit sequence. The shading in the figure indicates that the slave has control of the bus.

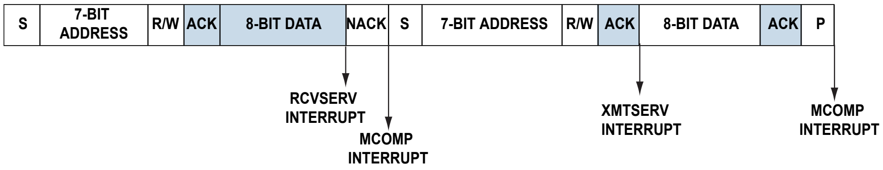


Figure 30-8: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (`TWI_ISTAT.RXSERV`) interrupt. This interrupt request is generated due to the arrival of a data byte in the receive FIFO. Set the `TWI_MSTRCTL.RSTART` bit to indicate a repeated start and clear the `TWI_MSTRCTL.DIR` bit if the following transfer is a data transmit.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt request has occurred due to the completion of the data receive transfer. If no errors occur, a start condition initiates. Clear the `TWI_MSTRCTL.RSTART` bit and program the `TWI_MSTRCTL.DCNT` bits with the desired number of bytes to transmit.
- Transmit FIFO service (`TWI_ISTAT.TXSERV`) interrupt. This interrupt request is generated due to a FIFO access. Simple data handling is the only requirement.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt request is generated when the transfer completes.

NOTE: There is no timing constraint to meet the conditions—program the bits as required. Refer to Clock Stretching During Repeated Start section for more on how the controller stretches the clock during repeated start transfers.

Clock Stretching

Clock stretching is an added function of the TWI controller in master mode operation. This behavior uses self-induced stretching of the I^2C clock while waiting to service interrupts. Hardware initiates stretching automatically. No programming is necessary. The TWI controller as a master supports three modes of clock stretching:

- Clock Stretching During FIFO Underflow
- Clock Stretching During FIFO Overflow

- Clock Stretching During Repeated Start

Clock Stretching During FIFO Underflow

During a master mode transmit, an interrupt request occurs the instant the transmit FIFO becomes empty. The most recent byte begins transmission. If the `TWI_ISTAT.TXSERV` interrupt request is not serviced, the concluding acknowledge phase of the transfer stretches.

Stretching of the clock continues until new data bytes are written to the transmit FIFO (`TWI_TXDATA8` or `TWI_TXDATA16` registers). No other action is required to release the clock and continue the transmission. This behavior continues until the transmission completes (`TWI_MSTRCTL.DCNT`=0). The transmission concludes (`TWI_ISTAT.MCOMP`). The ***Clock Stretching during FIFO Underflow*** figure and table show the stretching.

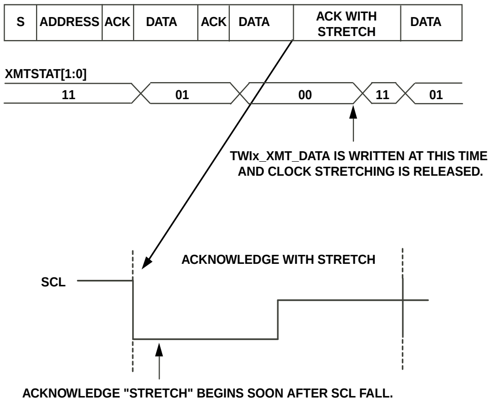

S
ADDRESS
ACK
DATA
ACK
DATA
ACK WITH STRETCH
DATA
XMTSTAT[1:0]
11
01
00
11
01
TWIx_XMT_DATA IS WRITTEN AT THIS TIME AND CLOCK STRETCHING IS RELEASED.
ACKNOWLEDGE WITH STRETCH
SCL
ACKNOWLEDGE "STRETCH" BEGINS SOON AFTER SCL FALL.

Figure 30-9: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear the interrupt request source bits.

Clock Stretching During FIFO Overflow

During a master mode receive operation, an interrupt occurs at the instant the receive FIFO becomes full. It is during the acknowledge phase of this received byte that clock stretching begins. The TWI module makes no attempt to initiate the reception of another byte. Stretching of the clock continues until the data bytes previously received are read from the receive FIFO buffer (`TWI_RXDATA8` or `TWI_RXDATA16` registers). No other action is required to release the clock and continue the reception of data. This behavior continues until the reception is complete (`TWI_MSTRCTL.DCNT`=0). Reception concludes (`TWI_ISTAT.MCOMP`). The ***Clock Stretching During FIFO Overflow*** figure and table show the clock stretching.

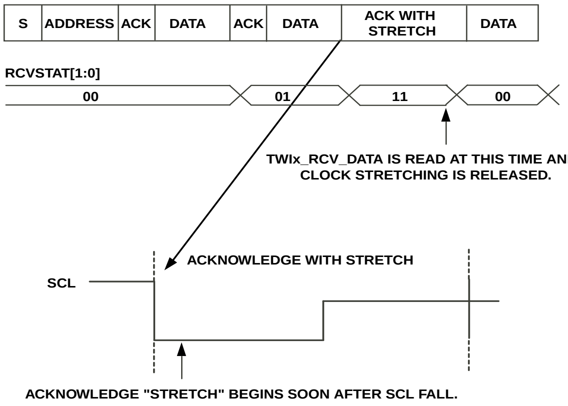

S
ADDRESS
ACK
DATA
ACK
DATA
ACK WITH STRETCH
DATA
RCVSTAT[1:0]
00
01
11
00
TWIx_RCV_DATA IS READ AT THIS TIME AND CLOCK STRETCHING IS RELEASED.
ACKNOWLEDGE WITH STRETCH
SCL
ACKNOWLEDGE "STRETCH" BEGINS SOON AFTER SCL FALL.

Figure 30-10: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Acknowledge: Clear the interrupt source bits.	Interrupt: MCOMP – Master receive complete.

Clock Stretching During Repeated Start

The repeated start feature in I^2C protocol requires a transition between two subsequent transfers. With the use of clock stretching, the task of managing transitions becomes simpler and common to all transfer types.

Once an initial TWI master transfer completes (transmit or receive), the clock initiates a stretch during the repeated start phase between transfers. Concurrent with this event, the initial transfer generates a `TWI_ISTAT.MCOMP` interrupt to signify the initial transfer has completed (`TWI_MSTRCTL.DCNT`=0). This initial transfer is handled without any special bit setting sequences or timing.

The clock stretching logic described applies here. With no system-related timing constraints, the subsequent transfer (receive or transmit) is set up and activated. This sequence can repeat as many times as required to string a series of repeated start transfers together. The ***Clock Stretching during Repeated Start Condition*** figure and table show the clock stretching.

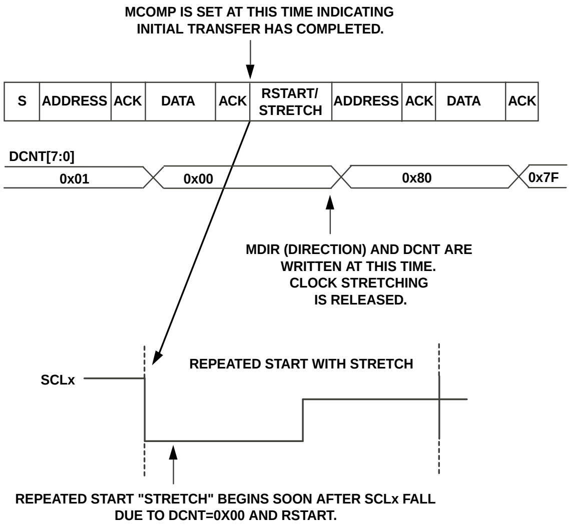

MCOMP IS SET AT THIS TIME INDICATING INITIAL TRANSFER HAS COMPLETED.
S
ADDRESS
ACK
DATA
ACK
RSTART/ STRETCH
ADDRESS
ACK
DATA
ACK
DCNT[7:0]
0x01
0x00
0x80
0x7F
MDIR (DIRECTION) AND DCNT ARE WRITTEN AT THIS TIME. CLOCK STRETCHING IS RELEASED.
REPEATED START WITH STRETCH
SCLx
REPEATED START "STRETCH" BEGINS SOON AFTER SCLx FALL DUE TO DCNT=0X00 AND RSTART.

Figure 30-11: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear the interrupt request source bits. Write to TWIx_MASTER_CTL, setting MDIR (receive), clearing RSTART, and setting new DCNT value (nonzero).
Interrupt: RCVSERV – Receive FIFO is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear the interrupt request source bits.

TWI Programming Model

The topics in this section provide information on the basic programming steps required to set up and run the two wire interface.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operations.

Perform general setup before setting either the master or slave enable bits.

1. Program the `TWI_CTL.EN` bit to enable the TWI controller and set the prescale value (`TWI_CTL.PRESCALE` bit).
2. Program the prescale value to the binary representation of f_{SCLK0}/10 MHz. Round up all values to the next whole number.
3. Set the `TWI_CTL.EN` bit to enable the controller.

Once the TWI controller is enabled, a bus busy condition can be detected. This condition clears after t_{BUF} has expired, assuming no additional bus activity has been detected.

Slave Mode

When enabled, slave mode operation supports both receive and transmit data transfers.

It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. The following setup reflects this functionality.

1. Program the `TWI_SLVADDR` register. The TWI uses the appropriate 7 bits in determining a match during the address phase of the transfer.
2. Program the `TWI_TXDATA8.VALUE` or `TWI_TXDATA16` registers. These values are the initial data values for transmission when the slave is addressed and transmission is needed. This step is optional. If no data is written when the slave is addressed and transmission is needed, the serial clock (`TWI_SCL`) stretches. An interrupt is generated until data is written to the transmit FIFO.
3. Program the `TWI_IMSK` register. There are enable-bits associated with the desired interrupt sources. For example, programming the value 0x000F results in an interrupt request output to the processor, when the TWI module detects a valid address match. An interrupt request also occurs when a valid slave transfer completes or has an error, or a subsequent transfer has begun and the previous transfer has not been serviced.
4. Program the `TWI_SLVCTL` register. This step prepares and enables slave mode operation. For example, programming the value 0x0005 enables slave mode operation and requires 7-bit addressing. It indicates that data in the transmit FIFO buffer is for slave mode transmission.

The ***Slave Mode Interaction*** table and ***TWI Slave Mode Program Flow*** diagram represent the interaction between the TWI controller and the processor using this example.

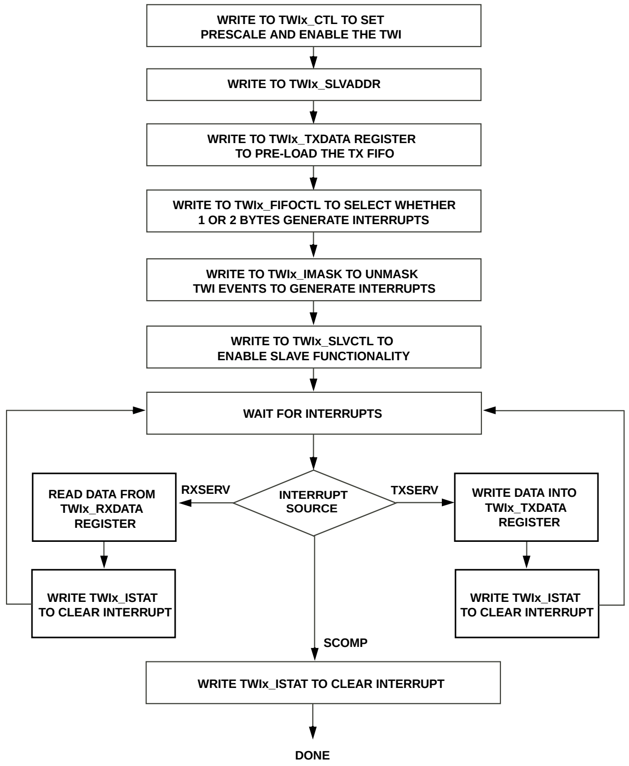

WRITE TO TWIx_CTL TO SET PRESCALE AND ENABLE THE TWI
WRITE TO TWIx_SLVADDR
WRITE TO TWIx_TXDATA REGISTER TO PRE-LOAD THE TX FIFO
WRITE TO TWIx_FIFOCTL TO SELECT WHETHER 1 OR 2 BYTES GENERATE INTERRUPTS
WRITE TO TWIx_IMASK TO UNMASK TWI EVENTS TO GENERATE INTERRUPTS
WRITE TO TWIx_SLVCTL TO ENABLE SLAVE FUNCTIONALITY
WAIT FOR INTERRUPTS
INTERRUPT SOURCE
RXSERV
TXSERV
READ DATA FROM TWIx_RXDATA REGISTER
WRITE DATA INTO TWIx_TXDATA REGISTER
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
SCOMP
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
DONE

Figure 30-12: TWI Slave Mode Program Flow

Table 30-3: Slave Mode Interaction

TWI Controller	Processor
Interrupt: SINIT – Slave transfer in progress.	Acknowledge: Clear the interrupt source bits.
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read TWIx_FIFO_STAT. Read the receive FIFO buffer.
...	...
Interrupt: SCOMP – Slave transfer complete.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

Master Mode Program Flow

The *Master Mode Program Flow* figure shows the program for the TWI in master mode.

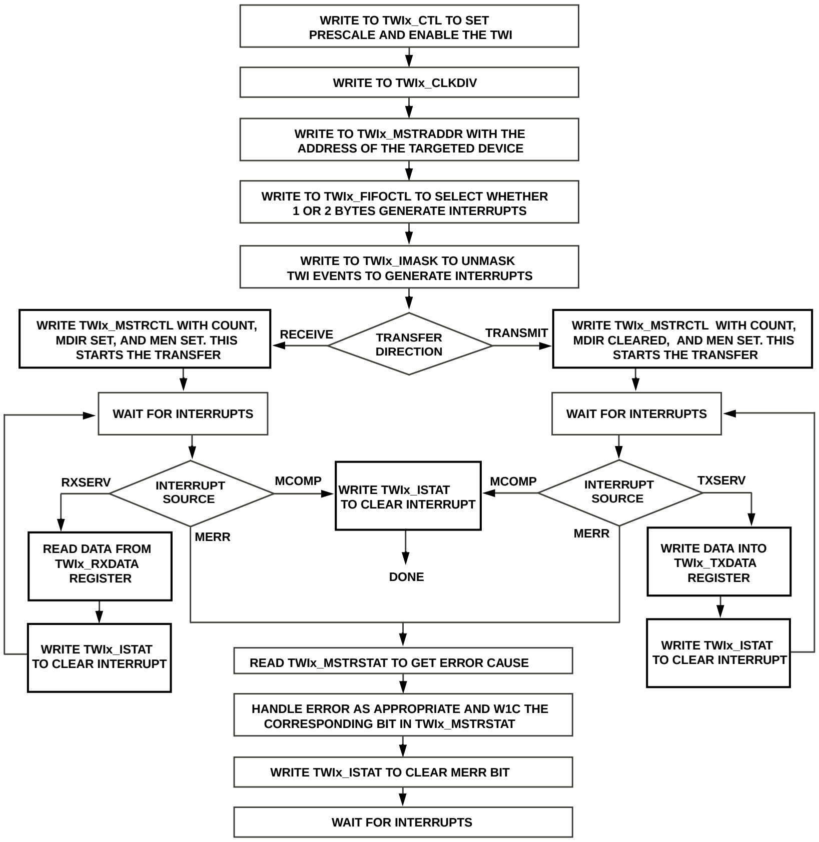

WRITE TO TWIx_CTL TO SET PRESCALE AND ENABLE THE TWI
WRITE TO TWIx_CLKDIV
WRITE TO TWIx_MSTRADDR WITH THE ADDRESS OF THE TARGETED DEVICE
WRITE TO TWIx_FIFOCTL TO SELECT WHETHER 1 OR 2 BYTES GENERATE INTERRUPTS
WRITE TO TWIx_IMASK TO UNMASK TWI EVENTS TO GENERATE INTERRUPTS
WRITE TWIx_MSTRCTL WITH COUNT, MDIR SET, AND MEN SET. THIS STARTS THE TRANSFER
RECEIVE
TRANSFER DIRECTION
TRANSMIT
WRITE TWIx_MSTRCTL WITH COUNT, MDIR CLEARED, AND MEN SET. THIS STARTS THE TRANSFER
WAIT FOR INTERRUPTS
WAIT FOR INTERRUPTS
RXSERV
INTERRUPT SOURCE
MCOMP
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
MCOMP
INTERRUPT SOURCE
TXSERV
READ DATA FROM TWIx_RXDATA REGISTER
MERR
DONE
MERR
WRITE DATA INTO TWIx_TXDATA REGISTER
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
READ TWIx_MSTRSTAT TO GET ERROR CAUSE
WRITE TWIx_ISTAT TO CLEAR INTERRUPT
HANDLE ERROR AS APPROPRIATE AND W1C THE CORRESPONDING BIT IN TWIx_MSTRSTAT
WRITE TWIx_ISTAT TO CLEAR MERR BIT
WAIT FOR INTERRUPTS

Figure 30-13: Master Mode Program Flow

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis.

An example of programming steps for a receive and for a transmit is given separately in following sections. The programming step for clock setup listed here is common to both transfer types.

1. Program the `TWI_CLKDIV` register to define the minimum high and minimum low duration for the clock.

The `TWI_CLKDIV.CLKHI` and `TWI_CLKDIV.CLKLO` fields do not guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for `TWI_SCL`. The slew rate controls falling edges. The RC time constant formed by the pull-up resistor and the SCL capacitance govern rising edges. See the “Register Descriptions” section for more details.

Master Mode Transmit

Follow these programming steps for a single master mode transmission:

1. Program the `TWI_MSTRADDR` register. This step defines the address transmitted during the address phase of the transfer.

2. Program the `TWI_TXDATA8` or `TWI_TXDATA16` register. This step configures the initial data transmitted. It is an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.
3. Program the `TWI_FIFOCTL` register. The programming indicates if the transmit FIFO buffer interrupt requests occur with each byte transmitted (8-bits) or with every 2 bytes transmitted (16-bits).
4. Program the `TWI_IMSK` register. This step enables the bits associated with the desired interrupt request sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, and the master transfer has an error.
5. Program the `TWI_MSTRCTL` register. This step prepares and enables master mode operation. For example, programming the value 0x0201: enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

The ***Master Mode Transmit Setup Interaction*** table represents the interaction between the TWI controller and the processor using this example.

Table 30-4: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits.

Master Mode Receive

Follow these programming steps for a single master mode receive.

1. Program the `TWI_MSTRADDR` register. This step defines the address transmitted during the address phase of the transfer.
2. Program the `TWI_FIFOCTL` register. This step indicates if the receive FIFO buffer interrupt requests occur with each byte received (8-bits) or with every 2 bytes received (16-bits).
3. Program the `TWI_IMSK` register. This step configures the enable bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt request output to the processor when the master transfer completes, and the master transfer has an error.
4. Program the `TWI_MSTRCTL` register. This step prepares and enables master mode operation. For example, programming the value 0x0205: enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

The ***Master Mode Receive Setup Interaction*** table shows the interaction between the TWI controller and the processor using this example.

Table 30-5: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.

NOTE: After the `TWI_MSTRCTL.DCNT` bit decrements to zero, the TWI master device sends a NAK to indicate to the slave transmitter to release the bus. This operation allows the master to send the stop signal to terminate the transfer.

ADSP-SC58x TWI Register Descriptions

Two-Wire Interface (TWI) contains the following registers.

Table 30-6: ADSP-SC58x TWI Register List

Name	Description
`TWI_CLKDIV`	SCL Clock Divider Register
`TWI_CTL`	Control Register
`TWI_FIFOCTL`	FIFO Control Register
`TWI_FIFOSTAT`	FIFO Status Register
`TWI_IMSK`	Interrupt Mask Register
`TWI_ISTAT`	Interrupt Status Register
`TWI_MSTRADDR`	Master Mode Address Register
`TWI_MSTRCTL`	Master Mode Control Registers
`TWI_MSTRSTAT`	Master Mode Status Register
`TWI_RXDATA16`	Rx Data Double-Byte Register
`TWI_RXDATA8`	Rx Data Single-Byte Register
`TWI_SLVADDR`	Slave Mode Address Register
`TWI_SLVCTL`	Slave Mode Control Register
`TWI_SLVSTAT`	Slave Mode Status Register
`TWI_TXDATA16`	Tx Data Double-Byte Register
`TWI_TXDATA8`	Tx Data Single-Byte Register

SCL Clock Divider Register

During master mode operation, the `TWI_CLKDIV` holds values, which the TWI uses to create the high and low durations of the serial clock (SCL). The clock signal SCL is an output in master mode and an input in slave mode. The values in the `TWI_CLKDIV.CLKLO` and `TWI_CLKDIV.CLKHI` fields add up to the CLKDIV value the following equation.

CLKDIV = TWI SCL period / 10 MHz time reference

Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns. For example, for an SCL of 400 KHz (period = 1/400 KHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

CLKDIV = 2500 ns / 100 ns = 25

For an SCL with a 30% duty cycle, use `TWI_CLKDIV.CLKLO` = 17 and `TWI_CLKDIV.CLKHI` = 8.

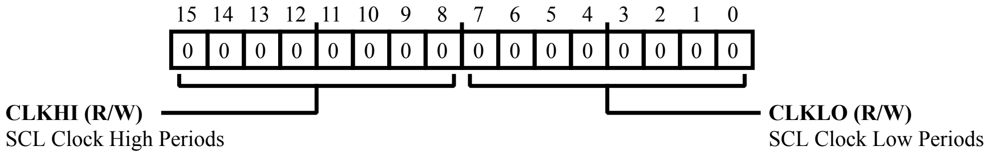


Figure 30-14: TWI_CLKDIV Register Diagram

Table 30-7: TWI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	CLKHI	SCL Clock High Periods. The `TWI_CLKDIV.CLKHI` specifies the number of 10 MHz time reference periods the serial clock (SCL) waits before a new clock low period begins, assuming a single master.
7:0 (R/W)	CLKLO	SCL Clock Low Periods. The `TWI_CLKDIV.CLKLO` specifies the number of internal time reference periods the serial clock (SCL) is held low.

Control Register

The `TWI_CTL` enables the TWI, establishes a relationship between the system clock (SCLK0_0) and the TWI controller's internally timed events, and enables SCCB compatibility.

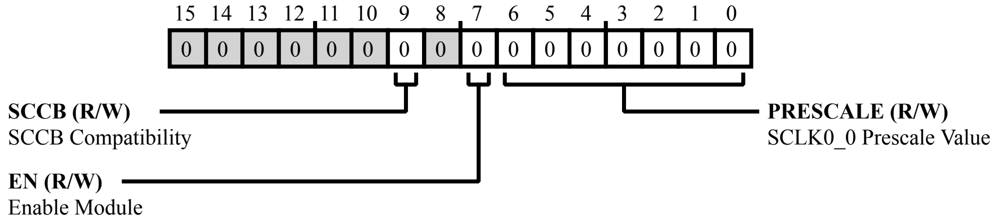


Figure 30-15: TWI_CTL Register Diagram

Table 30-8: TWI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SCCB	SCCB Compatibility. The `TWI_CTL.SCCB` enables SCCB compatible operation for the TWI. SCCB compatibility is an optional feature and should not be used in an I^{2}C bus system. When this feature is enabled, all slave asserted acknowledgement bits are ignored by this master. This feature is valid only during transfers where the TWI is mastering an SCCB bus. Slave mode transfers should be avoided when this feature is enabled because the TWI controller always generates an acknowledge in slave mode.	
		0	Disable SCCB compatibility. When disabled, master transfers are not SCCB compatible.
		1	Enable SCCB compatibility. When enabled, master transfers are SCCB compatible. All slave-asserted acknowledgment bits are ignored by this master.
7 (R/W)	EN	Enable Module. The `TWI_CTL.EN` enables TWI controller operation for either master and/or slave mode of operation. It is recommended that this bit be set at the time `TWI_CTL.PRESCALE` is initialized and remain set. This method guarantees accurate operation of bus busy detection logic.	
		0	Disable
		1	Enable

Table 30-8: TWI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	PRESCALE	SCLK0_0 Prescale Value. The `TWI_CTL.PRESCALE` holds the pre-scaled value for the TWI internal time reference. This reference is derived from SCLK0_0 according to the formula: `TWI_CTL.PRESCALE` = f_{SCLK0_0}/10MHz The `TWI_CTL.PRESCALE` specifies the number of system clock (SCLK0_0) periods used in the generation of one internal time reference. The value of `TWI_CTL.PRESCALE` must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value.

FIFO Control Register

The `TWI_FIFOCTL` control bits affect only the FIFO and are not tied in any way with master or slave mode operation.

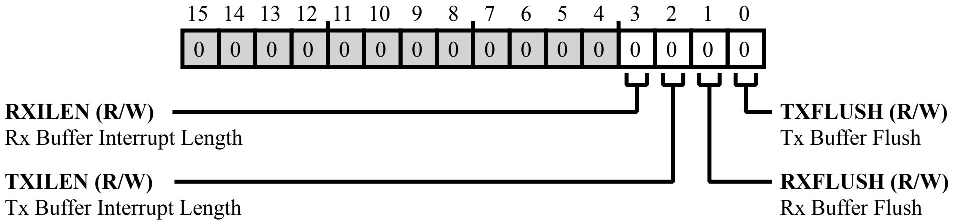

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXILEN (R/W)
Rx Buffer Interrupt Length
TXILEN (R/W)
Tx Buffer Interrupt Length
TXFLUSH (R/W)
Tx Buffer Flush
RXFLUSH (R/W)
Rx Buffer Flush

Figure 30-16: TWI_FIFOCTL Register Diagram

Table 30-9: TWI_FIFOCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RXILEN	Rx Buffer Interrupt Length. The `TWI_FIFOCTL.RXILEN` determines the rate at which receive buffer interrupts are to be generated. Interrupts may be generated with each byte received or after two bytes are received. Interrupt status is available in `TWI_FIFOSTAT.RXSTAT`.	
		0	RXSERVI on 1 or 2 Bytes in FIFO
		1	RXSERVI on 2 Bytes in FIFO
2 (R/W)	TXILEN	Tx Buffer Interrupt Length. The `TWI_FIFOCTL.TXILEN` determines the rate at which transmit buffer interrupts are to be generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. Interrupt status is available in `TWI_FIFOSTAT.TXSTAT`.	
		0	TXSERVI on 1 Byte of FIFO Empty
		1	TXSERVI on 2 Bytes of FIFO Empty
1 (R/W)	RXFLUSH	Rx Buffer Flush. The `TWI_FIFOCTL.RXFLUSH` directs the TWI to flush the contents of the receive buffer and update `TWI_FIFOSTAT.RXSTAT` to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive, the receive buffer in this state responds to the receive logic as if it is full.	
		0	Normal Operation of Rx Buffer
		1	Flush Rx Buffer

Table 30-9: TWI_FIFOCTL Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">0
(R/W)</td><td rowspan="3">TXFLUSH</td><td colspan="2">Tx Buffer Flush.
The TWI_FIFOCTL.TXFLUSH directs the TWI to flush the contents of the transmit buffer and update TWI_FIFOSTAT.TXSTAT to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds to the transmit logic as if it is empty.</td></tr>
<tr><td>0</td><td>Normal Operation of Tx Buffer</td></tr>
<tr><td>1</td><td>Flush Tx Buffer</td></tr>
</table>

FIFO Status Register

The `TWI_FIFOSTAT` fields indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

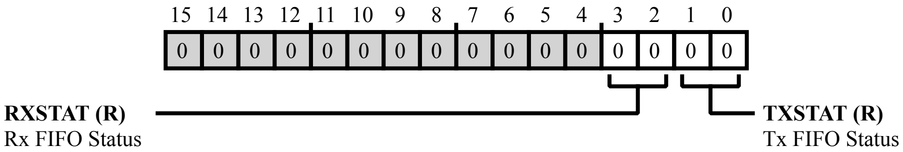


Figure 30-17: TWI_FIFOSTAT Register Diagram

Table 30-10: TWI_FIFOSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/NW)	RXSTAT	Rx FIFO Status. The read-only `TWI_FIFOSTAT.RXSTAT` indicates the number of valid data bytes in the receive FIFO buffer. The status is updated with each FIFO buffer read using the peripheral data bus or write access by the receive shift register. Simultaneous accesses are allowed.	
		0	Empty. The FIFO is empty.
		1	Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral read of the FIFO is allowed.
		2	Reserved
		3	Full. The FIFO is full and contains two bytes of data. Either a single or double byte peripheral read of the FIFO is allowed.
1:0 (R/NW)	TXSTAT	Tx FIFO Status. The read-only `TWI_FIFOSTAT.TXSTAT` field indicates the number of valid data bytes in the FIFO buffer. The status is updated with each FIFO buffer write using the peripheral data bus or read access by the transmit shift register. Simultaneous accesses are allowed.	
		0	Empty. The FIFO is empty. Either a single or double byte peripheral write of the FIFO is allowed.
		1	Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral write of the FIFO is allowed.
		2	Reserved
		3	Full. The FIFO is full and contains two bytes of data.

Interrupt Mask Register

The TWI_IMSK enables interrupt sources to assert the interrupt output. Each mask bit corresponds with one interrupt request source bit in TWI_ISTAT. Reading and writing TWI_IMSK does not affect the contents of the TWI_ISTAT.

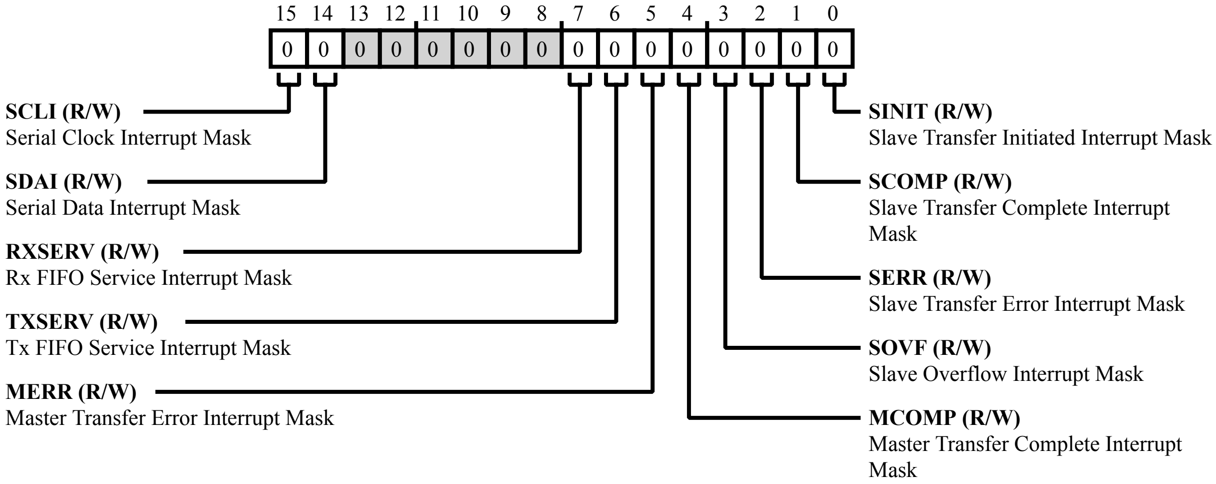

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCLI (R/W)
Serial Clock Interrupt Mask
SDAI (R/W)
Serial Data Interrupt Mask
RXSERV (R/W)
Rx FIFO Service Interrupt Mask
TXSERV (R/W)
Tx FIFO Service Interrupt Mask
MERR (R/W)
Master Transfer Error Interrupt Mask
SINIT (R/W)
Slave Transfer Initiated Interrupt Mask
SCOMP (R/W)
Slave Transfer Complete Interrupt Mask
SERR (R/W)
Slave Transfer Error Interrupt Mask
SOVF (R/W)
Slave Overflow Interrupt Mask
MCOMP (R/W)
Master Transfer Complete Interrupt Mask

Figure 30-18: TWI_IMSK Register Diagram

Table 30-11: TWI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLI	Serial Clock Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
14 (R/W)	SDAI	Serial Data Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
7 (R/W)	RXSERV	Rx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
6 (R/W)	TXSERV	Tx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Table 30-11: TWI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	MERR	Master Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
4 (R/W)	MCOMP	Master Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
3 (R/W)	SOVF	Slave Overflow Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
2 (R/W)	SERR	Slave Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
1 (R/W)	SCOMP	Slave Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
0 (R/W)	SINIT	Slave Transfer Initiated Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Interrupt Status Register

The `TWI_ISTAT` contains information about functional areas requiring servicing. Many of the bits serve as an indicator to further read and service various status registers. After servicing the interrupt source associated with a bit, the user must clear that interrupt source bit by writing a 1 to it.

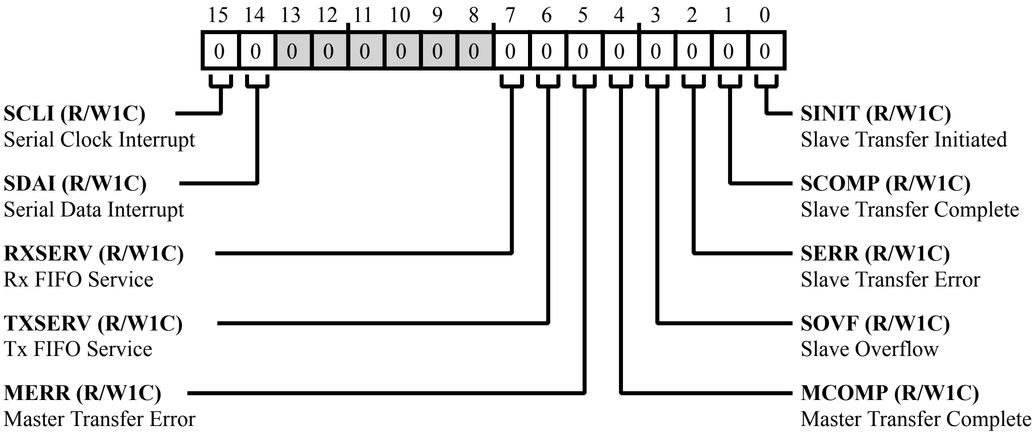

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCLI (R/W1C)
Serial Clock Interrupt
SDAI (R/W1C)
Serial Data Interrupt
RXSERV (R/W1C)
Rx FIFO Service
TXSERV (R/W1C)
Tx FIFO Service
MERR (R/W1C)
Master Transfer Error
SINIT (R/W1C)
Slave Transfer Initiated
SCOMP (R/W1C)
Slave Transfer Complete
SERR (R/W1C)
Slave Transfer Error
SOVF (R/W1C)
Slave Overflow
MCOMP (R/W1C)
Master Transfer Complete

Figure 30-19: TWI_ISTAT Register Diagram

Table 30-12: TWI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	SCLI	Serial Clock Interrupt. If the TWI is enabled (`TWI_CTL.EN`), SCLI is set on a high-to-low transition of the serial clock pin (SCLx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt. No transition was detected on the SCLx pin.
		1	Interrupt Detected. A high-to-low transition was detected on the SCLx pin. This bit is W1C.
14 (R/W1C)	SDAI	Serial Data Interrupt. If the TWI is enabled (`TWI_CTL.EN`), SDAI is set on a high-to-low transition of the serial data pin (SDAx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt. No transition was detected on the SDAx pin.
		1	Interrupt Detected. A high-to-low transition was detected on the SDAx pin. This bit is W1C.

Table 30-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RXSERV	Rx FIFO Service. If `TWI_FIFOCTL.RXILEN` =0, the `TWI_ISTAT.RXSERV` is set each time the `TWI_FIFOSTAT.RXSTAT` field is updated to either 01 or 11. If `TWI_FIFOCTL.RXILEN` =1, the `TWI_ISTAT.RXSERV` is set each time `TWI_FIFOSTAT.RXSTAT` is updated to 11.	
		0	No Interrupt. The FIFO does not require servicing, or the `TWI_FIFOSTAT.RXSTAT` field has not changed since this bit was last cleared.
		1	Interrupt Detected. The receive FIFO buffer has one or two 8-bit words of data available to be read.
6 (R/W1C)	TXSERV	Tx FIFO Service. If `TWI_FIFOCTL.TXILEN` =0, the `TWI_ISTAT.TXSERV` is set each time the `TWI_FIFOSTAT.TXSTAT` field is updated to either 01 or 00. If `TWI_FIFOCTL.TXILEN` =1, the `TWI_ISTAT.TXSERV` is set each time `TWI_FIFOSTAT.TXSTAT` is updated to 00.	
		0	No Interrupt. FIFO does not require servicing, or the `TWI_FIFOSTAT.TXSTAT` field has not changed since this bit was last cleared.
		1	Interrupt Detected. The transmit FIFO buffer has one or two 8-bit locations available to be written.
5 (R/W1C)	MERR	Master Transfer Error. The `TWI_ISTAT.MERR` indicates that a master error has occurred. The conditions surrounding the error are indicated by the master status register (`TWI_MSTRSTAT`).	
		0	No Interrupt
		1	Interrupt Detected
4 (R/W1C)	MCOMP	Master Transfer Complete. The `TWI_ISTAT.MCOMP` indicates that the initiated master transfer has completed. In the absence of a repeat start, the bus has been released.	
		0	No Interrupt
		1	Interrupt Detected
3 (R/W1C)	SOVF	Slave Overflow. The `TWI_ISTAT.SOVF` indicates that the `TWI_ISTAT.SCOMP` bit was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another.	
		0	No Interrupt
		1	Interrupt Detected

Table 30-12: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	SERR	Slave Transfer Error. The `TWI_ISTAT.SERR` indicates that a slave error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.	
		0	No Interrupt
		1	Interrupt Detected
1 (R/W1C)	SCOMP	Slave Transfer Complete. The `TWI_ISTAT.SCOMP` indicates that the transfer is complete and either a stop, or a restart was detected.	
		0	No Interrupt
		1	Interrupt Detected
0 (R/W1C)	SINIT	Slave Transfer Initiated. The `TWI_ISTAT.SINIT` indicates whether or not a slave transfer is in progress.	
		0	No Interrupt. A transfer is not in progress, or an address match has not occurred since the last time this bit was cleared.
		1	Interrupt Detected. The slave has detected an address match, and a transfer has been initiated.

Master Mode Address Register

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of `TWI_MSTRADDR`. When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is b#1010000X, where X is the read/write bit, the `TWI_MSTRADDR` is programmed with b#1010000, which corresponds to 0x50. When sending out the address on the bus, the TWI controller appends the read/write bit as appropriate based on the state of the `TWI_MSTRCTL.DIR` bit.

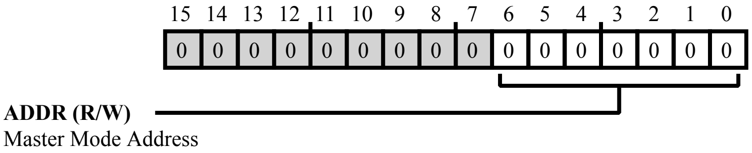


Figure 30-20: TWI_MSTRADDR Register Diagram

Table 30-13: TWI_MSTRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Master Mode Address.

Master Mode Control Registers

The TWI_MSTRCTL controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

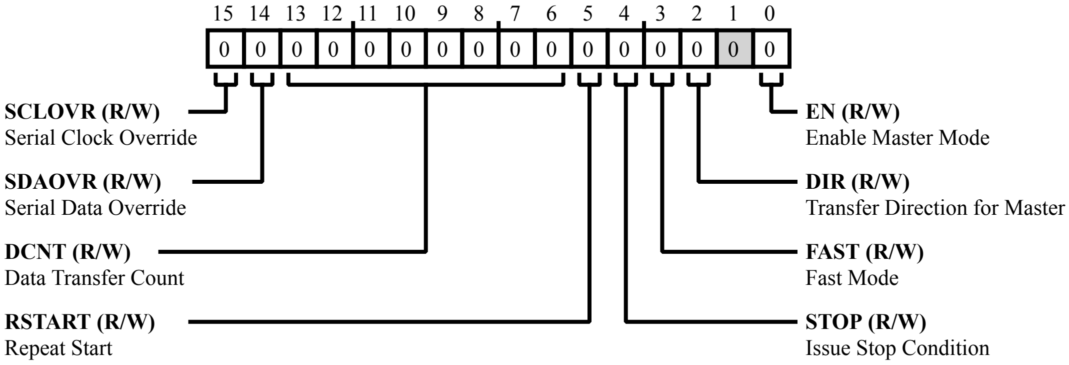

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCLOVR (R/W)
Serial Clock Override
SDAOVR (R/W)
Serial Data Override
DCNT (R/W)
Data Transfer Count
RSTART (R/W)
Repeat Start
EN (R/W)
Enable Master Mode
DIR (R/W)
Transfer Direction for Master
FAST (R/W)
Fast Mode
STOP (R/W)
Issue Stop Condition

Figure 30-21: TWI_MSTRCTL Register Diagram

Table 30-14: TWI_MSTRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLOVR	Serial Clock Override. The TWI_MSTRCTL.SCLOVR provides direct control of the serial clock line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SCLOVR is set, the TWI overrides normal serial clock output, driving it to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SCLOVR is cleared, the TWI permits normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic.	
		0	Permit Normal SCL Operation
		1	Override Normal SCL Operation
14 (R/W)	SDAOVR	Serial Data Override. The TWI_MSTRCTL.SDAOVR provides direct control of the serial data line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SDAOVR is set, the TWI overrides normal serial data operation under the control of the transmit shift register and acknowledge logic, driving serial data output to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SDAOVR is cleared, the TWI permits normal serial data operation.	
		0	Permit Normal SDA Operation
		1	Override Normal SDA Operation

Table 30-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:6 (R/W)	DCNT	Data Transfer Count. The `TWI_MSTRCTL.DCNT` indicates the number of data bytes to transfer. As each data word is transferred, the TWI decrements this counter. When `TWI_MSTRCTL.DCNT` decrements to 0, a stop condition is generated. Setting `TWI_MSTRCTL.DCNT` to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the `TWI_MSTRCTL.STOP` bit. In the event a master transmit is aborted due to a slave data NAK, the value of `TWI_MSTRCTL.DCNT` equals the number of bytes not sent. The byte which was NAK'ed by the slave is counted as a sent byte.
5 (R/W)	RSTART	Repeat Start. The `TWI_MSTRCTL.RSTART` enables the TWI to issue a repeat start condition at the conclusion of the current transfer (`TWI_MSTRCTL.DCNT` =0) and begin the next transfer. The current transfer concludes with updates to the appropriate status and interrupt bits. If errors occurred during the previous transfer, a repeat start does not occur. In the absence of any errors, master enable (`TWI_MSTRCTL.EN`) does not self clear on a repeat start. 0 Disable Repeat Start 1 Enable Repeat Start
4 (R/W)	STOP	Issue Stop Condition. The `TWI_MSTRCTL.STOP` directs the TWI to issue a stop condition. The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached). At that time, the `TWI_IMSK` is updated along with any associated status bits. 0 Permit Normal Operation 1 Issue Stop
3 (R/W)	FAST	Fast Mode. The `TWI_MSTRCTL.FAST` selects whether the TWI operates in fast mode or standard mode. In fast mode, the TWI uses timing specifications for transfers at up to 400K bits/s. In standard mode, the TWI uses timing specifications for transfers at up to 100K bits/s. 0 Select Standard Mode 1 Select Fast Mode
2 (R/W)	DIR	Transfer Direction for Master. The `TWI_MSTRCTL.DIR` selects the transfer direction for the TWI as master initiated receive or transmit. 0 Master Transmit 1 Master Receive

Table 30-14: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Master Mode. The `TWI_MSTRCTL.EN` enables master mode functionality. A start condition is generated if the bus is idle. This bit self clears at the completion of a transfer (after `TWI_MSTRCTL.DCNT` decrements to zero), including transfers terminated due to errors. If disabled (=0) during operation, the transfer is aborted, and all logic associated with master mode transfers are reset. Serial data and serial clock (SDA, SCL) are no longer driven. Write-1-to-clear status bits are not affected.	
		0	Disable
		1	Enable

Master Mode Status Register

The `TWI_MSTRSTAT` holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupt requests, but these bits offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Note that while `TWI_MSTRSTAT.SCLSEN` is set (this condition could be due to having no pull-up resistor on `TWI_SCL` or another agent is driving `TWI_SCL` low), the acknowledge bits (`TWI_MSTRSTAT.ANAK` and `TWI_MSTRSTAT.DNAK`) do not update. This result occurs because the acknowledge conditions are sampled during the high phase of `TWI_SCL`.

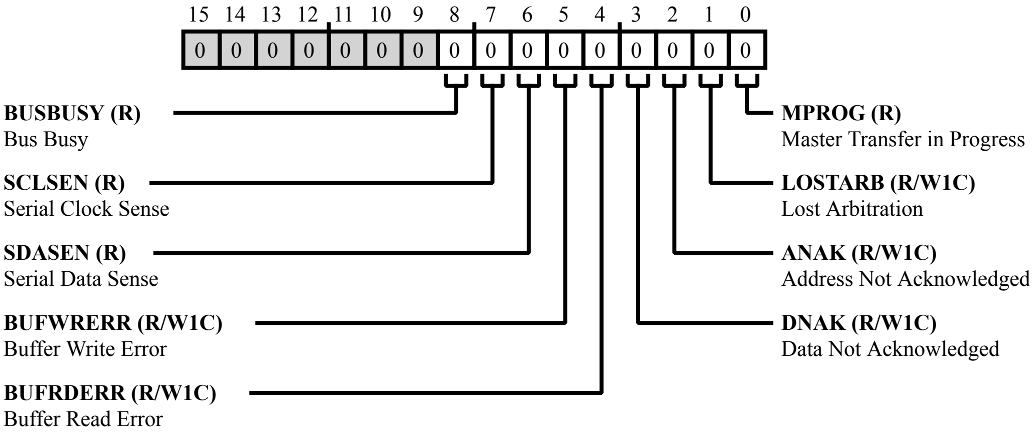

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BUSBUSY (R)
Bus Busy
SCLSEN (R)
Serial Clock Sense
SDASEN (R)
Serial Data Sense
BUFWRERR (R/W1C)
Buffer Write Error
BUFRDERR (R/W1C)
Buffer Read Error
MPROG (R)
Master Transfer in Progress
LOSTARB (R/W1C)
Lost Arbitration
ANAK (R/W1C)
Address Not Acknowledged
DNAK (R/W1C)
Data Not Acknowledged

Figure 30-22: TWI_MSTRSTAT Register Diagram

Table 30-15: TWI_MSTRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	BUSBUSY	Bus Busy. The `TWI_MSTRSTAT.BUSBUSY` indicates whether the bus is currently busy or free. This indication is not limited to only this device but is for all devices. On a start condition, the setting of the register value is delayed due to the input filtering. On a stop condition the clearing of the register value occurs after t_{BUF}.	
		0	Bus Free. The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time.
		1	Bus Busy. The bus is busy. Clock or data activity has been detected.

Table 30-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SCLSEN	Serial Clock Sense. The `TWI_MSTRSTAT.SCLSEN` indicates the active or inactive state of the serial clock. Use this status bit when direct sensing of the serial clock line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SCL Inactive "One". An inactive "one" is being sensed on the serial clock.
		1	SCL Active "Zero". An active "zero" is being sensed on the serial clock. The source of the active driver is not known and can be internal or external.
6 (R/NW)	SDASEN	Serial Data Sense. The `TWI_MSTRSTAT.SDASEN` indicates the active or inactive status of the serial data. Use this status bit when direct sensing of the serial data line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SDA Inactive "One". An inactive "one" is currently being sensed on the serial data line.
		1	SDA Active "Zero". An active "zero" is currently being sensed on the serial data line. The source of the active driver is not known and can be internal or external.
5 (R/W1C)	BUFWRERR	Buffer Write Error. The `TWI_MSTRSTAT.BUFWRERR` indicates whether the current master transfer was aborted due to a receive buffer write error. The receive buffer and receive shift register were both full at the same time. This bit is W1C.	
		0	No Status
		1	Buffer Write Error
4 (R/W1C)	BUFRDERR	Buffer Read Error. The `TWI_MSTRSTAT.BUFRDERR` indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Buffer Read Error
3 (R/W1C)	DNAK	Data Not Acknowledged. The `TWI_MSTRSTAT.DNAK` indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Data NAK

Table 30-15: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	ANAK	Address Not Acknowledged. The `TWI_MSTRSTAT.ANAK` indicates whether the current master transfer was aborted due to the detection of a NAK during the address phase of the transfer. This bit is W1C.	
		0	No Status
		1	Address NAK
1 (R/W1C)	LOSTARB	Lost Arbitration. The `TWI_MSTRSTAT.LOSTARB` indicates whether the current transfer was aborted due to the loss of arbitration with another master. This bit is W1C.	
		0	No Status
		1	Lost Arbitration
0 (R/NW)	MPROG	Master Transfer in Progress. The `TWI_MSTRSTAT.MPROG` indicates whether or not a master transfer is in progress. If clear (`TWI_MSTRSTAT.MPROG` =0), currently no transfer is taking place. This can occur after a transfer is complete or while an enabled master is waiting for an idle bus.	
		0	No Status
		1	Master Transfer in Progress

Rx Data Double-Byte Register

The `TWI_RXDATA16` holds a 16-bit data value read from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

The data is read in little endian byte order, where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (`TWI_FIFOSTAT.RXSTAT`) field is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the read data is unknown and the existing FIFO buffer data and its status remains unchanged.

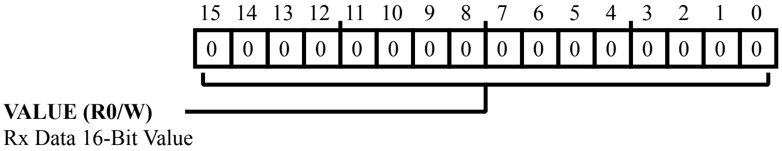


Figure 30-23: TWI_RXDATA16 Register Diagram

Table 30-16: TWI_RXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Rx Data 16-Bit Value.

Rx Data Single-Byte Register

The `TWI_RXDATA8` holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in first-out order. Although peripheral bus reads are 16 bits, a read access to `TWI_RXDATA8` accesses only one transmit data byte from the FIFO buffer. With each access, the receive status (`TWI_FIFOSTAT.RXSTAT`) field is updated. If an access is performed while the FIFO buffer is empty, the data is unknown and the FIFO buffer status remains indicating it is empty.

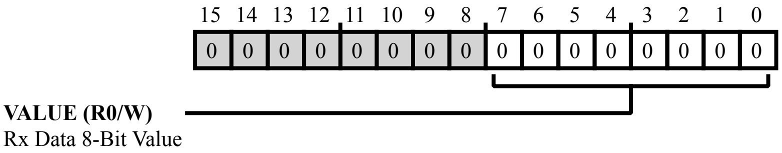


Figure 30-24: TWI_RXDATA8 Register Diagram

Table 30-17: TWI_RXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Rx Data 8-Bit Value.

Slave Mode Address Register

The `TWI_SLVADDR` holds the slave mode address, which is the valid address to which the slave-enabled TWI controller responds. The TWI controller compares this value with the received address during the addressing phase of a transfer.

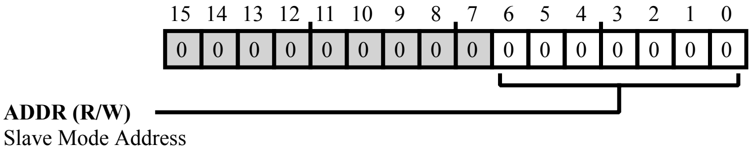

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR (R/W)
Slave Mode Address

Figure 30-25: TWI_SLVADDR Register Diagram

Table 30-18: TWI_SLVADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Slave Mode Address.

Slave Mode Control Register

The TWI_SLVCTL controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

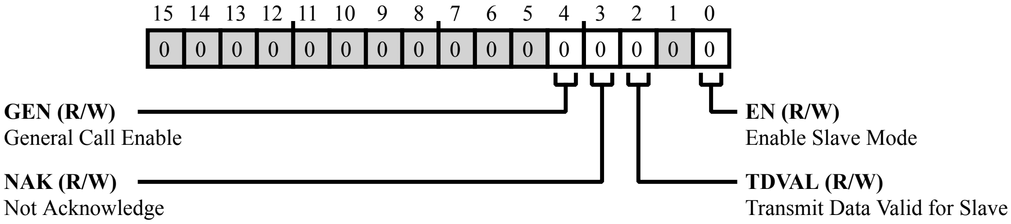


Figure 30-26: TWI_SLVCTL Register Diagram

Table 30-19: TWI_SLVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	GEN	General Call Enable. The TWI_SLVCTL.GEN enables general call address matching. When enabled, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated. Note that general call address detection is available only when slave mode is enabled.	
		0	Disable General Call Matching
		1	Enable General Call Matching
3 (R/W)	NAK	Not Acknowledge. The TWI_SLVCTL.NAK directs the TWI to generate a NAK (if set) or an ACK (if cleared) at the conclusion of data transfer for slave receive. For NAK, the slave is still considered to be addressed at the conclusion of transfer.	
		0	Generate ACK
		1	Generate NAK
2 (R/W)	TDVAL	Transmit Data Valid for Slave. The TWI_SLVCTL.TDVAL selects whether the data in the transmit FIFO is available (valid) for slave transmission (TWI_SLVCTL.TDVAL set). If the FIFO data is not available (invalid) for slave transmission (TWI_SLVCTL.TDVAL cleared), the data in the transmit FIFO is for master mode transmits, and the data is not allowed to be used during a slave transmit; the transmit FIFO is treated as if it is empty.	
		0	Data Invalid for Slave Tx
		1	Data Valid for Slave Tx

Table 30-19: TWI_SLVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Slave Mode. The `TWI_SLVCTL.EN` enables slave operation. Enabling slave and master modes of operation concurrently is allowed. If disabled, no attempt is made to identify a valid address. If `TWI_SLVCTL.EN` is cleared during a valid transfer, clock stretching ceases, the serial data line is released, and the current byte is not acknowledged.	
		0	Disable
		1	Enable

Slave Mode Status Register

During and at the conclusion of register slave mode transfers, the `TWI_SLVSTAT` holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupt requests. Master mode operation does not affect slave mode status bits.

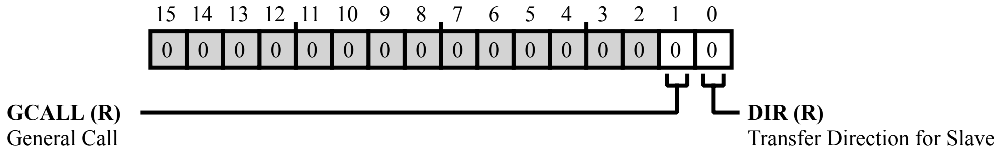


Figure 30-27: TWI_SLVSTAT Register Diagram

Table 30-20: TWI_SLVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	GCALL	General Call. The `TWI_SLVSTAT.GCALL` indicates whether or not--at the time of addressing--the address was determined to be a general call. This bit self clears if slave mode is disabled (`TWI_SLVCTL.EN` =0).	
		0	Not a General Call Address
		1	General Call Address
0 (R/NW)	DIR	Transfer Direction for Slave. The `TWI_SLVSTAT.DIR` indicates whether--at the time of addressing--the transfer direction was determined to be slave transmit or receive. This bit self clears if slave mode is disabled (`TWI_SLVCTL.EN` =0).	
		0	Slave Receive
		1	Slave Transmit

Tx Data Double-Byte Register

The `TWI_TXDATA16` register holds a 16-bit data value written into the FIFO buffer. To reduce interrupt latency output rates and peripheral bus access times, a double byte transfer data access can be done. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little endian byte order, where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (`TWI_FIFOSTAT.TXSTAT`) field is updated. If an access is performed while the FIFO buffer is not empty, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register when read back returns zero.

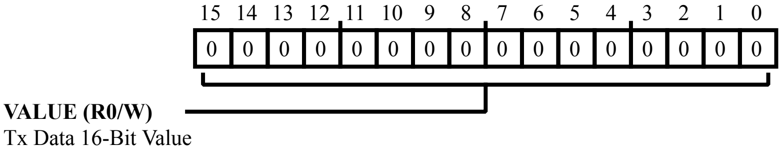


Figure 30-28: TWI_TXDATA16 Register Diagram

Table 30-21: TWI_TXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Tx Data 16-Bit Value.

Tx Data Single-Byte Register

The `TWI_TXDATA8` register holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in first-out order. For 16-bit peripheral bus writes, a write access to this register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (`TWI_FIFOSTAT.TXSTAT`) field is updated. If an access is performed while the FIFO buffer is full, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register returns zero when read back.

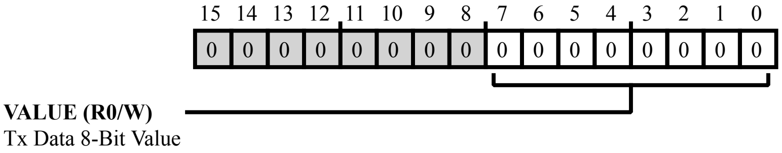


Figure 30-29: TWI_TXDATA8 Register Diagram

Table 30-22: TWI_TXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Tx Data 8-Bit Value.

31 Ethernet Media Access Controller (EMAC)

The EMAC peripheral in the processor enables network connectivity to applications through an Ethernet interface. The module is fully compliant to the following standards:

NOTE: On the processor, EMAC0 is 10/100/1000 BaseT-compliant and supports AVB (Audio Video Bridging) standards.

- Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Standard 802.3-2005, Institute of Electrical and Electronics Engineers (IEEE).
- Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Standard 1588–2008, Institute of Electrical and Electronics Engineers (IEEE).
- Reduced Media Independent Interface Specification, Revision 1.2, RMII Consortium.
- Reduced Gigabit Media Independent Interface (RGMII), Revision 2.6, HP/Marvell.
- Timing and Synchronization for Time-Sensitive Applications in Bridged LANs, Standard 802.1AS-2011, Institute of Electrical and Electronics Engineers (IEEE).
- Forwarding and Queuing Enhancements for Time-Sensitive Streams, Standard 802.1Qav-2009, Institute of Electrical and Electronics Engineers (IEEE).
- The EMAC supports IEEE 803.3az-2010 standard for Energy Efficient Ethernet. This feature enables the Media Access Control (MAC) sublayer along with a family of physical layers to operate in the Low-Power Idle (LPI) mode.

NOTE: Copyright © 2010 Synopsys, Inc.; portions of this chapter are included with permission from Synopsys, Inc.

EMAC Features

The Ethernet features include the following:

- Supports 10/100 Mbps data transfer rates with external PHY interfaced through RMII
- Full-duplex and half-duplex support for Ethernet
- Dedicated DMA controller with independent read write channels

- Supports dual-buffer (ring) or linked-list (chained) descriptor chaining
- Direct interface with the system crossbar bus
- Provides support for CSMA/CD protocol for half-duplex operation
- IEEE 802.3x flow control for full-duplex and half-duplex
- Automatic network monitoring statistics with management counters
- Flexible address filtering options for uni-cast, multi-cast, and broadcast addresses
- Support for promiscuous mode in reception
- Supports IEEE 802.1Q VLAN tag detection
- VLAN tag-based frame filtering - Perfect match and hash-based filtering
- Supports programmable Inter-frame Gap (IFG)
- Checksum offload engine for checking IPv4 header checksum and TCP/UDP/ICMP checksum encapsulated in IPv4 or IPv6 datagrams
- Station management interface for PHY device configuration and management
- Includes FIFOs for buffering: 1024 bytes for transmit FIFO and 512 bytes for receive FIFO
- Automatic CRC and pad generation controllable on a per-frame basis
- CRC replacement, source address field insertion or replacement, and VLAN insertion, replacement, and deletion in transmitted frames with per-frame control
- Layer 3 and layer 4-based frame filtering - TCP or UDP over IPv4 or IPv6

The Ethernet MAC EMAC0 supports the following extra features

- Supports Gigabit data transfer rates with external PHY interfaced through RGMII
- Ethernet frame timestamping as described in IEEE 1588–2002 and IEEE 1588–2008. The transmit or receive status of each frame include 64-bit timestamps
- Hardware assisted time stamping capable of up to 8-ns resolution
- Automatic detection of PTP messages through Ethernet, IPv4, and IPv6 packets
- Four programmable PPS outputs that physically represent PTP system time
- Auxiliary snapshot to timestamp external events
- Four auxiliary input pins available
- Two separate channels (channel 1 and channel 2) or queues for transmission and reception of time-sensitive traffic. Channel 0 is available by default and carries the legacy best-effort Ethernet traffic on the transmit side.
- IEEE 802.1-Qav specified credit-based shaper (CBS) algorithm for channel 1 and channel 2

- Slot number function to schedule the data fetching by DMA from the system memory for channel 1 and channel 2
- Separate DMA, Tx FIFO, and Rx FIFO (MTL) for each additional channel
- Programmable control to route received VLAN tagged non-AV packets to channels or queues
- Standard IEEE 802.3az-2010 for Energy Efficient Ethernet
- EMAC0 supports the following FIFO sizes: 2048 bytes for transmit FIFO and also for receive FIFO
-

EMAC Functional Description

This section provides information on the function of Ethernet MAC peripheral.

Hardware for the Media Access Control protocol

This function allows applications to support TCP/IP based network communication. At the system end, the module supports direct connection with the system crossbar bus for memory or MMR transactions. It supports RMII (Reduced Media Independent Interface), RGMII (Reduced Gigabit Media Independent Interface), and SMI (Station Management Interface) for interfacing with the external PHY chip.

Dedicated DMA Controller with independent read/write channels

Performs both data and status transfers between the application and the media independent interfaces. Internal transmit and receive FIFOs are used to buffer and regulate the frames. A dedicated interrupt line connects the EMAC interrupt sources to the System Event Controller (SEC).

MAC Management Counters (MMC) block

An extended set of registers that collect various statistics compliant with IEEE 802.3 definitions regarding the operation of the interface. The registers are updated for each new transmitted or received frame when the condition to update the counter is met. The EMAC provides a set of such counters, along with extended usage control.

PTP (Precision Time Protocol) engine

Provides hardware assistance for the implementation of the IEEE 1588 version 1 and version 2 standards, which allows time synchronization between systems.

Audio Video (AV) functionality

Enables transmission of time-sensitive traffic over bridged local area networks (LANs). The EMAC provides hardware support for IEEE 802.1-Qav specified credit-based shaper (CBS) algorithm. In addition, slot number function allows scheduling of fetching of data by DMA from system memory.

ADSP-SC58x EMAC Register List

The Ethernet MAC (EMAC) module provides an Ethernet interface to the processor. The interface is compliant to IEEE Standard 802.3-2005. A set of registers govern EMAC operations. For more information on EMAC functionality, see the EMAC register descriptions.

Table 31-1: ADSP-SC58x EMAC Register List

Name	Description
EMAC_ADDR0_HI	MAC Address 0 High Register
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_ADDR1_HI	MAC Address 1 High Register
EMAC_ADDR1_LO	MAC Address 1 Low Register
EMAC_DBG	Debug Register
EMAC_DMA0_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA0_BMSTAT	DMA SCB Status Register
EMAC_DMA0_BUSMODE	DMA Bus Mode Register
EMAC_DMA0_IEN	DMA Interrupt Enable Register
EMAC_DMA0_MISS_FRM	DMA Missed Frame Register
EMAC_DMA0_OPMODE	DMA Operation Mode Register
EMAC_DMA0_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA0_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA0_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA0_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA0_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA0_STAT	DMA Status Register
EMAC_DMA0_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA0_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA0_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA0_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA1_BUSMODE	DMA Bus Mode Register
EMAC_DMA1_CHCBSCTL	Channel 1 Credit Shaping Control Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_DMA1_CHCBSSTAT	Channel 1 Average Traffic Transmitted Register
EMAC_DMA1_CHHIC	Channel 1 High Credit Value Register
EMAC_DMA1_CHISC	Channel 1 Idle Slope Credit Value Register
EMAC_DMA1_CHLOC	Channel 1 Low Credit Value Register
EMAC_DMA1_CHSFCS	Channel 1 Control Bits for Slot Function Register
EMAC_DMA1_CHSSC	Channel 1 Send Slope Credit Value Register
EMAC_DMA1_IEN	DMA Interrupt Enable Register
EMAC_DMA1_MISS_FRM	DMA Missed Frame Register
EMAC_DMA1_OPMODE	DMA Operation Mode Register
EMAC_DMA1_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA1_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA1_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA1_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA1_RXPOLL	DMA Rx Poll Demand Register
EMAC_DMA1_STAT	DMA Status Register
EMAC_DMA1_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA1_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA1_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA1_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA2_BUSMODE	DMA Bus Mode Register
EMAC_DMA2_CHCBSCTL	Channel 2 Credit Shaping Control Register
EMAC_DMA2_CHCBSSTAT	Channel 2 Avg Traffic Transmitted Status Register
EMAC_DMA2_CHHIC	Channel 2 High Credit Value Register
EMAC_DMA2_CHISC	Channel 2 Idle Slope Credit Value Register
EMAC_DMA2_CHLOC	Channel 2 Low Credit Value Register
EMAC_DMA2_CHSFCS	Channel 2 Control Bits for Slot Function Register
EMAC_DMA2_CHSSC	Channel 2 Send Slope Credit Value Register
EMAC_DMA2_IEN	DMA Interrupt Enable Register
EMAC_DMA2_MISS_FRM	DMA Missed Frame Register
EMAC_DMA2_OPMODE	DMA Operation Mode Register
EMAC_DMA2_RXBUF_CUR	DMA Rx Buffer Current Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_DMA2_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA2_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA2_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA2_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA2_STAT	DMA Status Register
EMAC_DMA2_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA2_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA2_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA2_TXPOLL	DMA Tx Poll Demand Register
EMAC_FLOWCTL	FLow Control Register
EMAC_GIGE_CTLSTAT	RGMII Control and Status Register
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_IMSK	Interrupt Mask Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_ISTAT	Interrupt Status Register
EMAC_L3L4_CTL	Layer3 and Layer4 Control Register
EMAC_L3_ADDR0	Layer 3 Address0 Register
EMAC_L3_ADDR1	Layer 3 Address1 Register
EMAC_L3_ADDR2	Layer 3 Address2 Register
EMAC_L3_ADDR3	Layer 3 Address3 Register
EMAC_L4_ADDR	Layer 4 Address Register
EMAC_LPI_CTLSTAT	Low Power Idle Control and Status Register
EMAC_LPI_TMRSCTL	Low Power Idle Timeout Register
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register
EMAC_MAC_AVCTL	AV MAC Control Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RXALIGN_ERR	Rx alignment Error Register
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXCTLFRM_G	Rx Good Control Frames Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXRCV_ERR	Rx Error Frames Received Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_CTL	Time Stamp Control Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_PPS0INTVL	Time Stamp PPS Interval Register
EMAC_TM_PPS0NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_PPS0TGTM	Time Stamp Target Time Seconds Register
EMAC_TM_PPS0WIDTH	PPS Width Register
EMAC_TM_PPS1INTVL	PPS 1 Interval Register
EMAC_TM_PPS1NTGTM	PPS 1 Target Time Nanoseconds Register
EMAC_TM_PPS1TGTM	PPS 1 Target Time Seconds Register
EMAC_TM_PPS1WIDTH	PPS 1 Width Register
EMAC_TM_PPS2INTVL	PPS 2 Interval Register
EMAC_TM_PPS2NTGTM	PPS 2 Target Time Nanoseconds Register
EMAC_TM_PPS2TGTM	PPS 2 Target Time Seconds Register
EMAC_TM_PPS2WIDTH	PPS 2 Width Register
EMAC_TM_PPS3INTVL	PPS 3 Interval Register
EMAC_TM_PPS3NTGTM	PPS 3 Target Time Nanoseconds Register
EMAC_TM_PPS3TGTM	PPS 3 Target Time Seconds Register
EMAC_TM_PPS3WIDTH	PPS 3 Width Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register

Table 31-1: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register
EMAC_TXMCASTFRM_GB	Tx Multicast Frames (Good/Bad) Register
EMAC_TXMULTCOL_G	Tx Multiple Collision (Good) Register
EMAC_TXOCTCNT_G	Tx Octet Count (Good) Register
EMAC_TXOCTCNT_GB	Tx OCT Count (Good/Bad) Register
EMAC_TXOVRSIZE_G	Number of Tx Frames (Good) greater than maxsize
EMAC_TXPAUSEFRM	Tx Pause Frame Register
EMAC_TXSNGCOL_G	Tx Single Collision (Good) Register
EMAC_TXUCASTFRM_GB	Tx Unicast Frames (Good/Bad) Register
EMAC_TXUNDR_ERR	Tx Underflow Error Register
EMAC_TXVLANFRM_G	Tx VLAN Frames (Good) Register
EMAC_VLANTAG	VLAN Tag Register
EMAC_VLAN_HSHTBL	VLAN Hash Table Register
EMAC_VLAN_INCL	VLAN Tag Inclusion or Replacement Register
EMAC_WDOG_TIMOUT	Watchdog Timeout Register

ADSP-SC58x EMAC Interrupt List

Table 31-2: ADSP-SC58x EMAC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
148	EMAC0_STAT	EMAC0 Status	None	
149	EMAC1_STAT	EMAC1 Status	None	

ADSP-SC58x EMAC Trigger List

Table 31-3: ADSP-SC58x EMAC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
55	`EMAC0_STAT`	EMAC0 Status	None
56	`EMAC1_STAT`	EMAC1 Status	None

Table 31-4: ADSP-SC58x EMAC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

EMAC Definitions

The following definitions are helpful for using the EMAC.

EMAC SCB

System Crossbar Interface of EMAC

AVB

Audio Video Bridging

EEE

Energy Efficient Ethernet

EMAC DMA

DMA Controller of EMAC

EMAC MFL

MAC FIFO Layer inside EMAC

EMAC CORE

CORE layer inside EMAC which performs the actual Ethernet operations, including interface with PHY through the reduced media interface(s).

MMC

MAC Management Counter

SMI

Station Management Interface that controls PHY through MDIO and MDC signals.

RMII

Reduced Media Independent Interface

MAC

Media Access Control

PTP

Precision Time Protocol

EMAC Block Diagram and Interfaces

The ***EMAC Simplified Block Diagram*** illustrates the overall functional architecture of the Ethernet MAC peripheral. The EMAC module is comprised of four major layers: EMAC SCB, EMAC DMA, EMAC MFL, and EMAC CORE. Each of these layers (subblocks) is explained in depth in their respective sections in this chapter.

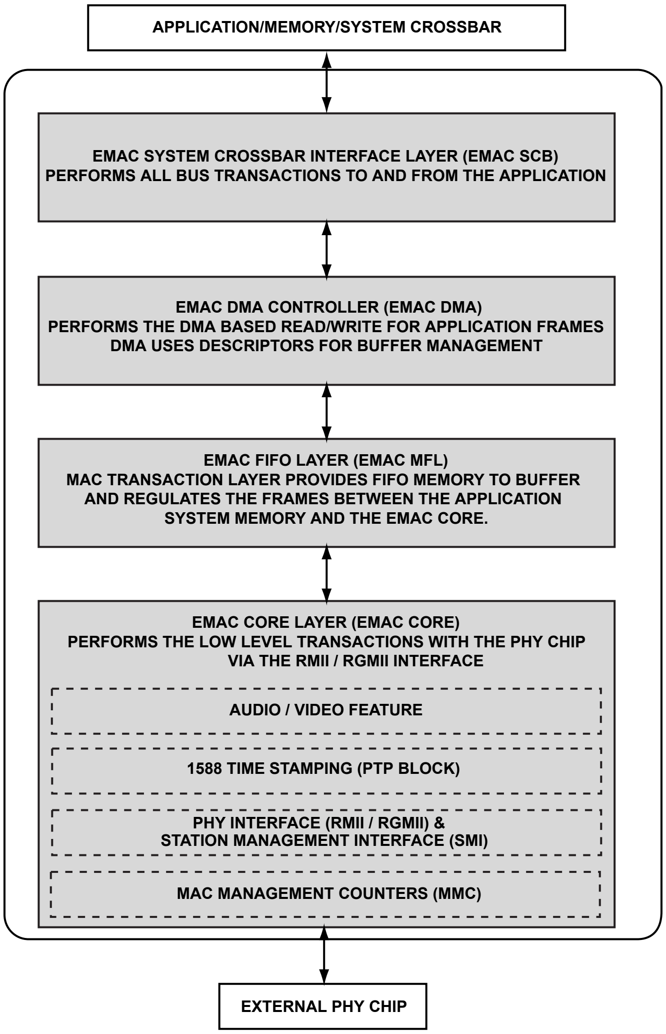

APPLICATION/MEMORY/SYSTEM CROSSBAR
EMAC SYSTEM CROSSBAR INTERFACE LAYER (EMAC SCB)
PERFORMS ALL BUS TRANSACTIONS TO AND FROM THE APPLICATION
EMAC DMA CONTROLLER (EMAC DMA)
PERFORMS THE DMA BASED READ/WRITE FOR APPLICATION FRAMES
DMA USES DESCRIPTORS FOR BUFFER MANAGEMENT
EMAC FIFO LAYER (EMAC MFL)
MAC TRANSACTION LAYER PROVIDES FIFO MEMORY TO BUFFER
AND REGULATES THE FRAMES BETWEEN THE APPLICATION
SYSTEM MEMORY AND THE EMAC CORE.
EMAC CORE LAYER (EMAC CORE)
PERFORMS THE LOW LEVEL TRANSACTIONS WITH THE PHY CHIP
VIA THE RMII / RGMII INTERFACE
AUDIO / VIDEO FEATURE
1588 TIME STAMPING (PTP BLOCK)
PHY INTERFACE (RMII / RGMII) &
STATION MANAGEMENT INTERFACE (SMI)
MAC MANAGEMENT COUNTERS (MMC)
EXTERNAL PHY CHIP

Figure 31-1: EMAC Simplified Block Diagram (RGMII, AV Features, and PTP are only available on EMAC0)

A more comprehensive block diagram is shown in the ***EMAC Complete Block Diagram***. It includes most of the important blocks inside the EMAC. The EMAC is connected to processor memory and the system crossbar through the System Crossbar Bus Interface (SCB) and System Peripheral Bus Interface (SPB). These connections are which are part of the SCB layer. The SPB interface is connected to all modules that require MMR programming.

The DMA controller performs application data transfer frame by frame, through well-defined descriptor structures. A FIFO layer acts as a buffer between the DMA controller and EMAC CORE.

The EMAC CORE is the most important block because it contains subblocks to support IEEE802.3 based communication with external network interfaces of 10/100/1000 Mbps speeds. It includes the PTP subblock, which assists applications requiring time synchronization; the AV Feature subblock, which enables transmission of time-sensitive traffic over bridged LANs; and a MMC subblock, which generates packet transfer statistics.

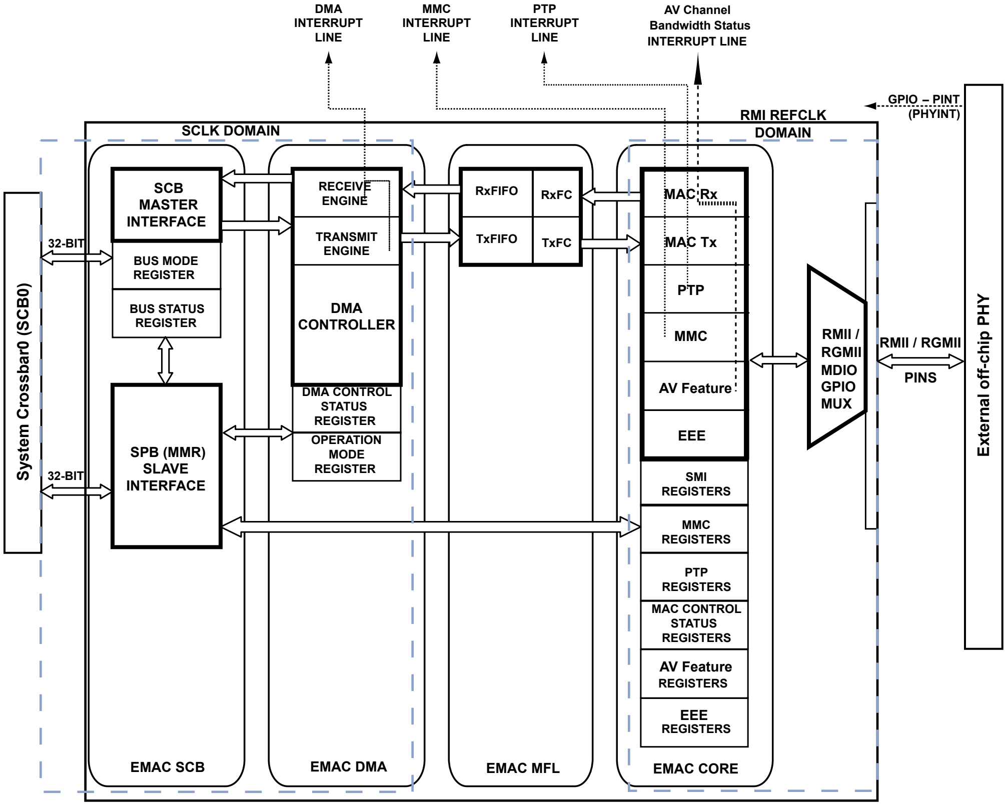

DMA INTERRUPT LINE
MMC INTERRUPT LINE
PTP INTERRUPT LINE
AV Channel Bandwidth Status INTERRUPT LINE
GPIO – PINT (PHYINT)
SCLK DOMAIN
RMI REFCLK DOMAIN
System Crossbar0 (SCB0)
32-BIT
32-BIT
SCB MASTER INTERFACE
BUS MODE REGISTER
BUS STATUS REGISTER
SPB (MMR) SLAVE INTERFACE
RECEIVE ENGINE
TRANSMIT ENGINE
DMA CONTROLLER
DMA CONTROL STATUS REGISTER
OPERATION MODE REGISTER
RxFIFO
RxFC
TxFIFO
TxFC
MAC Rx
MAC Tx
PTP
MMC
AV Feature
EEE
SMI REGISTERS
MMC REGISTERS
PTP REGISTERS
MAC CONTROL STATUS REGISTERS
AV Feature REGISTERS
EEE REGISTERS
RMII / RGMII MDIO GPIO MUX
RMII / RGMII
PINS
External off-chip PHY
EMAC SCB
EMAC DMA
EMAC MFL
EMAC CORE

Figure 31-2: EMAC Complete Block Diagram (RGMII, AV Features, and PTP are only available on EMAC0)

EMAC CORE Subblocks

The *Core Transmit Engine Subblocks* table summarizes the core transmit engine subblocks and their functions. Refer to the EMAC CORE section for further explanation of each of these subblocks.

Table 31-5: CORE Transmit Engine Subblocks

CORE Transmit Engine Sub Block	Function
Transmit Bus Interface	Interface to the FIFO.
Transmit Frame Controller	Appends Zero-PAD data, if required, for short frames. Appends CRC for frame checksum from the CRC generator.
Transmit Protocol Engine	Generates preamble and SFD, as per 802.3 protocol. Generates jam pattern in half-duplex mode, for collisions. Jabber timeout, for excessively large frames. Flow control for half-duplex mode (back pressure). Generates transmit frame status.
Transmit Scheduler	Maintains the inter-frame gap between two transmitted frames. Follows the truncated binary exponential back-off algorithm for half-duplex mode.

Table 31-5: CORE Transmit Engine Subblocks (Continued)

CORE Transmit Engine Sub Block	Function
Transmit CRC Generator	Generate CRC for the frame checksum field of the Ethernet frame.
Transmit Flow Control	Receives the pause frame, appends the calculated CRC, and sends the frame to the protocol engine module.
Transmit Checksum Offload Engine	Supports checksum calculation and insertion in the transmit path, for IPV4/TCP/UDP/ICMP packets.

The ***Core Receive Engine Subblocks*** table summarizes the core receive engine subblocks and their function. Refer to the EMAC CORE section for more information on each of these subblocks.

Table 31-6: Core Receive Engine Subblocks

CORE Receive Engine Subblock	Functionality Overview
Receive Protocol Engine	Strips the incoming preamble and SFD. Checks for correct length or type field. Performs internal loopback, if necessary. Generates receive status. Supports watchdog of received frames. Supports jumbo frames.
Receive CRC Module	Checks for CRC error, by comparing with FCS.
Receive Frame Controller Module	Packs incoming 8-bit input stream to 32-bit data internally. Performs frame filtering, for uni-cast, multi-cast, and broadcast frames. Attaches the calculated IP checksum input from checksum offload engine. Updates the receive status to bus interface.
Receive Flow Control Module	Detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame. Works in full duplex mode.
Receive IP Checksum Offload Engine	Calculates IPv4 header checksums and verify against the received IPv4 header checksums. Identifies a TCP, UDP, or ICMP payload in the received IP datagrams.
Receive Bus Interface Unit Module	Interface to the FIFO.
Address Filtering Module	Filters destination and source address based on uni-cast, multi-cast, and broadcast frames. Provides CRC hash filtering.

EMAC PHY Interface

The EMAC can interface to the PHY through the RMII interface standard. The ***RMII Pins*** table shows the RMII pins available in the EMAC, in terms of their generic names. Refer to the data sheet for exact pin names.

Table 31-7: RMII Pins

Sl. No.	Generic Signal Name (IEEE Standards)	RMII Pin Functionality
1.	TXD0	RMII transmit data pin D0 (di-bit lower)
2.	TXD1	RMII transmit data pin D1 (di-bit higher)
3.	RXD0	RMII receive data pin D0 (di-bit lower)
4.	RXD1	RMII receive data pin D1 (di-bit higher)
5.	RMII CLK	RMII common clock (for TX and RX), also called reference clock
6.	TXEN	RMII transmit enable pin (TX valid)
7.	CRS	RMII carrier sense / receive data valid
8.	MDC	Serial management clock driven by EMAC
9.	MDIO	Serial management bidirectional data

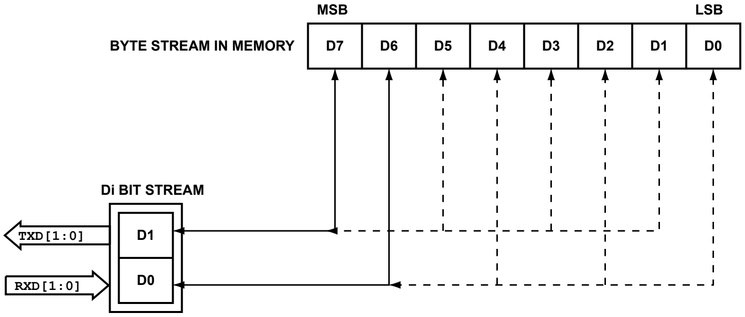


Figure 31-3: RMII Di-bit Data Transfer

Extended EMAC PHY Interface

The EMAC can interface to the PHY through the RGMII interface standard. The ***RGMII Pins*** table shows the RGMII pins available in the EMAC, in terms of their generic names. Refer to the data sheet for exact pin names.

Table 31-8: RGMII Pins

Sl. No.	Generic Signal Name (IEEE Standards)	RGMII Pin Functionality
1.	TXD3-0	RGMII transmit data pins D3-0
2.	RXD3-0	RGMII receive data pins D3-0
3.	TXCLK	RGMII transmit reference clock driven by EMAC
4.	RXCLK	RGMII receive reference clock driven by PHY
5.	TXCTL	RGMII transmit enable pin (TX valid)
6.	RXCTL	RGMII receive data valid

Table 31-8: RGMII Pins (Continued)

Sl. No.	Generic Signal Name (IEEE Standards)	RGMII Pin Functionality
7.	MDC	Serial management clock driven by EMAC
8.	MDIO	Serial management bidirectional data

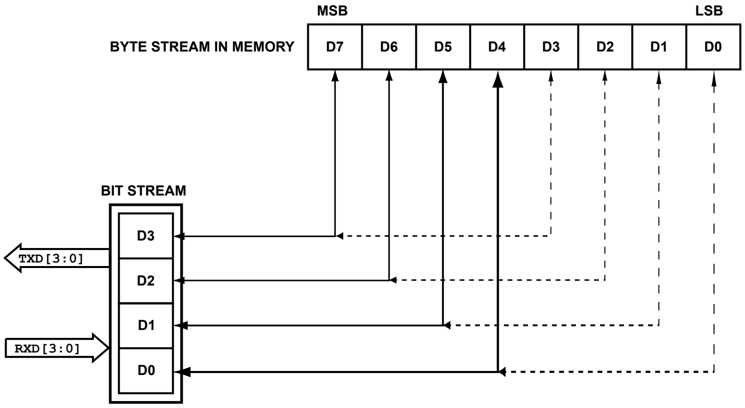


Figure 31-4: RGMII Data Bit Transfer

PHY Interface Selection

The EMAC1 supports only 10/100 Mbps data-transfer rates with external PHY interfaced through RMII. The EMAC0 supports both 10/100 Mbps data-transfer rates with external PHY interfaced through RMII and 10/100/1000 Mbps data-transfer rates with external PHY interfaced through RGMII.

Select the external PHY interface for EMAC0 using the `PADS_PCFG0` register, as shown in the following table.

`PADS_PCFG0.EMACPHYISEL`	0 = RMII Interface 1 = RGMII Interface
`PADS_PCFG0.EMACRESET`	0 = reset is asserted 1 = reset is deasserted To select PHY interface, set `PADS_PCFG0.EMACPHYISEL` bit as required and then set `PADS_PCFG0.EMACRESET`.

RGMII Board Design Recommendations

Use the following guidelines during when performing board design when using the RGMII interface.

MAC to PHY (Transmit)

The Ethernet MAC transmits data to the Ethernet PHY. The Ethernet MAC sends data with tskewT (the timing of TXC at the MAC) that meets the RGMII specification (tskewT = –500 ps to +500 ps skew window for transmitter to drive data). The RGMII specification requires that at the PHY end, tskewR is sampled at 1.0 to 2.6 ns. According to the RGMII standard, clocks must be routed such that an additional trace delay of greater than 1.5 ns and less than 2 ns is added to the associated clock signal.

To meet this standard without adding trace delays, most of the PHYs in the industry already include delay logic that can compensate for this on-board delay. These PHY types can manage a tskewR of ±500 ps (the skew for TXC data sampling seen inside the PHY).

The PHY or the on-board delay must delay the clock signal by 1.5 to 2.0 ns so that tskewR is sampled at 1.0 to 2.6 ns. The ***MAC to PHY Delay Diagram (Transmitting Data)*** figure shows where the delays must occur.

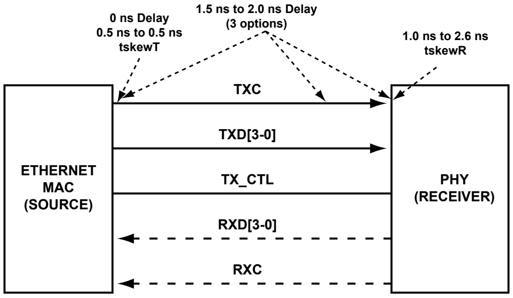

0 ns Delay
0.5 ns to 0.5 ns
tskewT
1.5 ns to 2.0 ns Delay
(3 options)
1.0 ns to 2.6 ns
tskewR
TXC
TXD[3-0]
ETHERNET
MAC
(SOURCE)
TX_CTL
PHY
(RECEIVER)
RXD[3-0]
RXC

Figure 31-5: MAC to PHY Delay Diagram (Transmitting Data)

PHY to MAC (Receive)

The Ethernet MAC receives data from the Ethernet PHY. Just as in the transmit case, a trace delay of greater than 1.5 ns and less than 2 ns must be added to the associated clock signal, as required by the RGMII specification. Also similar to the transmit case, most of the PHYs in the industry already include delay logic that can compensate for the RXC clock as well.

As shown in the ***MAC to PHY Delay Diagram (Receiving Data)*** figure, a board trace or the PHY can be used to generate the required 1.5–2.0 ns delay to RXC and the 1.0–2.6 ns tskewR.

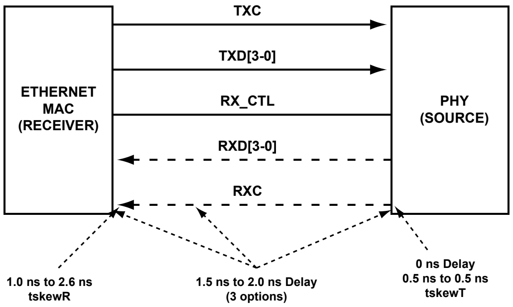

TXC
TXD[3-0]
ETHERNET
MAC
(RECEIVER)
RX_CTL
PHY
(SOURCE)
RXD[3-0]
RXC
1.0 ns to 2.6 ns
tskewR
1.5 ns to 2.0 ns Delay
(3 options)
0 ns Delay
0.5 ns to 0.5 ns
tskewT

Figure 31-6: MAC to PHY Delay Diagram (Receiving Data)

There following are two options for board design.

- The on-board delay must delay the clock signal by 1.5 to 2.0 ns. In this case, no additional delay must be introduced by the PHY.
- Most of the PHYs in the industry already include a mode that can introduce a delay logic. If this mode of the PHY is used then the trace lengths on the board need to be matched exactly.

The second option is recommended since it is easier to implement the delay by using the mode in the PHY rather than during the board design. This is the approach followed in the ADI EZ-Kits.

NOTE: Refer to the product specific datasheet for exact processor timing.

For the trace length recommendations for the EMAC signals, please refer to the [SYNOPSIS?] datasheet.

For the exact requirements as recommended by the RGMII protocol, please refer to the RGMII specification.

Clock Sources

The Ethernet MAC is clocked internally from SCLK0_0. Check the processor data sheet for the valid frequency range of the appropriate SCLK0_0 signal for Ethernet operation.

Source a 50-MHz clock externally to operate the EMAC in RMII mode. This clock is the same for both transmit and receive. The MDC station management clock is derived from the SCLK0_0 and driven from the MAC to the PHY, when accessing any PHY registers. EMAC1 only supports the RMII interface.

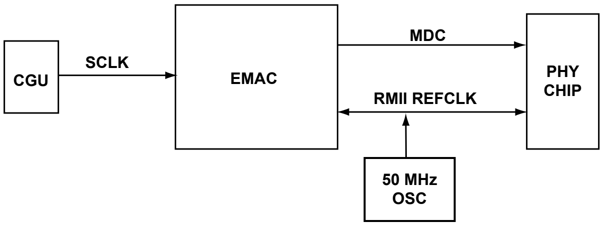


Figure 31-7: EMAC Clock Sources for RMII PHY interface

EMAC0 Clock Sources

EMAC0 supports RGMII and RMII interfaces. The external PHY sources a 2.5 MHz or 25 MHz clock (for 10/100 or gigabit Ethernet respectively) to operate the EMAC RXCLK in RGMII mode. The RGMII TXCLK is driven from CLK07 of the CDU (Clock Distribution Unit) and needs to be configured to 125 MHz regardless of the EMAC0 speeds (10/100/1000 Mbit/s). The `EMAC_MACCFG.PS` and `EMAC_MACCFG.FES` bits are used to divide the clocks down.

For EMAC0 to operate in RMII mode, configuration similar to the one shown in the ***EMAC Clock Sources for RGMII PHY Interface*** is required.

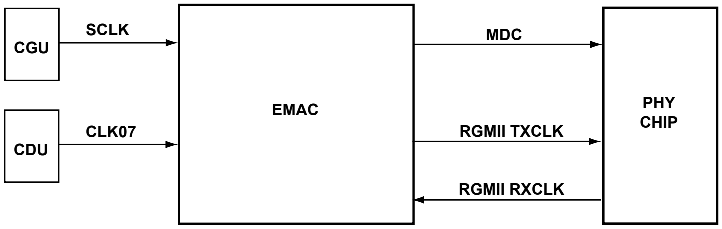


Figure 31-8: EMAC Clock Sources - RGMII PHY Interface

EMAC Architectural Concepts

This section explains different architectural concepts relevant to EMAC peripheral, such as EMAC SCB, EMAC DMA, EMAC MFL, EMAC CORE, and others.

EMAC Feature Summary

The ***EMAC Feature Summary*** table provides a summary of the features that are available on EMAC0 and EMAC1.

Table 31-9: EMAC Feature Summary

Feature	Value (EMAC0)	Value (EMAC1)
Speed of Operation	10/100/1000 Mbps	10/100 Mbps
EMAC_VER.UVER hardcoded value	0x10	0x11
PHY Interface	RGMII or RMII	RMII
Receive FIFO Size (in Bytes)	2048	512
Transmit FIFO Size (in Bytes)	2048	1024
Energy Efficient Ethernet (EEE)	Yes	No
PTP (IEEE 1588)	Yes	No
AV Feature	Yes	No
No of TX Channels	3	1
No of RX Channels	3	1

EMAC System Crossbar Interface (EMAC SCB)

The EMAC SCB bus interface provides the bus connectivity to support highly effective throughput of data traffic. System bus use is maximized by allowing simultaneous read and write transfers initiated from different DMA channels. The EMAC controller connects directly to the SCB0 crossbar. The following interfaces are available with the design.

- A 32-bit SCB master interface for reading and writing to and from the application memory.
- A 32-bit SPB slave interface for register programming.

Refer to the “System Crossbars (SCB)” chapter for more information on how the crossbar operates. This chapter details only the EMAC-specific information.

Table 31-10: EMAC-SCB Interface Data Transfer Specifications with Crossbar

Specification Term	Comments
1 beat in SCB	SINGLE burst
BLEN4 bursts	4 beats in SCB
BLEN8 bursts	8 beats in SCB
BLEN16 bursts	16 beats in SCB

Table 31-10: EMAC-SCB Interface Data Transfer Specifications with Crossbar (Continued)

Specification Term	Comments
Bus size	32-bit fixed bus size; equals 1 beat
INCR bursts	Incrementing Bursts
INCR ALIGNED bursts	Incrementing aligned bursts
UNDEF bursts	Undefined burst length
PBL	Programmable Burst Length for DMA

The ***EMAC DMA Read/Write channels with System Crossbar*** figure shows DMA write channel and read channel datapaths and their connection to the system crossbar.

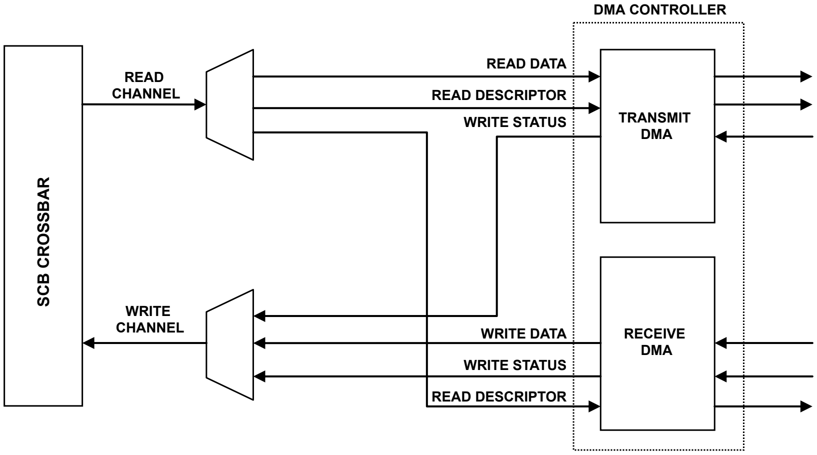


Figure 31-9: EMAC DMA Read/Write channels with System Crossbar

NOTE: Transmit descriptor read and receive descriptor write-back (status update) operations can occur simultaneously. However, transmit descriptor read and write-back operations cannot occur simultaneously. Transmit DMA (or receive DMA) does not initiate the next transfer unless the previous one is complete.

Priority of SCB Requests

The descriptor transfers have higher priority than the data transfers. For example, if there are two bus requests, such as a receive descriptor read and a transmit data read, the receive descriptor read has a higher priority. The next receive data write (subsequent to the receive descriptor read) does not depend on the completion of the transmit data-read transfer.

If there are requests for descriptor reads from both DMA channels, they are serviced based on a first-come first-serve. Receive DMA has higher priority if the descriptor-read requests are generated from both the DMA channels in the same clock cycle. Similarly, in the write channel, descriptor writes from DMA have higher priority than the data-write transfers for the receive DMA.

SCB Interface Programming Options

The SCB bus interface supports the following programmable options for the EMAC module. These options are available using the `EMAC_DMA0_BMMODE` register with the `EMAC_DMA0_BUSMODE` register. These programming options apply to DMA1 and DMA2 as well.

- *Outstanding transactions.* The EMAC-SCB supports up to four outstanding read/write requests on the SCB bus. Software can control these requests by programming the `EMAC_DMA0_BMMODE.WROSRLMT` and `EMAC_DMA0_BMMODE.RDOSRLMT` bits. Maximum outstanding requests = `EMAC_DMA0_BMMODE.WROSRLMT` + 1 (or) `EMAC_DMA0_BMMODE.RDOSRLMT` + 1.
- *Allowed burst sizes.* The allowed burst sizes are 4 (`EMAC_DMA0_BMMODE.BLEN4`), 8 (`EMAC_DMA0_BMMODE.BLEN8`), 16 (`EMAC_DMA0_BMMODE.BLEN16`) and the SINGLE burst. The EMAC-SCB uses only those burst sizes configured by the program (through the `EMAC_DMA0_BMMODE` register) for data transfer through the SCB bus. However, SINGLE burst is available by default, when the `EMAC_DMA0_BMMODE.UNDEF` bit is cleared. Data transfers are restricted to the maximum burst size from this list of programmed burst sizes.
- *Burst splitting and burst selection.* The EMAC-SCB splits the DMA requests into multiple bursts on the SCB system bus. Splitting is based on DMA count and software controllable burst enable bits (shown in the allowed burst sizes) as well as burst types (INCR and INCR_ALIGNED). Burst types are also controllable through the software. SINGLE burst is enabled when the `EMAC_DMA0_BMMODE.UNDEF` bit is not set. Burst length select priority is in the sequence: UNDEF, 16, 8, and 4.
- *INCR burst type*
  - If the `EMAC_DMA0_BMMODE.UNDEF` bit is set, the EMAC-SCB always chooses the maximum allowed burst length based on the `EMAC_DMA0_BMMODE.BLEN16`, `EMAC_DMA0_BMMODE.BLEN8`, `EMAC_DMA0_BMMODE.BLEN4` bits. When the DMA requests are not multiples of the maximum allowed burst length, the SCB can choose a burst-length of any value less than the maximum enabled. (All lesser burst-length enables are redundant). For example, when length bits are enabled and the DMA requests a burst of 42 beats, the SCB splits it into three bursts of 16, 16 and 10 beats respectively.
  - If `EMAC_DMA0_BMMODE.UNDEF` is not enabled, then the burst length is based on the priority of the enabled bits in the following order `EMAC_DMA0_BMMODE.BLEN16`, `EMAC_DMA0_BMMODE.BLEN8`, `EMAC_DMA0_BMMODE.BLEN4`. When the DMA requests a burst transfer, the SCB interface splits the requested bursts into multiple transfers using only the enabled burst lengths. This splitting can occur when the requested burst is not a multiple of the maximum enabled burst. If it cannot choose any of the enabled burst lengths, then it selects the burst length as 1.

  For example, the DMA requests a burst transfer of 42 beats, the SCB interface splits it into multiple bursts of size 16, 16, 8, 1 and 1 beats respectively. (In this case, the allowed burst sizes are enabled and the sequence is in decreasing burst sizes).
- *INCR_ALIGNED burst type.* When the address-aligned burst-type is enabled (`EMAC_DMA0_BMMODE.AAL`), the SCB interface splits the DMA requested bursts. The "INCR Burst Type" section explains burst splitting conditions further. Each burst-size aligns to the least significant bits of the start

address. The SCB interface initially generates smaller bursts so that the remaining transfers move with the maximum (enabled) fixed burst lengths.

For example, in the same setting as explained earlier for `EMAC_DMA0_BMMODE.UNDEF` set, the DMA requests a burst size of 42 beats at the start address of 0x000003A4.(`EMAC_DMA0_BMMODE.BLEN16`, `EMAC_DMA0_BMMODE.BLEN8`, and `EMAC_DMA0_BMMODE.BLEN4` are enabled). The SCB starts the first transfer with size 3 such that the address of the next burst is aligned (0x000003B0) for a burst of 16. Therefore, the sequence of bursts is 3, 16, 16, and 7, respectively.

When `EMAC_DMA0_BMMODE.UNDEF` is not set, then (having a start address of 0x000003A4 with 42 beats), the sequence of burst transfers is 1, 1, 1, 16, 16, 4, and 3 respectively. The sequence of smaller bursts at the beginning is used to align the address to the next higher enabled burst-lengths programmed in the register.

- ***Burst operations for DMA transactions.*** The `EMAC_DMA0_BUSMODE.PBL` (programmable burst length) field indicates the maximum number of beats to transfer in one DMA transaction. This value is also the maximum used in a single block read/write. It is shown in the following table.
  - For example, if `EMAC_DMA0_BUSMODE.PBL`=32 and if `EMAC_DMA0_BMMODE.BLEN16` is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If `EMAC_DMA0_BUSMODE.PBL`=8, and if `EMAC_DMA0_BMMODE.BLEN16` and `EMAC_DMA0_BMMODE.BLEN8` are enabled, the maximum burst is limited to `EMAC_DMA0_BMMODE.BLEN8`. If the `EMAC_DMA0_BUSMODE.PBL8` bit is set, the programmed `EMAC_DMA0_BUSMODE.PBL` value is multiplied by 8 times internally. However, the result cannot be more than the maximum limits specified.
  - Set the `EMAC_DMA0_BUSMODE.USP` bit to make the receive DMA burst length configuration independent of the transmit DMA configuration. When this bit is set, the EMAC uses the `EMAC_DMA0_BUSMODE.RPBL` bits to define the burst length of receive DMA. If the `EMAC_DMA0_BUSMODE.USP` bit is not set, the `EMAC_DMA0_BUSMODE.RPBL` bits are used for both transmit and receive. Programs must ensure that the PBL maximum limit is not violated.
  - The receive and transmit descriptors are always accessed in the maximum burst-size for the 16-bytes to be read (PBL-max limit is (TX or RX FIFO size/2)/4 words. (PBL maximum for transmit and receive limits burst-size).

Table 31-11: DMA PBL Max Limits

Burst Limit Max Term	Definition
PBL-maximum limit	(FIFO size/2)/4 words
PBL-maximum limit (transmit)	2048 bytes/2 /4 = 256 words
PBL-maximum limit (receive)	2048 bytes/2 /4 = 256 words

DMA Bursts Using the SCB Interface

The transmit DMA initiates a data transfer when sufficient space to accommodate the configured burst is available in the transmit FIFO. Or, the transmit DMA initiates a data transfer when the number of bytes until the end of frame is less than the configured burst-length. The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR4/8/16 and 1 beat transaction.

The receive DMA initiates a data transfer when sufficient data to accommodate the configured burst is available in the MTL receive FIFO. Or, the receive DMA initiates a data transfer when the end of frame is detected in the receive FIFO. For example, when the amount is less than the configured burst-length. The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR 4, 8, 16 or 1 beat transaction. If the end-of frame is reached before the fixed-burst ends on the SCB interface, then dummy transfers are performed to complete the fixed-burst. Otherwise (if `EMAC_DMA0_BUSMODE.FB` is reset), it transfers data using INCR (undefined length) transactions.

When the SCB interface is configured for address-aligned beats, both DMA engines ensure that the first burst transfer is less than or equal to the configured PBL size. (The address-aligned beats configuration uses the `EMAC_DMA0_BUSMODE.AAL` bit). Therefore, all subsequent beats start at an address that is aligned to the configured PBL.

SCB Bus Transaction Status

The SCB uses the `EMAC_DMA0_BMSTAT.BUSRD` and `EMAC_DMA0_BMSTAT.BUSWR` bits to indicate whether the channel is active or not.

Fatal Bus Error

The EMAC SCB asserts the error interrupt (`EMAC_DMA0_STAT.FBI`) when the corresponding fatal bus error interrupt is enabled in the DMA interrupt enable register. The application must reset the core to restart the DMA.

DMA Controller (EMAC DMA)

The EMAC has a built-in DMA controller that performs reads and writes of application data and descriptors through the SCB master interface.

The DMA controller has independent transmit and receive engines, and a CSR (control and status register) space. The transmit engine transfers data from system memory to a FIFO, while the receive engine transfers data from the FIFO to the system memory. The controller uses a descriptor chain-based transfer mechanism to move data efficiently from source to destination with minimal processor core intervention. The DMA is specially designed for packet-oriented data transfers such as Ethernet frames. The controller can be programmed to interrupt the application for situations such as frame transmit and receive transfer completion, and other normal or abnormal conditions.

The DMA and the application device driver communicate through two internal data structures:

1. DMA control and status registers (CSR).
2. Data buffers and descriptor lists. Descriptor lists operate in ring mode and chain mode, as shown in the ***EMAC DMA Descriptor Models*** figure.

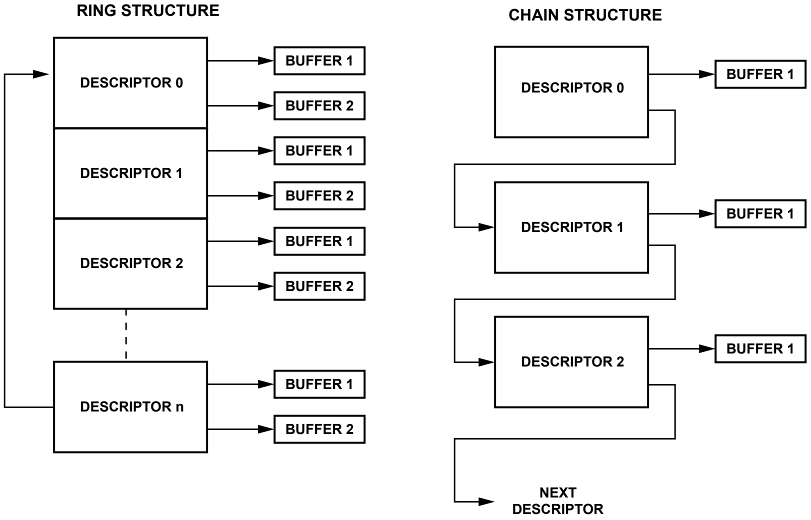


Figure 31-10: EMAC DMA Descriptor Models

Descriptors that reside in the application memory act as pointers to receive and transmit buffers. Descriptors have the following extra attributes.

- There are two descriptor lists, one for receive, and one for transmit. The base address of each list is written into the address registers of the receive and transmit descriptor lists respectively.
- A descriptor list is forward linked (either implicitly or explicitly). The last descriptor can point back to the first entry to create a ring structure.
- Explicit chaining of descriptors is accomplished by setting the second address chained in both receive and transmit descriptors.
- The descriptor lists reside in the address space of the application memory.
- Each descriptor can point to a maximum of two buffers. This attribute enables two buffers, physically addressed, rather than contiguous buffers in memory.

A data buffer resides in the physical memory space of the application. It consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers can contain only data. The descriptor maintains buffer status. *Data chaining* refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA skips to the next frame buffer when the end-of-frame is detected. Data chaining is enabled or disabled.

NOTE: It is possible to define a skip length (in terms of N × 32-bit words) between two subsequent descriptors, when using ring mode. Program the `EMAC_DMA0_BUSMODE.DSL`/`EMAC_DMA1_BUSMODE.DSL`/`EMAC_DMA2_BUSMODE.DSL` field to enable this attribute. With this option available, programs are not always restricted to a contiguous memory location in ring mode.

DMA Related Registers

The *Summary of DMA Related Registers* table provides a summary of DMA registers relative to their function. Refer to the “Register Descriptions” sections for complete bit descriptions of each of these registers.

Table 31-12: Summary of DMA Related Registers

Register Name	Description
Bus Mode[*1]	Establishes the bus operating modes for the DMA based on the SCB master interface.
Transmit Poll Demand	Enables the transmit DMA to check whether the DMA owns the current descriptor. The transmit poll demand command wakes up the TxDMA when it is in suspend mode. The TxDMA can go into suspend mode because of an underflow error in a transmitted frame or because of the unavailability of descriptors owned by transmit DMA. Issue this command anytime and the TxDMA resets this command once it starts refetching the current descriptor from host memory.
Receive Poll Demand	Enables the receive DMA to check for new descriptors. This command wakes up the RxDMA from the SUSPEND state. The RxDMA can go into SUSPEND state only because of the unavailability of descriptors owned by it.
Receive Descriptor List Address	Points to the start of the receive descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (32- bit data bus). The DMA internally converts the descriptor list to a bus width aligned address by making the corresponding LSBs low.
Transmit Descriptor List Address	Points to the start of the transmit descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (for 32-bit data bus). The DMA internally converts it to bus width aligned address by making the corresponding LSB to low.
DMA Status	Contains all the status bits that the DMA reports to the application. The software driver reads this register during an interrupt service routine or during polling. Most of the fields in this register interrupt the host.
Operation Mode	Establishes the transmit and receive operating modes and commands. The operation mode register is the last control register written as part of DMA initialization.
Interrupt Enable	Enables the interrupts reported by DMA status register. After a hardware or software reset, all interrupts are disabled.
Missed Frame and Buffer Overflow Counter	The DMA maintains two counters to track the number of missed frames during reception. This register reports the current value of the counter, which is used for diagnostic purposes.
Receive Interrupt Watchdog Timer	When written with non-zero value, enables the watchdog timer for receive interrupt (RI) in the DMA status register.
SCB Bus Mode	Controls the SCB interface master behavior. It controls the burst splitting and the number of outstanding requests.
SCB Status	Provides the active status of the SCB interface read and write channels.
Current Host Transmit Descriptor	Points to the start address of the current transmit descriptor read by the DMA.
Current Host Receive Descriptor	Points to the start address of the current receive descriptor read by the DMA.
Current Host Transmit Buffer Address	Points to the current transmit buffer address the DMA is reading.
Current Host Receive Buffer Address	Points to the current receive buffer address the DMA is reading.

[*1] Do not write to the `EMAC_DMA0_BUSMODE.DSL`/`EMAC_DMA1_BUSMODE.DSL`/`EMAC_DMA2_BUSMODE.DSL` registers until the first write updates. Otherwise, the second write operation does not update properly. For correct operation, the delay between two writes to the same register location must be at least 8 cycles of 50 MHz RMII REFCLK.

Table 31-13: DMA Registers with Consecutive Writes

Registers with Implications for Consecutive Writes
DMA Bus Mode

DMA Descriptors

The DMA module in the Ethernet subsystem transfers data based on a linked list of descriptors. The descriptor addresses must be aligned to the 32-bit bus width. The descriptors can be either 4 x 32-bit words (16 bytes) or 8 x 32-bit words (32 bytes). Configure the controller for the appropriate word length using the DMA bus mode register. The descriptor words are numbered from 0 to 7 for both the transmit and receive engine.

Typical factors for deciding the descriptor word size are as follows:

- When the time stamping or receive checksum engines are not enabled, the extended descriptors are not required. The software can use descriptors with the default size of 16 bytes (4 words).
- When the time stamping feature is enabled, the software must allocate 32 bytes (8 words) of memory for every descriptor. (The time stamping feature is used with the IEEE 1588 PTP engine).
- When only the receive checksum offload is enabled (time stamping disabled), software must allocate 32 bytes (8 words) of memory for every descriptor. However, only word 4 of the extended words (descriptors 4–7) contains the required status information. Treat the rest of the extended words as reserved or dummy.

Transmit Descriptor

The *Transmit Descriptor Words* figure shows the transmit descriptor structure in memory. The application software must program the TDES0 control bits during descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits. The following tables give the contents of the transmitter descriptor word 0 (TDES0) through word 7 (TDES7).

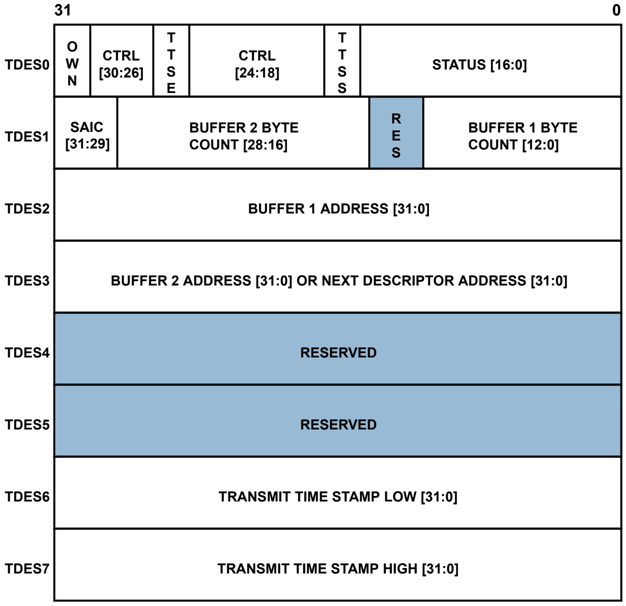

31
0
TDES0
OWN
CTRL [30:26]
TTSE
CTRL [24:18]
TTSS
STATUS [16:0]
TDES1
SAIC [31:29]
BUFFER 2 BYTE COUNT [28:16]
RES
BUFFER 1 BYTE COUNT [12:0]
TDES2
BUFFER 1 ADDRESS [31:0]
TDES3
BUFFER 2 ADDRESS [31:0] OR NEXT DESCRIPTOR ADDRESS [31:0]
TDES4
RESERVED
TDES5
RESERVED
TDES6
TRANSMIT TIME STAMP LOW [31:0]
TDES7
TRANSMIT TIME STAMP HIGH [31:0]

Figure 31-11: Transmit Descriptor Words

Table 31-14: Transmit Descriptor Fields (TDES0)

Bit	Name	Description
31	OWN	Ownership. When set, this bit indicates that the DMA owns the descriptor. When this bit is reset, it indicates that the application owns the descriptor. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the first descriptor of the frame must be set after all subsequent descriptors belonging to the same frame have been set. This configuration avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
30	IC	Interrupt on Completion. When set, this bit sets the transmit interrupt (DMA status register [0]) after the present frame is transmitted.
29	LS	Last Segment. When set, this bit indicates that the buffer contains the last segment of the frame.
28	FS	First Segment. When set, this bit indicates that the buffer contains the first segment of a frame.
27	DC	Disable CRC. When this bit is set, the EMAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This functionality is valid only when the first segment (TDES0[28]) is set.
26	DP	Disable Pad. When set, the EMAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes. The CRC field is added despite the state of the DC (TDES0[27]) bit. This functionality is valid only when the first segment (TDES0[28]) is set.
25	TTSE	Transmit Time Stamp Enable. When set, this bit enables IEEE1588 hardware time stamping for the transmit frame referenced by the descriptor. This field is valid only when the first segment control bit (TDES0[28]) is set.
24	CRCR	CRC Replacement Control. When set, the EMAC replaces the last four bytes of the transmitted packet with recalculated CRC bytes. The host should ensure that the CRC bytes are present in the frame being transferred from the transmit buffer. This bit is valid when the first segment bit (TDES0[28]) and the disable CRC bit (TDES0[27]) are set.
24	Reserved	
23:22	CIC	Checksum Insertion Control. These bits control the checksum calculation and insertion. Bit encodings are as follows: 00 = Checksum Insertion disabled. 01 = Only IP header checksum calculation and insertion are enabled. 10 = IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. 11 = IP header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.
21	TER	Transmit End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.

Table 31-14: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
20	TCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When TDES0[20] bit is set, TBS2 (TDES1[28:16]) is a ***do-not-care*** value. TDES0[21] takes precedence over TDES0[20].
19:18	Reserved	
19:18	VLIC	VLAN Insertion Control. When set, these bits request the MAC to perform VLAN tagging or untagging before transmitting the frames. If the frame is modified for VLAN tags, the MAC automatically recalculates and replaces the CRC bytes. Bit encodings are as follows. 00 = Do not add a VLAN tag. 01 = Remove the VLAN tag from the frames before transmission. This option should be used only with the VLAN frames. 10 = Insert a VLAN tag with the tag value programmed in the VLAN tag inclusion or replacement `EMAC_VLAN_INCL` register. 11 = Replace the VLAN tag in frames with the tag value programmed in the VLAN tag inclusion or replacement `EMAC_VLAN_INCL` register. This option should be used only with the VLAN frames. These bits are valid when the first segment bit (TDES0[28]) is set.
17	TTSS	Transmit Time Stamp Status. This bit is a status bit to indicate that a time stamp is captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a time stamp value captured for the transmit frame. This field is only valid when the last segment control bit of the descriptor (TDES0[29]) is set.
16	IHE	IP Header Error. When set, this bit indicates that the EMAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application. It indicates an error status when there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet length or type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the header length field has a value less than 0x5.
15	ES	Error Summary. Indicates the logical OR of the following bits: TDES0[14] = Jabber Timeout TDES0[13] = Frame Flush TDES0[11] = Loss of Carrier TDES0[10] = No Carrier TDES0[9] = Late Collision TDES0[8] = Excessive Collision TDES0[2] = Excessive Deferral TDES0[1] = Underflow Error TDES0[16] = IP Header Error TDES0[12] = IP Payload Error

Table 31-14: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
14	JT	Jabber Timeout. When set, this bit indicates that the EMAC transmitter has experienced a jabber timeout. This bit is only set when the `EMAC_MACCFG.JB` bit is not set.
13	FF	Frame Flushed. When set, this bit indicates that the DMA or MFL flushed the frame due to a software flush command given by the CPU.
12	IPE	IP Payload Error. When set, this bit indicates that EMAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application. It issues an error status when there is a mismatch.
11	LC	Loss of Carrier. When set, this bit indicates a loss of carrier occurred during frame transmission. This functionality is valid only for the frames transmitted without collision when the EMAC operates in half-duplex mode. Loss of Carrier. When set, this bit indicates that the EMAC aborted the frame transmission because of a collision occurring after the collision window (64 byte-times, including preamble, in RMII mode; 512 byte-times, including preamble and carrier extension, in RGMII mode). This bit is not valid if the underflow error bit is set.
10	NC	No Carrier. When set, this bit indicates that the carrier sense signal from the PHY did not assert during transmission.
9	LC	Late Collision. When set, this bit indicates that the EMAC aborted the frame transmission because of a collision occurring after the collision window (64 byte-times, including preamble). This bit is not valid if the underflow error bit is set. Late Collision. When set, this bit indicates that the EMAC aborted the frame transmission because of a collision occurring after the collision window (64 byte-times, including preamble, in RMII mode; 512 byte-times, including preamble and carrier extension, in RGMII mode). This bit is not valid if the underflow error bit is set.
8	EC	Excessive Collision. When set, this bit indicates that the EMAC aborted the transmission after 16 successive collisions, while attempting to transmit the current frame. If the `EMAC_MACCFG.DR` disable retry bit is set, this bit is set after the first collision, and the transmission of the frame aborts.
7	VF	VLAN Frame. When set, this bit indicates that the transmitted frame was a VLAN-type frame.
6:3	CC	Collision Count. When set, these bit indicate that the EMAC aborted the frame transmission due to a collision occurring after the collision window (64 byte-times, including preamble). This bit is not valid if the Underflow Error bit is set. This field is updated only in half-duplex mode.
6:3	SLOTNUM	Slot Number Control Bits. In AV mode, these bits indicate the slot interval in which the data should be fetched from the corresponding buffers, addressed by TDES2 or TDES3. When the transmit descriptor is fetched, the DMA compares the slot number value in this field with the slot function control and status register (RSN). It fetches the data from the buffers only if there is a match in values. These bits are valid only for AV channels (not channel 0).
2	ED	Excessive Deferral. When set, this bit indicates that the transmission has ended because of excessive deferral when the `EMAC_MACCFG.DC` deferral check bit is set high. Excessive deferral is

Table 31-14: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
		over 24,288-bit times. (155,680-bits times in 1,000-Mbps mode or when jumbo frame is enabled).
1	UF	Underflow Error. When set, this bit indicates that the EMAC aborted the frame because data arrived late from the application memory. Underflow error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the suspended state and sets both transmit underflow (`EMAC_DMA0_STAT.UNF`) and transmit interrupt (`EMAC_DMA0_STAT.TI`) bits.
0	DB	Deferred Bit. When set, this bit indicates that the EMAC defers before transmission because of the presence of carrier. This bit is valid only in half-duplex mode.

Table 31-15: Transmit Descriptor Word 1 (TDES1)

Bit	Name	Description
31-29	Reserved	
31–29	SAIC	Source Address Insertion Control. Request the MAC to add or replace the source address field in the Ethernet frame with the value given in the MAC address register 0 or MAC address register 1. If the source address field is modified in a frame, the MAC automatically recalculates and replaces the CRC bytes. SAIC[2] chooses between MAC address register 0 and MAC address register 1 for source address insertion or replacement. The following list describes SAIC[1:0]. 00 = Do not include the source address. 01 = Include or insert the source address. For reliable transmission, the application must provide frames without source addresses. 10 = Replace the source address. For reliable transmission, the application must provide frames with source addresses. 11 = Reserved. This field is valid only when the first segment bit (TDES0[28]) is set.
28–16	TBS2	Transmit Buffer 2 Size. These bits indicate the second data buffer size in bytes. This field is not valid if TDES0[20] is set.
15–13	Reserved	
12–0	TBS1	Transmit Buffer 1 Size. These bits indicate the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

Table 31-16: Transmit Descriptor 2 (TDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There is no limitation on the buffer address alignment

Table 31-17: Transmit Descriptor 3 (TDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	Indicates the physical address of buffer 2 when DMA uses a descriptor ring structure. If the second address chained (TDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. The buffer address pointer must align to the bus width only when TDES1[24] is set. LSBs are ignored internally.

Table 31-18: Transmit Descriptor 6 (TDES6)

Bit	Name	Description
31–0	TTSL	Transmit Frame Time Stamp Low. The DMA updates this field with the least significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the last segment bit (LS) in the descriptor is set and time stamp status (TTSS) bit is set.

Table 31-19: Transmit Descriptor 7 (TDES7)

Bit	Name	Description
31–0	TTSH	Transmit Frame Time Stamp High. The DMA updates this field with the most significant 32 bits of the time stamp captured for the corresponding receive frame. This field has the time stamp only if the last segment bit (LS) in the descriptor is set and time stamp status (TTSS) bit is set.

DMA Transmit Process

The following sections describe how the transmission process works for direct memory access on the EMAC controller.

- Default (Non-OSF) Mode
- OSF Mode Enabled
- Transmit Frame Processing
- Transmit Polling Suspended

Default (Non-OSF) Mode

The following sequence decribes the default process for DMA transmit works. The sequence applies DMA1 and DMA2 as well.

1. The application sets up the transmit descriptor (using TDES0- TDES3) and sets the OWN bit (TDES0) after setting up the corresponding data buffers with Ethernet frame data.
2. Once the `EMAC_DMA0_OPMODE.ST` bit is set, the DMA enters the run state.
3. While in the run state, the DMA polls the transmit descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the application, or if an error condition occurs, transmission suspends. Both the transmit buffer unavailable (`EMAC_DMA0_STAT.TU`) and normal interrupt summary (`EMAC_DMA0_STAT.NIS`) bits are set. The transmit engine proceeds to Step 9.

4. If the acquired descriptor is flagged as owned by DMA (TDES0 [31] = 1#b1), the DMA decodes the transmit data buffer address from the acquired descriptor.
5. The DMA fetches the transmit data from the application memory and transfers the data to the MFL for transmission.
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps 3, 4, and 5 repeat until the end-of-Ethernet-frame data transfers to the MFL.
7. Frame transmission completes. If IEEE 1588 time stamping was enabled for the frame, the time stamp value obtained from MFL is written to the transmit descriptor (TDES2 and TDES3) that contains the end-of-frame buffer. (The transmit status indicates if IEEE 1588 time stamping enables). The status information is then written to this transmit descriptor (TDES0). Because the OWN bit is cleared during this step, the application now owns this descriptor. If time stamping was not enabled for this frame, the DMA does not alter the contents of TDES2 and TDES3.
8. Transmit interrupt (`EMAC_DMA0_STAT.TI`) is set after completing transmission of a frame. The frame has interrupt on completion (TDES1 [31]) set in its last descriptor. The DMA engine then returns to Step 3.
9. In the suspend state, the DMA tries to reacquire the descriptor (and returns to Step 3) when it receives a transmit poll demand and the `EMAC_DMA0_STAT.UNF` bit is cleared.

NOTE: If the `EMAC_DMA0_OPMODE.OSF` bit is not set, the actual inter frame gap (IFG) is more than the value programmed in the `EMAC_MACCFG` register.

OSF Mode Enabled

While in the run state, the transmit process can simultaneously acquire two frames without closing the status descriptor of the first (if the `EMAC_DMA0_OPMODE.OSF` bit is set). As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the status information of the first frame.

In OSF mode, the run state transmit DMA operates in the following sequence.

1. The DMA operates as described in steps 1–6 of Default (Non-OSF) Mode.
2. Without closing the previous last descriptor of the frame, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into suspend mode and skips to Step 7.
4. The DMA fetches the transmit frame from the application memory and transfers the frame to the MFL until the end-of-frame data is transferred. It closes the intermediate descriptors when this frame splits across multiple descriptors.
5. The DMA waits for the frame transmission status and time stamp of the previous frame. Once the status is available, the DMA writes the time stamp to TDES2 and TDES3, if the time stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared OWN bit, to the corresponding TDES0, thus

closing the descriptor. If time stamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3.

6. If enabled, the transmit interrupt is set; the DMA fetches the next descriptor, and then proceeds to Step 3 (when status is normal). If the previous transmission status shows an underflow error, the DMA goes into suspend mode (Step 7).
7. In suspend mode, if a pending status and time stamp are received from the MFL, the DMA writes the time stamp (if enabled for the current frame) to TDES2 and TDES3. The DMA then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to suspend mode.
8. After receiving a transmit poll demand (`EMAC_DMA0_TXPOLL`), the DMA can exit suspend mode and enter the run state. (Go to Step 1 or Step 2 depending on pending status)

NOTE: If the `EMAC_DMA0_OPMODE.OSF` bit is set, the DMA fetches the next descriptor in advance of closing the current descriptor. Therefore, the descriptor chain must have more than two different descriptors for proper operation.

NOTE: If the `EMAC_DMA0_OPMODE.OSF` bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to update. In the meantime, the MFL receives the second frame into the FIFO while transmitting the first frame. The difference in cycles is not seen for the first descriptor, because the time taken for the complete descriptor processing remains the same whether `EMAC_DMA0_OPMODE.OSF` is set or not. The difference appears only for the following descriptor because its processing began earlier.

Transmit Frame Processing

The transmit DMA engine expects that the data buffers contain complete Ethernet frames, excluding: preamble, pad bytes, and FCS fields. The destination address, source address, and type or length fields contain valid data. If the transmit descriptor indicates that the EMAC CORE must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes.

Frames can be data-chained and can span several buffers. Frames must be delimited by the first descriptor (TDES0[28]) and the last descriptor (TDES0[29]), respectively.

As transmission starts, the first descriptor must have (TDES0[28]) set. Frame data transfers from the application buffer to the transmit FIFO. Concurrently, if last descriptor (TDES0[29]) of the current frame clears, the transmit process attempts to acquire the next descriptor. The transmit process expects this descriptor to have TDES0[28] clear. If TDES1[29] is clear, it indicates an intermediary buffer. If TDES0[29] is set, it indicates the last buffer of the frame.

After the last buffer of the frame has been transmitted, the DMA writes back the final status information. The DMA writes to the transmit descriptor 0 (TDES0) word of the descriptor that has the last segment set in transmit descriptor 0 (TDES0[29]). Now, if interrupt-on-completion (TDES0[30]) is set, the transmit interrupt (DMA_STAT [0]) is set, the next descriptor is fetched, and the process repeats.

Actual frame transmission begins after the MFL transmit FIFO has reached either a programmable transmit threshold (`EMAC_DMA0_OPMODE.TTC`), or a full frame is contained in the FIFO. There is also an option for store-and-forward mode (`EMAC_DMA0_OPMODE.TSF`). Descriptors are released (`OWN` bit TDES0 [31] clears) when the DMA finishes transferring the frame.

Transmit Polling Suspended

Either of the following conditions suspends transmit polling:

1. The DMA detects a descriptor owned by the application (TDES0 [31] = 0).
2. A frame transmission aborts when a transmit error due to underflow is detected. The appropriate transmit descriptor 0 (TDES0) bit is set.

If the second condition occurs, both of the abnormal interrupt summary ([15]) and transmit underflow bits ([5]) are set. The information is written to transmit descriptor 0, causing the suspension. If the DMA goes into a SUSPEND state due to the first condition, then both `EMAC_DMA0_STAT.NIS` and `EMAC_DMA0_STAT.TU` are set.

In both cases, the position in the transmit list is retained. The retained position is that of the descriptor following the last descriptor closed by the DMA.

The driver must explicitly issue a transmit poll demand command after rectifying the suspension cause. If the first condition occurs, the driver must give descriptor ownership to the DMA and then issue a poll demand command to resume the transfer.

Receive Descriptor

The ***Receive Descriptor Words*** figure shows the structure of the receive descriptor. It can have 32 bytes of descriptor data (8 DWORDs) when advanced time stamping or checksum is enabled.

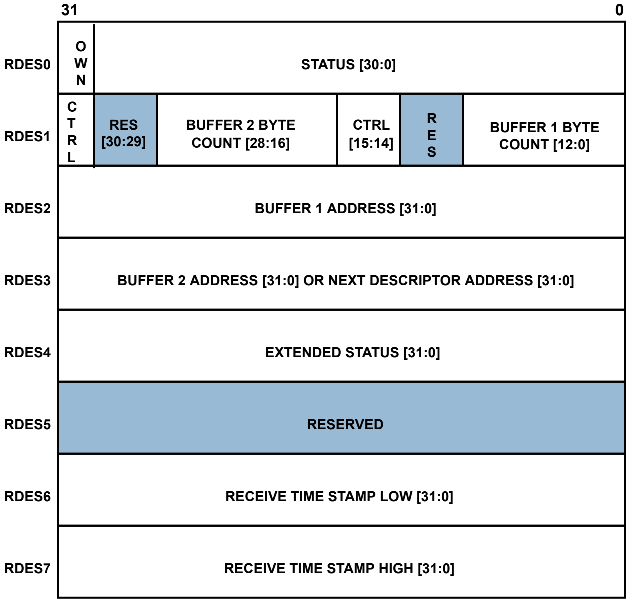

31
0
RDES0
OWN
STATUS [30:0]
RDES1
CTRL
RES [30:29]
BUFFER 2 BYTE COUNT [28:16]
CTRL [15:14]
RES
BUFFER 1 BYTE COUNT [12:0]
RDES2
BUFFER 1 ADDRESS [31:0]
RDES3
BUFFER 2 ADDRESS [31:0] OR NEXT DESCRIPTOR ADDRESS [31:0]
RDES4
EXTENDED STATUS [31:0]
RDES5
RESERVED
RDES6
RECEIVE TIME STAMP LOW [31:0]
RDES7
RECEIVE TIME STAMP HIGH [31:0]

Figure 31-12: Receive Descriptor Words

Table 31-20: Receive Descriptor Fields (RDES0)

Bit	Name	Description
31	OWN	Ownership. When set, this bit indicates that the DMA of the EMAC subsystem owns the descriptor. When this bit is reset, this bit indicates that the application owns the descriptor. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	AFM	Destination Address Filter Fail. When set, this bit indicates a frame that failed in the DA filter in the EMAC CORE.
29–16	FL	Frame Length. These bits indicate the byte length of the received frame that transferred to application memory (including CRC). This field is valid when last descriptor (RDES0[8]) is set and either the descriptor error (RDES0[14]) or overflow error bits are reset. This field is valid when last descriptor (RDES0[8]) is set. When the last descriptor and error summary bits are not set, this field indicates the accumulated number of bytes that transferred for the current frame.
15	ES	Error Summary. Indicates the logical OR of the following bits. RDES0[1] = CRC Error RDES0[3] = RGMII Receive Error RDES0[4] = Watchdog Timeout RDES0[6] = Late Collision RDES0[7] = Time Stamp Available RDES4[4:3] = IP Header/Payload Error RDES0[11] = Overflow Error RDES0[14] = Descriptor Error. This field is valid only when the last descriptor (RDES0[8]) is set.
14	DE	Descriptor Error. When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers. The DMA does not own the next descriptor. The frame is truncated. This field is valid only when the last descriptor (RDES0[8]) is set.
13	Reserved	
13	SAF	Source Address Filter Fail. When set, this bit indicates that the SA field of frame failed the SA filter in the EMAC Core.
12	LE	Length Error. When set, this bit indicates that the actual length of the frame received and that the length or type field does not match. This bit is valid only when the frame type (RDES0[5]) bit is reset.
11	OE	Overflow Error. When set, this bit indicates that the received frame is damaged due to buffer overflow in MFL.
10	VLAN	VLAN Tag. When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the EMAC CORE.
9	FS	First Descriptor. When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next descriptor contains the beginning of the frame.
8	LS	Last Descriptor. When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame

Table 31-20: Receive Descriptor Fields (RDES0) (Continued)

Bit	Name	Description
7	Time Stamp Available	When set, this bit indicates that a snapshot of the time stamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This functionality is valid only when the last descriptor bit (RDES0[8]) is set
6	LC	Late Collision. When set, this bit indicates that a late collision has occurred while receiving the frame in half-duplex mode.
5	FT	Frame Type. When set, this bit indicates that the receive frame is an Ethernet-type frame (the LT field is greater than or equal to 16'h0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for runt frames less than 14 bytes.
4	RWT	Receive Watchdog Timeout. When set, this bit indicates that the receive watchdog timer has expired while receiving the current frame. The current frame is truncated after the watchdog timeout.
3	Reserved	
3	RE	Receive Error. When set, this bit indicates that the RGMII PHY sent RGMII RXERR on RXCTL pin during frame reception. This error also includes the carrier extension error in RGMII and half-duplex mode.
2	DE	Dribble Bit Error. When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles).
1	CE	CRC Error. When set, this bit indicates that a Cyclic Redundancy Check (CRC) error occurred on the received frame. This field is valid only when the last descriptor (RDES0[8]) is set.
0	Extended Status Available	When set, this bit indicates that the extended status is available in descriptor word 4 (RDES4). This functionality is valid only when the last descriptor bit (RDES0[8]) is set.

Table 31-21: Receive Descriptor Fields 1 (RDES1)

Bit	Name	Description
31	DIC	Disable Interrupt on Completion. When set, this bit prevents setting the `EMAC_DMA0_STAT.RI` bit of the status register for the received frame ending in the buffer indicated by this descriptor. This activity, in turn, disables the assertion of the interrupt to the application due to RI for that frame.
30–29	Reserved	
28–16	RBS2	Receive Buffer 2 Size. These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4 (32-bit bus), even if the value of RDES3 (buffer2 address pointer) does not align to bus width. If the buffer size is not an appropriate multiple of 4, 8, or 16, the resulting behavior is undefined. This field is not valid if RDES1[14] is set.
15	RER	Receive End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.
14	RCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a *do-not-care* value. RDES1[15] takes precedence over RDES1[14].
13	Reserved	

Table 31-21: Receive Descriptor Fields 1 (RDES1) (Continued)

Bit	Name	Description
12–0	RBS1	Receive Buffer 1 Size. Indicates the size of the first data buffer in bytes. The buffer size must be a multiple of 4 (32-bit bus), even if the value of RDES2 (buffer1 address pointer) is not aligned. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses buffer 2 or next descriptor depending on the value of RCH (Bit 14).

Table 31-22: Receive Descriptor Fields 2 (RDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value stores the start of frame. The DMA performs a write operation with the RDES2[1:0] bits as 0 during the transfer of the start of frame. However, the frame data shifts per the address pointer of the actual buffer. The DMA ignores RDES2[1:0] when the address pointer is to a buffer where the middle or last part of the frame is stored. (RDES2[1:0] corresponds to a bus width of 32).

Table 31-23: Receive Descriptor Fields 3 (RDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	These bits indicate the physical address of buffer 2 when DMA uses a descriptor ring structure. If the second address chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. If RDES1[24] is set, the buffer (next descriptor) address pointer must be bus width-aligned (RDES3[1:0] = 0, corresponding to a bus width of 32. LSBs are ignored internally.) However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value stores the start of frame. The DMA ignores RDES3[1:0] when the address pointer is to a buffer where the middle or last part of the frame is stored. (RDES3[1:0] corresponds to a bus width of 32.)

Table 31-24: Receive Descriptor Fields 4 (RDES4)

Bit	Name	Description
31-26	Reserved	
25	Layer 4 Filter Match	When set, this bit indicates that the received frame matches layer 4 filter fields. This status is given only when one of the following conditions is true: • Layer 3 fields are not enabled and all enabled layer 4 fields match. • All enabled layer 3 and layer 4 filter fields match.
24	Layer 3 Filter Match	When set, this bit indicates that the received frame matches layer 3 IP address fields. This status is given only when one of the following conditions is true: • All enabled layer 3 fields match and all enabled layer 4 fields are bypassed. • All enabled filter fields match.

Table 31-24: Receive Descriptor Fields 4 (RDES4) (Continued)

Bit	Name	Description
23-21	Reserved	
20-18	VLAN Tag Priority	These bits give the VLAN tag's user value in the received packet. These bits are valid only when the RDES4[16] and RDES4[17] are set.
17	AV Tagged Packet Received	When set, this bit indicates that an AV tagged packet is received. Otherwise, this bit indicates that an untagged AV packet is received. This bit is valid when RDES4[16] is set.
16	AV Packet Received	When set, this bit indicates that an AV packet is received.
15	Reserved	
14	Timestamp Dropped	When set, this bit indicates that the time stamp is captured for this frame but dropped in the MFL RxFIFO because of overflow.
13	PTP Version	When set, this bit indicates that the received PTP message has the IEEE 1588 version 2 format. When reset, it has the version 1 format. This description is valid only if the message type (RDES4[11:8]) is non-zero.
12	PTP Frame Type	When set, this bit indicates that the PTP message transfers directly over Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message transfers over UDP-IPv4 or UDP-IPv6. Bits 6 and 7 have the information on IPv4 or IPv6.
11–8	Message Type	These bits are encoded to give the type of the message received. 0000 = No PTP message received 0001 = SYNC (all clock types) 0010 = Follow_Up (all clock types) 0011 = Delay_Req (all clock types) 0100 = Delay_Resp (all clock types) 0101 = Pdelay_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock) 0110 = Pdelay_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock) 0111 = Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock) 1xxx - Reserved
7	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet.
6	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet.
5	IP Checksum Bypassed	When set, this bit indicates that the checksum offload engine is bypassed.
4	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum that the core calculated does not match the corresponding checksum field in the received segment. (For example: the TCP, UDP, or ICMP checksum). The bit is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP header field.

Table 31-24: Receive Descriptor Fields 4 (RDES4) (Continued)

Bit	Name	Description
3	IP Header Error	When set, this bit indicates that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes. Or, it indicates that the IP datagram version is not consistent with the Ethernet type value.
2–0	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the receive checksum offload engine (COE). The COE also sets these bits to 2'b00 when it does not process the payload of the IP datagram. It does not process the payload because of an IP header error or fragmented IP. 000 = Unknown or did not process IP payload 001 = UDP 010 = TCP 011 = ICMP 1xx = Reserved

Table 31-25: Extended Receive Descriptor Fields 4 (RDES4)

Bit	Name	Description
31-26	Reserved	
25	Layer 4 Filter Match	When set, this bit indicates that the received frame matches layer 4 filter fields. This status is given only when one of the following conditions is true: • Layer 3 fields are not enabled and all enabled layer 4 fields match. • All enabled layer 3 and layer 4 filter fields match.
24	Layer 3 Filter Match	When set, this bit indicates that the received frame matches layer 3 IP address fields. This status is given only when one of the following conditions is true: • All enabled layer 3 fields match and all enabled layer 4 fields are bypassed. • All enabled filter fields match.
23-21	Reserved	
20-18	VLAN Tag Priority	These bits give the VLAN tag’s user value in the received packet. These bits are valid only when the RDES4[16] and RDES4[17] are set.
17	AV Tagged Packet Received	When set, this bit indicates that an AV tagged packet is received. Otherwise, this bit indicates that an untagged AV packet is received. This bit is valid when RDES4[16] is set.
16	AV Packet Received	When set, this bit indicates that an AV packet is received.
15	Reserved	
14	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for this frame but got dropped in the MFL RxFIFO because of overflow.

Table 31-26: Receive Descriptor Fields 6 (RDES6)

Bit	Name	Description
31–0	RTSL	Receive Frame Time Stamp Low. The DMA updates this field with the least significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame. The status bit (RDES0[8]) indicates the last descriptor.

Table 31-27: Receive Descriptor Fields 7 (RDES7)

Bit	Name	Description
31–0	RTSH	Receive Frame Time Stamp High. The DMA updates this field with the most significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame. The status bit (RDES0[8]) indicates the last descriptor.

EMAC DMA Receive Process

The following sections describe how the receive process for direct memory access works on the EMAC controller.

- Receive Frame Processing
- Receive Descriptor Acquisition
- Receive Process Suspended

The reception process for DMA works as follows:

1. The application sets up receive descriptors (RDES0–RDES3) and sets the OWN bit (RDES0 [31]).
2. Once the `EMAC_DMA0_OPMODE.SR` bit is set, the DMA enters the run state. While in the run state, the DMA attempts to acquire free descriptors by polling the receive descriptor list. If the fetched descriptor is not free (the application owns the descriptor), the DMA enters the suspend state and jumps to Step 9.
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the data buffers of the acquired descriptor.
5. When the buffer is full or the frame transfer is complete, the receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to Step 7. If IEEE 1588 time stamping is enabled, the DMA writes the time stamp (if available) to the current descriptor. If the DMA does not own the next fetched descriptor and the frame transfer is not complete, the DMA sets the descriptor error bit in the RDES0. (The bit is set unless flushing is disabled.) The DMA closes the current descriptor (clears the OWN bit). The DMA marks it as intermediate by clearing the last segment (LS) bit in the RDES0 value (marks it as last descriptor if flushing is not disabled). The DMA then proceeds to Step 8. If the DMA owns the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to Step 4.
7. If IEEE 1588 time stamping is enabled, the DMA writes the time stamp (if available) to RDES2 and RDES3 of the current descriptor. The DMA then takes the status of the receive frame from the MFL and writes the status word to RDES0 of the current descriptor. The OWN bit is cleared and the last segment bit is set.

8. The receive engine checks the OWN bit of the latest descriptor. If the host owns the descriptor (OWN bit is 0), the `EMAC_DMA0_STAT.RU` bit is set. The DMA receive engine enters the suspended state (Step 9). If the DMA owns the descriptor, the engine returns to Step 4 and awaits the next frame.
9. Before the receive engine enters the suspend state, partial frames are flushed from the receive FIFO (programs control flushing using the `EMAC_DMA0_OPMODE.DFF` bit).
10. The receive DMA exits the suspend state when a receive poll demand is given or the start of next frame is available from the receive FIFO of the MFL. The engine proceeds to Step 2 and refetches the next descriptor.

Receive Frame Processing

The EMAC transfers the received frames to the application memory only when:

- the frame passes the address filter subblock, and
- the frame size is greater than or equal to configurable threshold bytes set for the receive FIFO of MFL, or
- the complete frame is written to the FIFO in store-and-forward mode.

If the frame fails the address filtering, the EMAC block drops the frame (unless the `EMAC_MACFRMFILT.RA` bit is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the receive FIFO.

After receiving 64 bytes (configurable threshold), the MFL block requests that the DMA block begin transferring the frame data to the receive buffer pointed to by the current descriptor. The DMA sets first descriptor (RDES0 [9]) after the SCB becomes ready to receive the data (if DMA is not fetching transmit data from the application). The descriptors release when the `OWN` (RDES [31]) bit is reset to 0. The bit is reset either as the data buffer fills up or as the last segment of the frame is transferred to the receive buffer. If the frame is contained in a single descriptor, both the last descriptor (RDES [8]) and the first descriptor (RDES [9]) are set.

The DMA fetches the next descriptor, sets the last descriptor (RDES [8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then, the DMA sets the `EMAC_DMA0_STAT.RI` bit. The same process repeats unless the DMA encounters a descriptor the application owns. If this encounter occurs, the receive process sets the `EMAC_DMA0_STAT.RU` bit and then enters the suspend state. The position in the receive list is retained.

Receive Descriptor Acquisition

The receive engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted when any of the following conditions is satisfied:

- The `EMAC_DMA0_OPMODE.SR` bit is set immediately after being placed in the run state.
- The data buffer of current descriptor is full before the frame ends for the current transfer.
- The controller completes frame reception, but the current receive descriptor is not yet closed.
- The receive process suspends because of an application-owned buffer (RDES0 [31] = 0) and a new frame is received.
- A receive poll demand issues.

Receive Process Suspended

If a new receive frame arrives while the receive process is in the suspend state, the DMA refetches the current descriptor in the application memory. If the DMA now owns the descriptor, the receive process reenters the run state and starts frame reception. If the application still owns the descriptor, by default, the DMA discards the current frame at the top of the receive FIFO and increments the missed frame counter. If more than one frame is stored in the receive FIFO, the process repeats.

Avoid the discarding or flushing of the frame at the top of the receive FIFO by setting the `EMAC_DMA0_OPMODE.DFF` bit. In such conditions, the receive process sets the receive buffer unavailable status and returns to the suspend state.

OWN Bit (Ownership) Semaphore

Usage or ownership of the transmit or receive descriptor between the application and EMAC is mutually exclusive. While the EMAC accesses the descriptor, the application cannot modify it. Conversely, while the host updates the descriptor, the EMAC cannot use the content of the descriptor. This functionality is implemented through the OWN bit in the transmit or receive descriptor, acting as a semaphore to prevent multiple, simultaneous access to the descriptors.

The following example is based on a usage case of 4 WORDs enabled for descriptors. A chain structure configuration is assumed. (The `EMAC_DMA0_BUSMODE.ATDS` bit is not set). However, the explanation of the `OWN` bit semaphore remains consistent irrespective of any particular mode of operation.

1. Transmit OWN Bit:
    - TDES0 – TDES3 words implement the transmit descriptors. TDES0 [31] is defined as the OWN bit. When TDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer addresses, by updating TDES0 through TDES3.
    - To release ownership of the descriptor to the EMAC, the application sets the transmit `OWN` bit, TDES0 [31], to 1. TDES0 [31] = 1 indicates that the descriptor is ready for the EMAC to use. DMA reads the descriptors, then fetches the data for transmission from the buffer locations pointed to by the transmit descriptors (TDES2 and TDES3). When either the last data buffer is empty or the end-of-frame is reached, DMA clears the TDES0 [31] bit to 0. Now, the transmit descriptor releases to the application for updates.
2. Receive `OWN` Bit:
    - RDES0 – RDES3 words implement the receive descriptors. RDES0 [31] is defined as the `OWN` bit. When RDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer locations for writing the received data, by updating RDES0 through RDES3. To give ownership of the descriptor to the EMAC, the host sets the receive `OWN` bit, RDES0 [31], to 1.
    - RDES0 [31] = 1 indicates that the descriptor is ready for use by the EMAC. DMA reads the descriptors, then writes the received data to the buffers with locations pointed to by the receive descriptors (RDES2

and RDES3). When either the last data buffer is full or the end-of-frame is reached, DMA clears the RDES0 [31] bit to 0. Now, the receive descriptor releases to the application for updates

Application Data Buffer Alignment

The transmit and receive data buffers do not have any restrictions on the start address alignment. The start address for the buffers aligns to any of the 4 bytes. However, the DMA always initiates transfers with the address aligned to the bus width with dummy data for the byte lanes not required. This alignment typically happens during the transfer of the beginning or end of an Ethernet frame.

Example for Buffer Read

If the transmit buffer address is 0x0002 and 15 bytes must transfer, the DMA reads 5 full words (5 x 32-bit data) from address 0x0000. However, when transferring data to the EMAC transmit FIFO, the extra bytes (the first 2 bytes) are dropped or ignored. Similarly, the last 3 bytes of the last transfer are also ignored. The DMA always transfers a full 32-bit data to the transmit FIFO, unless it is the end-of-frame.

Example for Buffer Write

If the receive buffer address is 0x0002 and 15 bytes of a received frame must transfer, the DMA writes 5 full words (5 x 32-bit data) to address 0x0000. However, the first 2 bytes of first transfer and the last 3 bytes of the third transfer have dummy data.

Buffer Size Calculations

The DMA engines do not update the size fields in the transmit and receive descriptors alone. The DMA updates only the status fields (RDES0 and TDES0) of the descriptors. The driver must perform the size calculations. The transmit DMA transfers the exact number of bytes (indicated by buffer size field of TDES1) towards the EMAC CORE. If a descriptor is marked as first (FS bit of TDES1 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (LS bit of TDES1), the DMA marks the last transfer from that data buffer as the end-of frame to the EMAC.

The receive DMA transfers data to a buffer until the buffer is full or the end-of frame is received from the MFL. If a descriptor is not marked as last (LS bit of RDES0), then the buffers of the descriptor are full. The amount of valid data in a buffer is its buffer size field minus the data buffer pointer offset, when the FS bit of that descriptor is set. The offset is zero when the data buffer pointer aligns to the data bus width. If a descriptor is marked as last, then the buffer cannot be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must:

- Read the frame length (FL bits of RDES0 [29:16]), and
- Subtract the sum of the buffer sizes of the preceding buffers in the frame

The receive DMA always transfers the start of next frame with a new descriptor.

EMAC FIFO Layer (EMAC MFL)

The MAC FIFO layer provides FIFO memory to buffer and regulates the frames between the application system memory and the EMAC CORE. It also allows the transfer of data between the application clock domain and the

EMAC clock domains. The MFL layer has transfer controllers for each direction, called the transmit controller (TxFIFO) and the receive controller (RxFIFO). The datapath for both directions is 32-bit wide and each controller has a dedicated FIFO.

The EMAC0 transmit FIFO size is fixed to 2048 bytes. The EMAC0 receive FIFO size is fixed to 2048 bytes.

The EMAC1 transmit FIFO size is fixed to 1024 bytes. The EMAC1 receive FIFO size is fixed to 512 bytes.

FIFO Layer Transmit Path

The DMA engine controls all transactions for the transmit path with the application. Ethernet frames read from the system memory are pushed into the FIFO by the DMA. The frame is then popped out and transferred to the EMAC CORE when triggered. When the end-of-frame transfers, the status of the transmission is taken from the EMAC CORE and transferred back to the DMA. The FIFO fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory through the SCB interface.

When the DMA enables the `EMAC_DMA0_OPMODE.OSF` bit, the MFL receives the second frame into the FIFO while transmitting the first frame. When the first frame has transferred, the status is sent to DMA. If the DMA has already completed sending the second packet to the MFL, it waits for the status of the first packet before proceeding to the next frame.

The following are the modes of operation for FIFO transactions.

1. Threshold mode. When the number of bytes in the FIFO crosses the configured threshold level, the data is ready to be popped out and forwarded to the EMAC CORE. The data is also ready when the end-of-frame is written before the threshold is crossed. The DMA uses the `TTC` bits of the DMA bus mode register to configure the threshold level.
2. Store-and-Forward mode. In this mode, the MFL pops the frame towards the EMAC CORE after a complete frame is stored in the FIFO. If the TX FIFO size is smaller than the Ethernet frame for transmission (such as a jumbo frame), then the frame forwards in two cases. The TX FIFO is almost full or the requested FIFO does not have space to accommodate the requested burst-length. Therefore, the FIFO read controller never stalls in store-and-forward mode even if the Ethernet frame length is bigger than the TX FIFO depth.

   The FIFO threshold in the store-and-forward mode is given by the following formula.

   FIFO threshold = FIFO_SIZE [(PBL + 3) (DataWidth/8)) where

   DataWidth = 32 bits and PBL = Burst Length programmed through the DMA_BUSMODE register.

   NOTE: To avoid occurrences of a TX underflow event when using the store-and-forward mode, or in other words, to ensure that the entire frame is stored in the FIFO before the MFL pops the frame towards the EMAC CORE for transmission, ensure that the FIFO threshold (calculated with the above formula) is greater than the packet size. The PBL needs to be programmed accordingly.

Transmit FIFO and Half-Duplex Retransmissions

While a frame transfers from the FIFO, a collision event can occur on the EMAC line interface in half-duplex mode. The EMAC then indicates a retry attempt to the MFL. The EMAC gives the status before the end-of-frame transfers from MFL. Then, the MFL enables the retransmission by popping out the frame again from the FIFO.

After more than 96 bytes pop out of FIFO, the FIFO controller frees up that space. The controller makes it available to the DMA to push in more data. Retransmission is not possible after this threshold is crossed or when the EMAC CORE indicates a late-collision event.

Transmit FIFO Flush Operation

The EMAC provides control to the software to flush the transmit FIFO in the MFL layer using the `EMAC_DMA0_OPMODE.FTF` bit. The flush operation is immediate. The MFL clears the transmit FIFO and the corresponding pointers to the initial state. It clears the FIFO and pointers even if it is in the middle of transferring a frame to the EMAC CORE. The data that the MAC transmitter has already accepted is not flushed. The data is scheduled for transmission and results in underflow. The transmit FIFO does not complete the transfer of the rest of the frame. As in all underflow conditions, a runt frame is transmitted and observed on the line. The status of the frame is marked with both underflow and frame flush events (TDES0 bits 13 and 1).

The MFL also stops accepting any data from the application (DMA) during the flush operation. The MFL generates and transfers transmit status words to the application for the number of frames flushed inside the MFL (including partial frames). Frames that completely flush in the MFL have the status bit for frame flush (TDES0 bit 13) set. The MFL completes the flush operation when the application (DMA) accepts all of the status words for the frames that flushed. The MFL then clears the transmit FIFO flush control register bit. The MFL starts accepting new frames from the application (DMA).

FIFO Layer Receive Path

The receive controller operates in the following sequence:

1. When the EMAC CORE receives a frame, it pushes in data with the frame start and end indicators. The MFL accepts the data and pushes it into the FIFO.
2. The receive controller takes the data out of the FIFO and sends it to the DMA.
    - Threshold mode (default). This mode is configured using `EMAC_DMA0_OPMODE.RTC`. When the FIFO receives 64 bytes or a full packet of data, the receive controller pops out the data and indicates its availability to the DMA. Some error frames cannot be dropped, because the error status is received at the end-of-frame. By this time, the start of that frame has already been read out of the FIFO.
    - Rx FIFO Store-and-Forward mode. This mode is configured using `EMAC_DMA0_OPMODE.RSF`. A frame is read out only after being written completely into the receive FIFO. In this mode, all error frames are dropped such that only valid frames are read out and forwarded to the application. Error frames are dropped when the EMAC CORE is configured for this feature.
3. After the end-of-frame transfers, the status word from the EMAC CORE is also the pushed FIFO. When the status of a partial frame due to overflow is given out, the frame length field in the status word is not valid.

Receive FIFO Multi-Frame Handling

Since the status is available immediately following the data, the MFL stores any number of frames into the FIFO, as long as it is not full.

Receive FIFO Error Handling

If the MFL Rx FIFO is full before it receives the end of frame data from the EMAC, the DMA declares an overflow condition. The whole frame (including the status word) drops. The overflow counter in the DMA (overflow counter-register) increments. This activity occurs even if the `EMAC_DMA0_OPMODE.FEF` bit is set. If the start address of such a frame has already transferred, the rest of the frame drops. A dummy end of frame is written to the FIFO along with the status word. The status indicates a partial frame due to overflow. In such frames, the frame length field is invalid.

The MFL receive control logic can filter error and undersized frames using the `EMAC_DMA0_OPMODE.FEF` and `EMAC_DMA0_OPMODE.FUF` bits. If the start address of the frame has already transferred to the Rx FIFO read controller, that frame is not filtered. The start address of the frame transfers to the read controller after the frame crosses the receive threshold (set by the `EMAC_DMA0_OPMODE.RTC` bits).

If the MFL receive FIFO is configured for store-and-forward mode, it can filter and drop all error frames.

EMAC CORE

The EMAC CORE is the lowest block in the EMAC peripheral and it performs all operations with the external world (PHY chip). It has independent transmit and receive modules. The modules interact with the EMAC FIFO layer at one end and interacts with the PHY chip through the RMII interface at the other end. Both modules have several subblocks which are discussed in subsequent sections.

Transmission is initiated when the MFL (FIFO layer) pushes in data with the start of frame. The CORE then transmits to the reduced media-independent interface. After the end of frame transfers out, the CORE gives the status of the transmission back to the MFL. The MFL forwards the transmission to the application through DMA.

A receive operation initiates when the EMAC detects an SFD on the RMII/RGMII. The CORE strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The frame drops in the core when it fails the address filter.

NOTE: The term *CORE* (written in capitals) refers to the internal block of Ethernet peripheral. Do not confuse the term with the *processor core*.

Table 31-28: EMAC CORE Related Registers

Register Name	Description
MAC Configuration*1	Establishes receive and transmit operating modes including: • Watchdog, Jabber, and Jumbo frame sizes • Inter Frame Gap • Speed Control – 10/100/1000 Mbps • Full or Half Duplex • Loopback Mode • Checksum Offload • Enabling Tx or Rx Engines

Table 31-28: EMAC CORE Related Registers (Continued)

Register Name	Description
MAC Frame Filter	Contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as pass bad frames and pass control frames.
Hash Table High/Low[1]	A 64-bit hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame passes through the CRC logic. The upper 6 bits of the CRC register index the contents of the hash table.
SMI Address[1]	Controls the management cycles to the external PHY through the Station Management interface. The register also includes a field to program the frequency of MDC.
SMI Data[1]	Stores write data for the PHY register at the address specified in SMI Address register. This register also stores read data from the PHY register at the address specified by SMI address register.
Flow Control[1]	Controls the generation and reception of the control (pause command) frames by the flow control module of the EMAC. The fields of the control frame are selected as specified in the 802.3x specification. The EMAC uses the pause time value from this register in the pause time field of the control frame. The host must make sure that the activate bit is cleared before writing to the register.
VLAN Tag[1]	Contains the IEEE 802.1Q VLAN tag to identify the VLAN frames. The MAC compares the 13th and 14th bytes of the receiving frame (length or type) with 16.h8100. The following 2 bytes are compared with the VLAN tag. If a match occurs, it sets the received VLAN bit in the receive frame status. The legal length of the frame increases from 1518 bytes to 1522 bytes.
Debug	Provides the status of all main modules of the transmit and receive datapaths and the FIFOs. An all-zero status indicates that the MAC core is in idle state (and FIFOs are empty) and no activity exists in the datapaths.
Interrupt Status	The contents of this register identify the events in the EMAC-CORE that can generate MMC and PTP-related interrupts.
Interrupt Mask	Enables the program to mask the interrupt signal because of the corresponding PTP event in the interrupt status register.
MAC Address0 High/Low[1]	Holds the upper or lower 16 bits of the MAC address of the station. The first DA byte that is received on the RMII interface corresponds to the LS byte (bits [7:0]) of the MAC address low register. For example, if 0x112233445566 is received (0x11 is the first byte) on the RMII as the destination address, then the macaddress0 register [47:0] is compared with 0x665544332211.
Operation Mode[1]	

*1 There must not be any further writes to these registers until the first write updates. Otherwise, the second write operation is not updated properly. For correct operation, the delay between two writes to the same register location must be at least 8 cycles of 50MHz RMII REFCLK.

NOTE: Refer to the “Register Description” section for the detailed bit-level explanation of the registers.

EMAC CORE Transmission Engine

The following modules constitute the transmission function (transmission engine components) of the EMAC:

- Transmit Bus Interface Module (TBU)
- Transmit Frame Controller Module (TFC)
- Transmit Checksum Offload Engine (TCOE)
- Transmit Protocol Engine Module (TPE)
- Transmit Scheduler Module (STX)
- Transmit CRC Generator Module (CTX)
- Transmit Flow Control Module (FTX)

Transmit Bus Interface Module (TBU)

This module interfaces the transmit path of the EMAC CORE with the MAC Layer FIFO interface. This module outputs the transmit status to the application at the end of normal transmission or collision.

Transmit Frame Controller Module (TFC)

The transmit frame controller regulates frames as well as converts the 32-bit input data into an 8-bit stream.

When the number of bytes received from the application falls below 60 (DA+SA+LT+DATA), the state machine automatically appends zeros to the transmitting frame. The state machine makes the data length exactly 46 bytes to meet the requirement for minimum data field of IEEE 802.3. The EMAC module can also be programmed to not append any padding.

The frame controller receives the computed CRC and appends it as the FCS field to the data transmitting out. When the EMAC is programmed to not append the CRC value to the end of Ethernet frames, the TFC module ignores the computed CRC. However, when the EMAC is programmed to append pads for frames (DA+SA+LT +DATA) less than 60 bytes, then the CRC is always appended at the end of padded frame.

Transmit Checksum Offload Engine (TCOE)

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. The most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams. Therefore, the EMAC has a checksum offload engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path.

NOTE: The checksum for TCP, UDP, or ICMP is calculated over a complete frame, and then inserted into its corresponding header field. Because of this requirement, this function is enabled only when the transmit FIFO configuration is for store-and-forward mode. (The `EMAC_DMA0_OPMODE.TSF` bit is set.) If the MAC configuration is for threshold (cut-through) mode, the transmit COE is bypassed.

NOTE: Programs must make sure that the transmit FIFO is deep enough to store a complete frame before that frame transfers to the EMAC CORE transmitter. The program must enable the checksum insertion only in the frames that are less than the following number of bytes in size (even in the store-and-forward mode):

FIFO depth – PBL – 3 FIFO locations, where PBL is the programmed burst-length in the DMA bus mode register.

IP Header Checksum

In IPv4 datagrams, the 16-bit header checksum field indicates the integrity of the header fields (bytes 11 and 12 of the IPv4 datagram). The COE detects an IPv4 datagram when the Ethernet type field of the frame has the value 0x0800 and the version field of the IP datagram has the value 0x4. The checksum field of the input frame is ignored during calculation and replaced with the calculated value.

The IP header error status bit in transmit descriptor word TDES0 indicates the result of this IP header checksum calculation. The status bit is set whenever the values of the Ethernet type field and the IP header version field are not consistent. Or, the status bit is set when the Ethernet frame does not have enough data, as indicated by the IP header length field. In other words, this bit is set when an IP header error is asserted under the following circumstances.

For IPv4 datagrams:

- The received Ethernet type is 0x0800, but the version field of the IP header is not equal to 0x4.
- The IPv4 header length field indicates a value less than 0x5 (20 bytes).
- The total frame length is less than the value given in the IPv4 header length field.

For IPv6 datagrams:

- The Ethernet type is 0x86dd but the IP header version field is not equal to 0x6.
- The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding header length field in an extension header) is received.

If the COE detects an IP header error, it still inserts an IPv4 header checksum if the Ethernet type field indicates an IPv4 payload.

NOTE: IPv6 headers do not have a checksum field. Therefore, the COE does not modify the IPv6 header fields.

TCP/UDP/ICMP Checksum

The TCP/UDP/ICMP checksum engine processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP, or ICMP.

NOTE: See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460, and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6, and ICMPv6 packet header specifications, respectively.

NOTE: For non-TCP/UDP/ICMP/ICMPv6 payloads, this checksum engine is bypassed and nothing further is modified in the frame.

NOTE: For ICMP-over-IPv4 packets, the checksum field in the ICMP packet must always be 0x0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 0x0000, an incorrect checksum can be inserted into the packet.

NOTE: This engine does not process fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an encapsulated security payload), and IPv6 frames with routing headers. The checksum engine bypasses the checksum insertion for such frames even if the checksum insertion is enabled.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. This engine can work in the following two ways.

- The TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the checksum field of the input frame. This engine includes the checksum field in the checksum calculation, and then replaces the checksum field with the final calculated checksum.
- The engine ignores the checksum field, includes the TCP, UDP, or ICMPv6 pseudo-header data into the checksum calculation, and overwrites the checksum field with the final calculated value.

The status bit for the payload checksum error in the transmit descriptor word TDES0 indicates the result of this operation. The checksum engine sets the status bit for the payload checksum error when:

- The checksum engine detects that the frame has been forwarded to the MAC transmitter engine in the store-and-forward mode, *and*
- The end of frame (EOF) has not been written to the FIFO, *or*
- The packet ends before the number of bytes indicated by the payload length field in the IP header is received.

When the packet is longer than the indicated payload length, the COE ignores them as stuff bytes, and no error is reported. When the engine detects the first type of error, it does not modify the TCP, UDP, or ICMP header. For the second error type, it still inserts the calculated checksum into the corresponding header field.

Transmit checksum offloading is enabled by setting the CIC bits [23:22] of TDES0 word in the transmit descriptor.

Transmit Protocol Engine Module (TPE)

The transmit protocol engine consists of a state-machine that controls the protocol-level operation of Ethernet frame transmission. The module performs the following functions to meet the IEEE 802.3 specifications.

- Generates preamble and SFD
- Generates carrier extension in half-duplex mode (only in RGMII mode)
- Supports frame bursting in half-duplex mode (only in RGMII mode)
- Generates jam pattern in half-duplex mode
- Contains time stamp snapshot logic for IEEE 1588 support
- Jabber timeout
- Flow control for half-duplex mode (back pressure)
- Generates transmit frame status

When a new frame transmission is requested, the protocol engine sends out the preamble and SFD, followed by the data received. The preamble is defined as 7 bytes of 10101010 pattern. The SFD is defined as 1 byte of 10101011 pattern.

The collision window is defined as 1 slot time (512-bit times for 10/100 Mbps and 4096 bit times for 1000 Mbps). The jam pattern generation is applicable only to half-duplex mode, not to full-duplex mode. A collision can occur any time from the beginning of the frame to the end of the CRC field. When a collision happens, the state machine sends a 32-bit jam pattern of 0x55555555 on the RMII/RGMII. The pattern informs all other stations that a collision has occurred. If the collision happens during the preamble transmission phase, it completes the transmission of preamble and SFD and then sends the jam pattern. If the collision occurs after the collision window and before the end of the FCS field, it sends a 32-bit jam pattern. It also sets the late collision bit in the transmit frame status.

In RGMII half-duplex mode (1,000 Mbps), the transmit state machine ensures that all valid carrier events exceed a slot time of 4,096 bit times. To accomplish this, any transmit frame shorter than 512 bytes from the TFC module is extended using a carrier extension. This is signaled to the PHY using RGMII_TXCTL pin and sending 0x00 through RGMII_TXD.

The module maintains a jabber timer to cut off the transmission of Ethernet frames when the TFC module transfers more than 2,048 (default) bytes. The timeout changes to 10,240 bytes when the jumbo frame is enabled.

The transmit state machine uses the deferral mechanism for the flow control (back pressure) in half-duplex mode. When the application asks to stop receiving frames, the module sends a jam pattern of 32 bytes. It sends the pattern whenever it senses a reception of a frame. Transmit flow control must be enabled. This activity results in a collision and the remote station backs off.

The application can request a flow control signal by setting the `EMAC_FLOWCTL.FCBBPA` bit. If the application requests a frame transmission, then it is scheduled and transmitted even when the back pressure is activated. If the back pressure is activated for a long time, then the remote stations abort their transmissions due to excessive collisions. (For example, a long time is when more than 16 consecutive collision events occur.)

If PTP time stamping is enabled for the transmit frame, this block takes a snapshot of the PTP system time when the SFD is put onto the transmit bus.

Transmit Scheduler Module (STX)

The transmit scheduler is responsible for scheduling the frame transmission on the RMII/RGMII . The two major functions of this module are:

- Maintain the inter-frame gap between two transmitted frames.
- Follow the truncated binary exponential back-off algorithm for half-duplex mode.

The scheduler maintains an idle period of the configured inter-frame gap (`EMAC_MACCFG.IFG` bits) between any two transmitted frames. The scheduler starts its IFG counter when the carrier signal of the reduced media-independent interface goes inactive. In half-duplex mode and when IFG is configured for 96-bit times, the scheduler follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The module resets its IFG counter when a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the scheduler continues the IFG count and enables the transmitter after the IFG interval.

Transmit CRC Generator Module (CTX)

The transmit CRC generator module generates the CRC for the FCS field of the Ethernet frame (DA + SA + LT + DATA + PAD).

This module calculates the 32-bit CRC for the FCS field of the Ethernet frame. The following polynomial defines the encoding:

$G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.$

Transmit Flow Control Module (FTX)

The transmit flow control module generates pause frames and transmits them to the frame controller as necessary, in full-duplex mode. The application can request the flow control module to send a pause frame by setting the `EMAC_FLOWCTL.FCBBPA` bit.

If the application has requested flow control, the flow control module generates and transmits a single pause frame. The value of the pause time in the generated frame contains the programmed pause time value configured using the `EMAC_FLOWCTL.PT` bit. The module can extend the pause or end the pause prior to the time specified in the previously transmitted pause frame. To change the pause, the application must request another pause frame transmission after programming the `EMAC_FLOWCTL.PT` bit with an appropriate value.

If the flow control signal goes inactive prior to the sampling time, the flow control module transmits a pause frame with zero pause time. This event indicates to the remote end that the receive buffer is ready to receive new data frames.

Source Address, VLAN, and CRC Insertion, Replacement, or Deletion

The MAC supports the following functions for transmit frames:

- Source address insertion or replacement
- VLAN insertion, replacement, or deletion
- CRC replacement

Source Address Insertion or Replacement

The software can use the SA insertion or replacement feature to instruct the MAC to do the following for transmit frames:

- Insert the content of the MAC address registers in the SA field.
- Replace the content of the SA field with the content of the MAC address registers.

The software can enable the SA insertion or replacement feature for all transmit frames or selective frames.

- To enable SA insertion or replacement feature for all frames, program bits[30:28] of the MAC Configuration register.

- To enable the SA insertion or replacement feature for selective frames, program the SA insertion control field (TDES1 Bits [31:29]) in the first transmit descriptor of the frame. When bit 31 of TDES1 is set, the SA insertion control field indicates insertion or replacement by MAC address1 registers. When bit 31 of TDES1 is reset, it indicates insertion or replacement by MAC address 0 registers. If MAC address1 registers are not enabled, then EMAC uses MAC address0 registers for insertion or replacement. The choice is not based on the value of the most significant bit of the SA insertion control field.

When SA insertion is enabled, the application ensures that the frames sent to the MAC do not have the SA field. This functionality is because the MAC does not check the presence of SA field in the transmit frame. MAC inserts the content of the MAC address registers in the SA field. Similarly, when SA replacement is enabled, the application ensures that frames sent to the MAC have the SA field. The MAC replaces the 6 bytes, following the destination address field in the transmit frame, with the content of the MAC address registers.

VLAN Insertion, Replacement, or Deletion

The software can use the VLAN insertion, replacement, or deletion feature to instruct the MAC to do the following for transmit frames:

- Insert or replace the VLAN type field (C-VLAN or S-VLAN indicated by bit 19 (CSVL) of VLAN tag inclusion or replacement register and VLAN tag field in the transmit frame with bit [15:0], VLT, of VLAN tag inclusion or replacement register.
- Delete the VLAN type and VLAN tag fields in transmit frame.

The software can enable the VLAN insertion, replacement, or deletion feature for all transmit frames or selective frames. To enable this function for all transmit frames, program Bits[17:16] of the VLAN Tag Inclusion or Replacement register. To enable this function for selective, program the VLAN insertion control field (TDES0 Bits [19:18]) in the first transmit descriptor of the frame. When VLAN replacement or deletion is enabled, the MAC checks the presence of the VLAN type field (0x8100 or 0x88a8), after the Destination Address (DA) and SA fields, in the transmit frame. The replace or delete operation does not occur when the VLAN type field is not detected in the 2 bytes following the DA and SA fields. However, when VLAN insertion is enabled, the MAC does not check the presence of VLAN type field in the transmit frame. MAC inserts the VLAN type and VLAN tag fields.

CRC Replacement

The software can use the CRC replacement feature to instruct the MAC to replace the FCS field in the transmit frame with the CRC computed by the MAC. This feature works on per-frame basis. To enable the CRC replacement feature, program the CRC replacement control field (bit 24 of transmit descriptor word 0 (TDES0)) in the first transmit descriptor of the frame.

NOTE: This feature is valid only when disable CRC control (bit 27 in TDES0) is enabled. The software provides the FCS field in the transmit frame. If SA or VLAN insertion control is enabled, the MAC appends or replaces the FCS field with the computed CRC when Disable CRC Control is enabled or disabled, respectively.

The ***CRC Replacement*** table shows how CRC replacement is performed based on the values of bit 27 (DC) and bit 24 (CRCR) of transmit descriptor word 0 (TDES0).

Table 31-29: CRC Replacement

DC	CRCR	Description
0	x	Append CRC. When DC = 0, the MAC appends the computed CRC irrespective of the CRCR setting.
1		Replace CRC
1	0	No operation (User has appended the CRC)

Source Address Filtering

The ***Source Address Filtering*** table provides filtering possibilities for the source address using the EMAC AFM module. The MAC receive frame filter register (`EMAC_MACFRMFILT`) contains these bits.

Table 31-30: Source Address Filtering

	SA Filter Operation			
Frame Type	PR	SAIF	SAF	Result
Unicast	1	X	X	Pass all frames
	0	0	0	Pass on perfect or group filter match but do not drop failing frames
	0	1	0	Fail on perfect or group filter match but do not drop frame
	0	0	1	Pass on perfect or group filter match and drop failing frames
	0	1	1	Fail on perfect or group filter match and drop failing frames

EMAC CORE Reception Engine

The following are the functional blocks (reception engine components) in the receive path of the EMAC core.

- Receive Protocol Engine Module (RPE)
- Receive CRC Module (CRX)
- Receive Frame Controller Module (RFC)
- Receive Flow Control Module (FRX)
- Receive Checksum Offload Engine (RCOE)
- Receive Bus Interface Unit Module (RBU)
- Address Filtering Module (AFM)

Receive Protocol Engine Module (RPE)

The receive protocol engine is a state-machine that strips the preamble, SFD, and carrier extension (in 1000 Mbps half-duplex mode) of received frame. Once the receive data valid signal (`ETH0_CRS/ETH1_CRS` signal of RMII or

RXCTL signal of RGMII) becomes active, the protocol engine begins hunting for the SFD field from the receive modifier logic. Until then, the state machine drops the receiving preambles. Once the SFD is detected, it begins sending the data of the Ethernet frame to the frame controller, beginning with the first byte following the SFD (destination address).

NOTE: According to the IEEE 802.3 Ethernet specifications, the EMAC receiver does not need to look or check for the preamble pattern. It has to wait only for the SFD pattern to identify the start of a frame. Then the EMAC receiver accepts a frame even when no preamble is received before the SFD pattern.

If PTP time stamping is enabled, the RPE takes a snapshot of the PTP system time when detecting any SFD of the frame on the reduced media-independent interface. Unless the MAC filters out and drops the frame, this time stamp passes on to the application

The protocol engine also decodes the length or type field of the receiving Ethernet frame. The state machine sends the data of the frame up to the count specified in the length or type field if these conditions are met:

- The length or type field is less than 0x600
- The MAC is programmed for the auto CRC or PAD stripping option

It then starts dropping bytes (including the FCS field).

If the length or type field is greater than or equal to 0x600, the protocol engine sends all received Ethernet frame data to the frame controller. The transfer does not depend on the value of the programmed auto-CRC strip option.

The EMAC is programmed with the watchdog timer enabled (default setting). In this configuration, frames above 2,048 (10,240 if jumbo frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS) are cut off at the protocol engine. Set the `EMAC_MACCFG.WD` bit to disable this feature. However, even when the watchdog timer is disabled, frames greater than 16 KB are cut off and a watchdog timeout status is issued.

The EMAC supports loopback of transmitted frames onto its receiver. By default, the EMAC loopback function is disabled. Set the `EMAC_MACCFG.LM` bit to enable the function.

At the end of every received frame, the protocol engine generates received frame status and sends it to the frame controller. Control, missed frame, and filter fail status are added to the receive status in the frame controller.

Receive CRC Module (CRX)

The receive CRC module checks for any CRC errors in the receiving frame.

This module calculates the 32-bit CRC for the received frame that includes the destination address field through the FCS field (DA+SA+LT+DATA+PAD+FCS). The following generating polynomial defines the encoding.

G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

Irrespective of the auto pad or CRC strip, the CRC module receives the entire frame to compute the CRC check for received frame.

Receive Frame Controller Module (RFC)

The main functions of the frame controller are:

- Converting the 8-bit stream data to 32-bit data
- Frame filtering
- Attaching the calculated IP checksum
- Updating the receive status

If the `EMAC_MACFRMFILT.RA` bit is set, the RFC module initiates the data transfer as soon as possible. At the end of the data transfer, the frame controller sends out the received frame status that includes the address filtering pass or fail status.

If the `EMAC_MACFRMFILT.RA` bit is reset, the frame controller performs frame filtering based on the destination or source address. (The application still must perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, for example.) After receiving the destination or source address bytes, the frame controller checks the filter-fail signal from the AFM module for an address match. On detecting a filter-fail from AFM, the frame is dropped and not transferred to the application.

Receive Flow Control Module (FRX)

The receive flow controller detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame. The flow controller is enabled only in full-duplex mode. The EMAC uses the `EMAC_FLOWCTL.RFE` bit to enable or disable the function for pause frame detection.

Once the receive flow control is enabled, the flow controller begins monitoring the received frame destination address for any match with the multicast address of the control frame (0x0180C2000001). If a match is detected, it indicates to the frame controller, that the destination address of the received frame matches the reserved control frame destination address. The RFC module then decides whether to transfer the received control frame to the application, based on the `EMAC_MACFRMFILT.PCF` bit setting.

The receive flow controller also decodes the type, opcode, and pause timer field of the receiving control frame. The flow controller requests the MAC transmitter pause the transmission of any data frame.

- If the byte count of the frame status indicates 64 bytes, ***and***
- If there is no CRC error

The transmission is paused for the decoded pause time value, multiplied by the slot time (64-byte times). Meanwhile, if another pause frame is detected with a zero pause time value, the module resets the pause time and gives another pause request to the transmitter. The module does not generate a pause request to the transmitter:

- If the received control frame does not match the type field (0x8808), opcode (0x00001), or byte length (64 bytes), ***or***
- If there is a CRC error

For a pause frame with a multicast destination address, the frame controller filters the frame based on the address match from the flow controller. For a pause frame with a unicast destination address, the filtering in the FRX module depends on:

- If the destination address matched the contents of the MAC address register 0 (`EMAC_ADDR0_HI` or `EMAC_ADDR0_LO`), ***and***
- If the `EMAC_FLOWCTL.UP` bit is set

The module detects a pause frame even with a unicast destination address. The EMAC uses the `EMAC_MACFRMFILT.PCF` bits to control the filtering for control frames in addition to the address filter module.

Receive Checksum Offload Engine (RCOE)

When checksum offloading is enabled, both IPv4 and IPv6 frames in the received Ethernet frames are detected and processed for data integrity. Programs can enable this module by setting the `EMAC_MACCFG.IPC` bit. The EMAC receiver identifies IPv4 or IPv6 frames by checking for value 0x0800 or 0x86DD, respectively, in the received Ethernet type field of frames. This identification applies to VLAN-tagged frames as well. *Extended descriptor mode (8 x32-bit words) must be enabled to get the IPC checksum engine status in RDES4.* To check status, poll bit 0 of RDES0 word of receive descriptor. Then, if this bit is set, parse bits [7:0] of RDES4 word.

The receive checksum offload engine calculates IPv4 header checksums and checks if they match the received IPv4 header checksums. The IP header error bit is set for any mismatch between the indicated payload type (Ethernet type field) and the IP header version. The IP header error bit is also set when the received frame does not have enough bytes, as indicated by the length field of the IPv4 header. (The bit is set when fewer than 20 bytes are available in an IPv4 or IPv6 header).

This engine also identifies a TCP, UDP, or ICMP payload in the received IP datagrams (IPv4 or IPv6). The engine calculates the checksum of such payloads properly, as defined in the TCP, UDP, or ICMP specifications. This engine includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation appears as a payload checksum error bit in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not tally to the expected payload length given in the IP header.

NOTE: The COE engine bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP, or ICMP. This information is given in the receive status(whether the checksum engine is bypassed or not).

The ***Checksum Error Status*** table shows bit combination in receive descriptors (frame status with full checksum offload engine enabled and advanced timestamps not enabled).

Table 31-31: Checksum Error Status

IEEE802.3 Frame: bit 5 of RDES0	Header Checksum Error (HCE): bit 3 of RDES4	Payload Checksum Error (PCE): bit 4 of RDES4	Frame Status
0	0	0	The frame is an IEEE 802.3 frame (length field value is less than 0x0600).
1	0	0	IPv4/IPv6 type frame in which no checksum error is detected.

Table 31-31: Checksum Error Status (Continued)

IEEE802.3 Frame: bit 5 of RDES0	Header Checksum Error (HCE): bit 3 of RDES4	Payload Checksum Error (PCE): bit 4 of RDES4	Frame Status
1	0	1	IPv4/IPv6 type frame in which a payload checksum error (as described for PCE) is detected
1	1	0	IPv4/IPv6 type frame in which IP header checksum error (as described for IPC HCE) is detected.
1	1	1	IPv4/IPv6 type frame in which both PCE and IPC HCE is detected.
0	0	1	IPv4/IPv6 type frame in which there is no IP HCE and the payload check is bypassed due to unsupported payload.
0	1	1	Type frame which is neither IPv4 or IPv6 (COE bypasses the checksum check completely)
0	1	0	Reserved

Receive Bus Interface Unit Module (RBU)

The receive bus interface unit (RBU) constructs the 32-bit data received from the frame controller into a 32-bit FIFO-based protocol.

Address Filtering Module (AFM)

The address filtering (AFM) module performs the destination checking function on all received frames and reports the address filtering status to the frame controller. The address checking is based on different parameters (frame filter register, `EMAC_MACFRMFILT`) chosen by the application. These parameters are inputs to the AFM module as control signals. The AFM module reports the status of the address filtering based on the combination of these inputs. The AFM module also reports whether the receiving frame is a multicast frame or a broadcast frame, as well as the address filter status. The AFM module uses the physical (MAC) address of the station and the multicast hash table for address checking.

- *Hash or Perfect Address Filter.* The destination address filter can be configured to pass a frame when its destination address matches either the hash filter or the perfect filter. Set the `EMAC_MACFRMFILT.HPF` bit, the corresponding `EMAC_MACFRMFILT.HUC`, or `EMAC_MACFRMFILT.HMC` bits. This configuration applies to both unicast and multicast frames. If the `EMAC_MACFRMFILT.HPF` bit is reset, only one of the filters (hash or perfect) is applied to the received frame.

  NOTE: Hash filtering is not perfect filtering because a 48-bit MAC address is reduced to a 6-bit hash value. So, there can be instances where more than one address has the same hash value.

- *Unicast Destination Address Filter.*
  - The AFM supports one MAC address two MAC addresses for unicast perfect filtering. If perfect filtering is selected, the AFM compares all 48 bits of the received unicast address with the programmed MAC address for any match. (The `EMAC_MACFRMFILT.HUC` bit is reset for perfect filtering).

  - In hash filtering mode, the AFM performs imperfect filtering for unicast addresses using a 64-bit hash table. (The `EMAC_MACFRMFILT.HUC` bit is set in hash filtering mode.) For hash filtering, the AFM uses the upper 6-bit CRC of the received destination address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register, and a value of 111111 selects bit 63 of the hash table register. If the corresponding bit (indicated by the 6-bit CRC) is set to 1, the unicast frame has passed the hash filter. Otherwise, the frame has failed the hash filter.
- *Multicast Destination Address Filter.*
  - Program the EMAC to pass all multicast frames by setting the `EMAC_MACFRMFILT.PM` bit. If the `EMAC_MACFRMFILT.PM` bit is reset, the AFM filters multicast addresses based on the `EMAC_MACFRMFILT.HMC` bit. In perfect filtering mode, the multicast address is compared with the programmed MAC destination address register. The EMAC also supports group address filtering.
  - In hash filtering mode, the AFM performs imperfect filtering using a 64-bit hash table. For hash filtering, the AFM uses the upper 6-bit CRC of the received multicast address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register and a value of 111111 selects bit 63 of the hash table register. If the corresponding bit is set to 1, then the multicast frame has passed the hash filter. Otherwise, the frame has failed the hash filter.
- *Broadcast Address Filter.* The AFM does not filter any broadcast frames in the default mode. However, if the EMAC is programmed to reject all broadcast frames by setting the `EMAC_MACFRMFILT.DBF` bit, the AFM asserts the filter fail signal, whenever a broadcast frame is received.
- *Unicast Source Addrress Filter.* EMAC can also perform a perfect filtering, based on the source address field of the received frames. By default, the AFM compares the SA field with the values programmed in the SA registers. The MAC address register 1 can be configured to contain SA instead of DA for comparison, by setting bit 30 of the corresponding register. Group filtering with SA is also supported. User can filter a group of addresses by masking one or more bytes of the address. The frames that fail the SA filter are dropped by the MAC if the `EMAC_MACFRMFILT.SAF` bit is set. When the bit is set, the result of SA filter and DA filter is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter fail result drops the frame and both filters have to pass in order to forward the frame to the application.
- *Inverse Filtering Operation.* There is an option to invert the filter-match result at the final output. The EMAC uses the `EMAC_MACFRMFILT.DAIF` bit to control this operation. The function of this bit applies to both unicast and multicast DA frames. The result of the unicast or multicast destination address filter is inverted in this mode.
- *Inverse Filtering Operation.* For both destination and source address filtering, there is an option to invert the filter-match result at the final output. This is controlled by the `EMAC_MACFRMFILT.DAIF` or `EMAC_MACFRMFILT.SAIF` bits. The `EMAC_MACFRMFILT.DAIF` bit is applicable for both unicast and multicast DA frames. The result of the unicast /multicast destination address filter is inverted in this mode. Similarly, when the `EMAC_MACFRMFILT.SAIF` bit is set, the result of unicast source address filter is reversed.

- ***Inverse Filtering Operation.*** There is an option to invert the filter-match result at the final output. The EMAC uses the `EMAC_MACFRMFILT.DAIF` bit to control this operation. The function of this bit applies to both unicast and multicast DA frames. The result of the unicast or multicast destination address filter is inverted in this mode.

Destination Address Filtering

The ***Destination Address Filtering*** table provides filtering possibilities for the destination address using the EMAC AFM module. The MAC receive frame filter register (`EMAC_MACFRMFILT`) contains these bits.

Table 31-32: Destination Address Filtering

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Do-not-care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames
	0	X	0	X	0	X	X	Pass on perfect or group filter match
	0	X	0	X	1	X	X	Fail on perfect or group filter match
	0	0	1	X	0	X	X	Pass on hash filter match
	0	0	1	X	1	X	X	Fail on hash filter match
	0	1	1	X	0	X	X	Pass on hash or perfect or group filter match
	0	1	1	X	1	X	X	Fail on hash or perfect or group filter match
Multicast	1	X	X	X	X	X	X	Pass all frames
	X	X	X	X	X	1	X	Pass all frames
	0	X	X	0	0	0	X	Pass on perfect or group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	0	0	X	Pass on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	0	0	X	Pass on hash or perfect or group filter match and drop PAUSE control frames if PCF = 0x

Table 31-32: Destination Address Filtering (Continued)

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Do-not-care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
	0	X	X	0	1	0	X	Fail on perfect or group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	1	0	X	Fail on hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	1	0	X	Fail on hash or perfect or group filter match and drop PAUSE control frames if PCF = 0x

Source Address Filtering

The *Source Address Filtering* table provides filtering possibilities for the source address using the EMAC AFM module. The MAC receive frame filter register (`EMAC_MACFRMFILT`) contains these bits.

Table 31-33: Source Address Filtering

Frame Type	SA Filter Operation			Result
	PR	SAIF	SAF	
Unicast	1	X	X	Pass all frames
	0	0	0	Pass on perfect or group filter match but do not drop failing frames
	0	1	0	Fail on perfect or group filter match but do not drop frame
	0	0	1	Pass on perfect or group filter match and drop failing frames
	0	1	1	Fail on perfect or group filter match and drop failing frames

VLAN Tag Based Filtering

EMAC0 supports VLAN tag perfect filtering and VLAN tag hash filtering.

VLAN Tag Perfect Filtering

In VLAN tag perfect filtering, the MAC compares the VLAN tag of the received frame and provides the VLAN frame status to the application. Based on the programmed mode, the MAC compares the lower 12 bits or all 16 bits of the received VLAN tag to determine the perfect match. If VLAN tag perfect filtering is enabled, the MAC forwards the VLAN-tagged frames along with VLAN tag match status. It drops the VLAN frames that do not match. Inverse matching for VLAN frames can be enabled using the `EMAC_VLANTAG.VTIM` bit. In addition, matching of S-VLAN tagged frames along with the default C-VLAN tagged frames can be enabled using `EMAC_VLANTAG.ESVL` bit. The VLAN frame status bit (Bit 10 of RDES0) indicates the VLAN tag match status for the matched frames.

NOTE: The source or destination address (if enabled) has precedence over the VLAN tag filters. A frame which fails the source or destination address filter is dropped irrespective of the VLAN tag filter results. By default, the VLAN tag-based perfect filter is available in all configurations.

VLAN Tag Hash Filtering

The MAC provides VLAN tag hash filtering with a 16-bit hash table. The MAC performs the VLAN hash matching based on the `EMAC_VLANTAG.VTHM` bit setting. If the `EMAC_VLANTAG.VTHM` bit is set, the most significant 4 bits of VLAN tag's CRC-32 are used to index the content of the VLAN hash table register (`EMAC_VLAN_HSHTBL`). A value of 1 in the `EMAC_VLAN_HSHTBL` register, corresponding to the index, indicates that the VLAN tag of the frame matched and the packet is forwarded. A value of 0 indicates that VLAN-tagged frame is dropped.

The MAC also supports the inverse matching of the VLAN frames. In the inverse matching mode, when the VLAN tag of a frame matches the perfect or hash filter, the packet is dropped. If the VLAN perfect and VLAN hash match are enabled, a frame matches if either the VLAN hash or the VLAN perfect filter matches. When inverse match is set, a packet is forwarded only when both perfect and hash filters indicate mismatch. The ***VLAN Matching and Final VLAN Match Status*** table shows the possibilities for VLAN matching and the final VLAN match status. When the `EMAC_MACFRMFILT.RA` bit is set, all frames are received and the VLAN match status is indicated in bit 10 of receive descriptor word 0 (RDES0). When the `EMAC_MACFRMFILT.RA` bit is not set and the `EMAC_MACFRMFILT.VTFE` bit register is set, the frame is dropped when the final VLAN match status is fail.

When VLAN VID is programmed to 0 in the `EMAC_VLANTAG.VL` bit field, all VLAN-tagged frames are perfect matches. But the status of the VLAN hash match depends on the VLAN hash enable bit and VLAN inverse filter bit. The ***VLAN Matching and Final VLAN Match Status*** table shows the possibilities for VLAN matching and the final VLAN match status.

Table 31-34: VLAN Matching and Final VLAN Match Status

VID	VLAN Perfect Filter Match Status (VPF)	VLAN Hash Enable Bit	VLAN Hash filter Match Status (VTHMS)	VLAN Inverse Filter Bit (VTIM)	Final VLAN Match Status
VID=0	Pass	0	X	X	Pass
	Pass	1	X	0	Pass
	Pass	1	Fail	1	Pass
	Pass	1	Pass	1	Fail
VID!=0	Pass	X	X	0	Pass
	Fail	0	X	0	Fail
	Fail	1	Fail	0	Fail
	Fail	1	Pass	0	Pass
	Fail	0	X	1	Pass
	Pass	X	X	1	Fail

Table 31-34: VLAN Matching and Final VLAN Match Status (Continued)

VID	VLAN Perfect Filter Match Status (VPF)	VLAN Hash Enable Bit	VLAN Hash filter Match Status (VTHMS)	VLAN Inverse Filter Bit (VTIM)	Final VLAN Match Status
	Fail	1	Pass	1	Fail
	Fail	1	Fail	1	Pass

Layer 3 and Layer 4 Frame Filtering

The MAC supports Layer 3 and Layer 4 based frame filtering. The Layer 3 filtering refers to the IP source or destination address filtering in the IPv4 or IPv6 frames whereas Layer 4 filtering refers to the source or destination port number filtering in TCP or UDP. The Layer 3 and Layer 4 frame filtering feature automatically enables the IPC full checksum offload engine on the receive side.

When Layer 3 and Layer 4 filtering is enabled, the frames are filtered in the following way:

Matched Frames. The MAC forwards the frames, which match all enabled fields, to the application along with the status. The MAC gives the matched field status only if the `EMAC_MACCFG.IPC` bit is set. The MAC forwards the frames, which match all enabled fields, to the application along with the status.

> **NOTE:** The source or destination address and VLAN tag filters (if enabled) have precedence over Layer 3 and Layer 4 filtering. This means that a frame which fails the source or destination address or VLAN tag filter is dropped irrespective of the Layer 3 and Layer 4 filter results.

Unmatched Frames. The MAC drops the frames that do not match any of the enabled fields. You can use the inverse match feature to block or drop a frame with specific TCP or UDP over IP fields and forward all other frames. If the Rx FIFO operates in the threshold mode and threshold is programmed to a small value, such that frame transfer to application starts before the failed Layer 3 and Layer 4 filter results are available, a partial frame with appropriate abort status may be received by the application.

Non-TCP or UDP IP Frames. By default, all non-TCP or UDP IP frames are bypassed from the Layer 3 and Layer 4 filters. You can optionally program the MAC to drop all non-TCP or UDP over IP frames.

Layer 3 and Layer 4 Filters Register Set

The MAC implements a set of registers for Layer 3 and Layer 4 based frame filtering. In a register set, there is a control register, such as the `EMAC_L3L4_CTL` register (Layer 3 and Layer 4 control register), to control the frame filtering. In addition, there are five address registers to program the Layer 3 and Layer 4 fields to be matched, which are:

- `EMAC_L4_ADDR` (Layer 4 Address Register)
- `EMAC_L3_ADDR0` (Layer 3 Address 0 Register)
- `EMAC_L3_ADDR1` (Layer 3 Address 1 Register)
- `EMAC_L3_ADDR2` (Layer 3 Address 2 Register)

- `EMAC_L3_ADDR3` (Layer 3 Address 3 Register)

Layer 3 Filtering

The MAC supports perfect matching or inverse matching for IP source and destination address. In addition, a program can match the complete IP address or mask the lower bits matching, that is, compare all bits of the address except the specified lower mask bits.

For IPv6 frames filtering, programs can enable the last four data registers of a register set to contain the 128-bit IP source or IP destination address. Program the IP source or destination address in the order defined in the IPv6 specification, that is, the first byte of the IP source or destination address in the received frame is in the higher byte of the register and the subsequent registers follow the same order.

For IPv4 frames filtering, a program can enable the second and third data registers of a register set to contain the 32-bit IP source address and IP destination address. The remaining two data registers are reserved. Program the IP source and destination address in the order defined in the IPv4 specification, that is, the first byte of IP source and destination address in the received frame in the higher byte of the respective register.

Layer 4 Filtering

The MAC supports perfect matching or inverse matching for TCP or UDP source and destination port numbers. Only one type (TCP or UDP) is programmable at a time. The first data register contains the 16-bit source and destination port numbers of TCP or UDP, that is, the lower 16 bits for source port number and higher 16 bits for destination port number. Program the TCP or UDP source and destination port numbers in the order defined in the TCP or UDP specification, that is, the first byte of TCP or UDP source and destination port number in the received frame is in the higher byte of the register.

EMAC Station Management Interface (SMI)

The IEEE 802.3 MII station management interface, also known as the MDIO management interface, allows the processor to monitor and control one or more external Ethernet physical-layer transceivers. (Physical-layer transceivers are commonly called PHYs). The management interface physically consists of a 2-wire serial connection composed of the MDC (management data clock) output signal and the MDIO (management data input/output) bidirectional data signal. The IEEE 802.3 MII station management interface also applies to RMII and RGMII.

The application can address only one register in the PHY in any given time and send control data or receive status information. All the transfers are initiated by the EMAC CORE, and the PHY chip only acts as a slave device.

Standard PHY control and status registers typically provide

- Device capability status bits (for example: auto-negotiation, duplex modes, 10/100 speeds, and protocols)
- Device status bits (for example: auto-negotiation complete, link status, remote fault)
- Device control bits (for example: reset, speed selection, loopback, and auto-negotiation start)

Upon power-up, an MDIO read access (at default rates) of device capabilities in PHY status registers can determine the supported PHY features.

The MII management logical interface specifies:

- A set of 16-bit device control or status registers within the PHYs, including both required registers with standardized bit definitions as well as optional vendor-specified registers.
- A 5-bit device addressing scheme which allows the MAC to select one of up to 32 externally connected PHY devices.
- A 5-bit register addressing scheme for selecting the target register within the addressed device.
- A transfer frame protocol for 16-bit read and write accesses to PHY registers through the MDC and MDIO signals under control of the MAC.

Table 31-35: Station Management Interface pins

Station Management Interface Pins	Pin Description
MDIO – Management Data I/O	A periodic clock that runs at a maximum period of 400 ns. Always driven by the EMAC to PHY.
MDC – Management Data Clock	Data signal driven by EMAC or PHY, depending on write or read access based on EMAC; synchronous to MDC.

MDC Clock Frequency

The EMAC uses the `EMAC_SMI_ADDR.CR` bit field to determine the frequency of MDC as shown in the ***MDC Clock Frequency Selection*** table. The clock range selection determines the frequency of the clock relative to the SCLK0_0 frequency. The table shows the suggested range of SCLK0_0 frequency applicable for each value of the `EMAC_SMI_ADDR.CR` field. The programmability based on SCLK0_0 frequency range ensures that the MDC clock frequency range is within the IEEE specifications of 1.0 MHz to 2.4 MHz. However, the EMAC MDC can also support higher frequencies for PHY devices that support the frequencies.

Table 31-36: MDC Clock Frequency Selection

`EMAC_SMI_ADDR.CR` Selection	Programmed SCLK0_0 Frequency Range	Frequency of MDC	Min and Max MDC Freq (Per Specifications)
0000	60–100 MHz	SCLK0_0/42	MIN = 1.43 MHz and MAX = 2.39 MHz
0010	20–35 MHz	SCLK0_0/16	MIN = 1.25 MHz and MAX = 2.19 MHz
0011	35–60 MHz	SCLK0_0/26	MIN = 1.35 MHz and MAX = 2.31 MHz

The ***MDIO Frame Parameters*** table provides MDIO data transfer parameters. The write and read sequences provided in the tables, ***MDIO Write Data Sequence*** and ***MDIO Read Data Sequence***, are based on these parameters.

Table 31-37: MDIO Frame Parameters

Parameter	Description
IDLE	The MDIO line is three-state (noted as Z in sequence); there is no clock on MDC.
PREAMBLE	32 continuous bits, each of value 1
START	Start of frame is 01
OPCODE	10 for read and 01 for write
PHY ADDR	5-bit address select for one of 32 PHYs (noted as AAAAA in sequence)
REG ADDR	Register address in the selected PHY (noted as RRRRR in sequence)
TA	Turnaround is Z0 for read and 10 for write (Z = high impedance)
DATA	Any 16-bit value. Driven by MAC or PHY based on direction (noted as DDD...DDD).

Table 31-38: MDIO Write Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	01	AAAAA	RRRRR	10	DDD...DDD	Z

Table 31-39: MDIO Read Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	10	AAAAA	RRRRR	Z0	DDD...DDD	Z

SMI Write Operation

When programs set the `EMAC_SMI_ADDR.SMIW` (write) and `EMAC_SMI_ADDR.SMIB` (busy) bits, the station management interface (SMI) initiates a write operation into the PHY registers. The write operation uses the management frame format (the PHY address, the register address in PHY, and the write data) specified in the IEEE specifications. (Section 22.2.4.5 of IEEE standard). The application must not change the `EMAC_SMI_ADDR` register contents or the `EMAC_SMI_DATA` register while the transaction is ongoing.

Write operations to the `EMAC_SMI_ADDR` register or the `EMAC_SMI_DATA` register during the transfer period are ignored (while the `EMAC_SMI_ADDR.SMIB` bit is high). The transaction completes without error. After the write operation has completed, the SMI indicates the same by resetting the `EMAC_SMI_ADDR.SMIB` bit. The EMAC drives the MDIO line for the complete duration of the frame. The ***SMI Write Operation through MDIO/MDC Pins*** figure shows this operation.

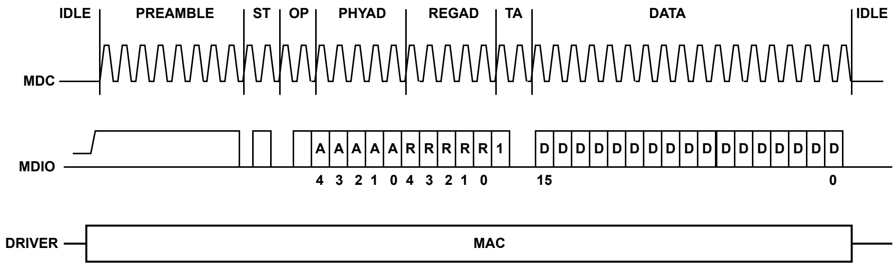


Figure 31-13: SMI Write Operation through MDIO/MDC Pins

SMI Read Operation

When programs set the `EMAC_SMI_ADDR.SMIB` bit with the `EMAC_SMI_ADDR.SMIW` bit cleared (=0), the station management interface (SMI) initiates a read operation in the PHY registers. It transfers the PHY address and the register address in the PHY to the SMI. The application must not change the `EMAC_SMI_ADDR` register contents or the `EMAC_SMI_DATA` register while the transaction is ongoing.

Write operations to the `EMAC_SMI_ADDR` register or the `EMAC_SMI_DATA` register during the transfer period are ignored (while the `EMAC_SMI_ADDR.SMIB` bit is high). The transaction completes without error. After the read operation has completed, the SMI resets the `EMAC_SMI_ADDR.SMIB` bit and updates the `EMAC_SMI_DATA` register with the data read from the PHY. The EMAC drives the MDIO line for the complete duration of the frame except during the data fields when the PHY drives the MDIO line. The ***SMI Read Operation through MDIO/MDC Pins*** figure shows this operation.

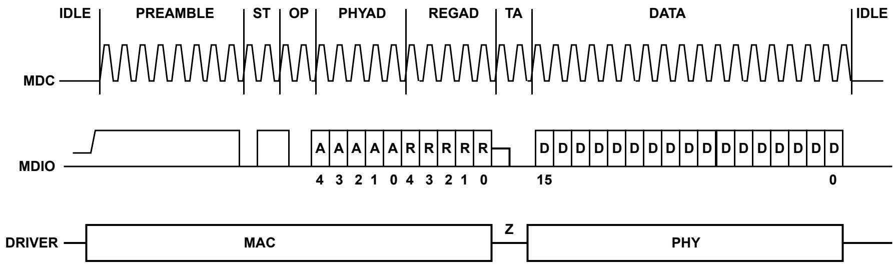


Figure 31-14: SMI Read Operation through MDIO/MDC Pins

EMAC Management Counters (MMC)

The EMAC provides a comprehensive set of 32-bit MAC management counters. It uses these counters for gathering statistics on the received and transmitted frames. The MMC subblock also includes

- A control register (`EMAC_MMC_CTL`) for managing the behavior of the counters
- Two 32-bit registers containing interrupts generated (`EMAC_MMC_RXINT` and `EMAC_MMC_TXINT`)
- Two 32-bit registers containing masks for the interrupt register (`EMAC_MMC_RXIMSK` and `EMAC_MMC_TXIMSK`)

The MMC receive counters are updated for frames passed by the address filtering subblock in the EMAC CORE. Statistics of frames dropped by the AFM module are not updated unless they are runt frames of less than 6 bytes. (Destination address bytes are not received fully.) The module is also capable of gathering statistics on encapsulated IPv4, IPv6, and TCP, UDP, or ICMP payloads in received Ethernet frames.

The MMC register naming conventions are as follows:

- Tx as a prefix or suffix indicates counters associated with transmission.
- Rx as a prefix or suffix indicates counters associated with reception.
- _G as a suffix indicates registers that count good frames only.
- _GB as a suffix indicates registers that count frames regardless of whether they are good or bad.

Transmitted frames are considered *good* when transmitted successfully. In other words, a transmitted frame is good if the frame transmission does not abort due to any of the following errors:

- Jabber timeout
- No carrier or loss of carrier
- Late collision
- Frame underflow
- Excessive deferral
- Excessive collision

Received frames are good when none of the following errors exists:

- CRC error
- Runt frame (shorter than 64 bytes)
- Alignment error
- Length error (non-type frames only)
- Out-of-range (non-type frames only, longer than maximum size)

The maximum frame size depends on the frame type, as follows:

- Untagged frame maxsize = 1518
- VLAN frame maxsize = 1522
- Jumbo frame maxsize = 9018
- Jumbo VLAN frame maxsize = 9022

The `EMAC_MMC_CTL` register also contains bits that control preset, freeze and roll-over of counters. The EMAC uses the `EMAC_MMC_CTL.RDRST` bit to enable an auto-reset feature whenever the counters are read. The EMAC uses the `EMAC_MMC_CTL.RST` bit to reset all the counters.

The MMC can trigger an interrupt when the corresponding bits are enabled in the transmit, receive, and IPC mask registers, and when the particular counter reaches half or full. The status is also updated in the corresponding interrupt register.

MMC Receive Interrupt Register

The `EMAC_MMC_RXINT` register maintains the interrupts that are generated when the receive statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When `EMAC_MMC_CTL.NOROLL` is set, then interrupts are set, but the counter remains at all ones. The `EMAC_MMC_RXINT` register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7:0) of the respective counter must be read to clear the interrupt bit.

MMC Transmit Interrupt Register

The `EMAC_MMC_TXINT` register maintains the interrupts generated when the transmit statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When `EMAC_MMC_CTL.NOROLL` is set, then interrupts are set, but the counter remains at all ones. The `EMAC_MMC_TXINT` register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7:0) of the respective counter must be read to clear the interrupt bit.

MMC Receive Checksum Offload Interrupt Register

The `EMAC_MMC_RXINT.CRCERR` register maintains the interrupts generated when receive IPC statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When `EMAC_MMC_CTL.NOROLL` is set, then interrupts are set, but the counter remains at all ones. The `EMAC_MMC_RXINT.CRCERR` register is 32 bits wide. When the MMC IPC counter that caused the interrupt is read, its corresponding interrupt bit is cleared. The least-significant byte lane (bits 7:0) of the counter must be read to clear the interrupt bit.

EMAC Precision Time Protocol (PTP) Engine

The following sections describe the precision time protocol (PTP) engine.

IEEE1588 and the PTP Engine

The Ethernet MAC peripheral includes a PTP engine to assist applications requiring time synchronization. The PTP module is tightly integrated with the EMAC CORE to aid hardware time stamping defined in the IEEE1588 2002/2008 standards. Applications can use accurate hardware time stamps through TCP/IP stacks (if using network layer communication) to exchange time information across devices connected over network. Applications can also use accurate hardware timestamps through Ethernet device drivers (if using MAC layer communication) to exchange time information.

PTP Engine

For calculation of drift in time between two Ethernet devices, the device records its system time whenever a timing message is sent or received (IEEE 1588 protocol). Due to the indeterministic delay of a software system for a node, the software is unable to capture an accurate time when the message is sent or received. However, the hardware can monitor the signal on the communication media and get an accurate message of arrival and departure time.

The PTP (precision time protocol) module is closely integrated with the EMAC module. It provides hardware assistance to implement both the IEEE 1588–2002 and IEEE 1588–2008 standards on Ethernet (IEEE 802.3). It takes one input clock signal as its PTP clock and maintains the timing information (called *system time*) at the nanosecond level.

The PTP module includes hardware for clock and system time adjustment. The pulse-per-second (PPS) signals physically represent the system time. PPS can be programmed to a fixed frequency or provide flexibility to the signal in terms of pulse width, interval, start, and stop time of the signals. The PTP module can be programmed to trigger an alarm interrupt when system time reaches specified time.

The PTP module can be programmed to detect different types of received frames, capture the system time, and time stamp those frames with the captured system time. Programs can configure any frame so that the PTP module capture the system time when it is transmitted. The PTP module can also capture the system time when an event is detected on the auxiliary snapshot trigger input pins (`EMAC_PTPAUXIN[n]`).

IEEE1588 Standard

Many systems require two independent devices to operate in a time synchronized fashion. If each system relied solely on its oscillator, differences between the characteristics and operating conditions of each oscillator would limit the ability of the clocks to operate synchronously. To serve applications requiring synchronized clocks, the system uses a periodic correction mechanism.

A simple way to synchronize multiple systems is to choose one system (with the best clock) as a master. The system master broadcasts the clock and timing information to other systems (slaves) and then the slaves adjust their clocks and timing according to that of master. However, this method has limitations. The master cannot broadcast the time at infinitesimal intervals. Path delay (propagation delay) exists between a master and a slave and the delay varies between each slave and master.

IEEE 1588 is also known as precision time protocol or PTP. The standard specifies a protocol used to synchronize the time and clock of multiple devices, dispersed but interconnected by any communication. For example, Ethernet (IEEE 802.3). According to the protocol, timing messages are exchanged between two devices. Then, one of the devices calculates its drift from other device and corrects its system time.(Both devices must have the same representation of their system time). The protocol resolves path delay between devices. It also helps synchronize the clocks of multiple devices and all of the limitations mentioned are resolved.

IEEE 1588 was published in 2002 where four types of timing messages were defined: Sync, Follow_Up, Delay_Req, and Delay_Resp. Here the protocol synchronizes two or more devices where one is a master and others are slaves. The master device sends Sync, Follow_Up, and Delay_Resp messages to the slave device in the system. The slave sends the Delay_Req messages to the master device. A following section provides more information on IEEE 1588–2002.

In 2008 a newer version of IEEE 1588 was introduced which provides further mechanisms to measure the peer-to-peer delay. Three more timing messages (PdelayReq, PdelayResp, and PdelayRespFollowup) were added to implement peer-to-peer synchronization. A following section provides more information on IEEE 1588–2008.

IEEE 1588–2002

The IEEE 1588–2002 standard defines the precision time protocol (PTP). The protocol allows precise synchronization of clocks in measurement and control systems that use network communication, local computing, and distributed objects. The protocol applies to systems that communicate by local area networks that support multicast messaging, including (but not limited to) Ethernet. This protocol also allows heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. The protocol supports system-wide synchronization accuracy in the submicrosecond range with minimal network and local clock computing resources.

The PTP is transported over UDP/IP. The system or network is classified into master and slave nodes for distributing the timing or clock information. The ***IEEE 1588–2002 PTP Process*** figure shows the process that PTP uses for synchronizing a slave node to a master node by exchanging PTP messages.

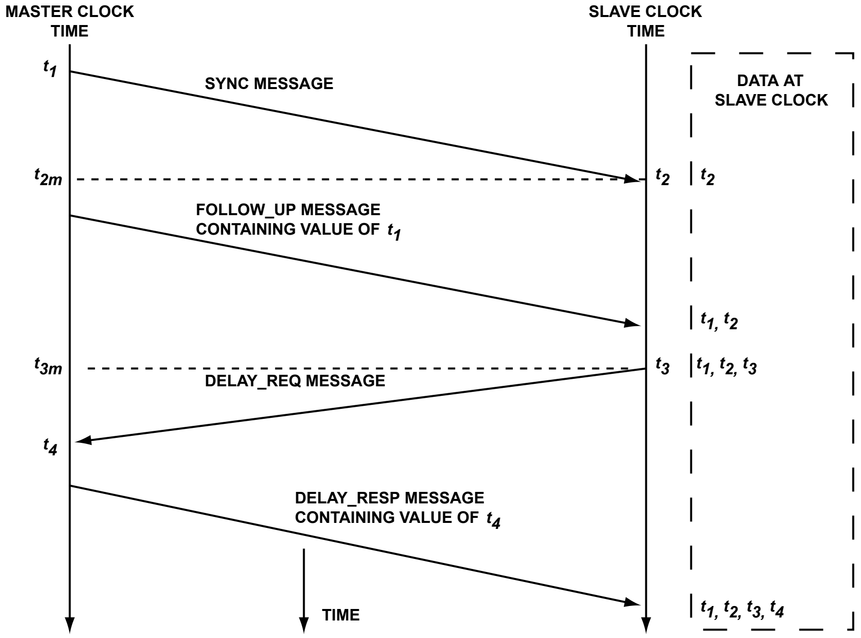

MASTER CLOCK TIME
SLAVE CLOCK TIME
DATA AT SLAVE CLOCK
t1
SYNC MESSAGE
t2m
t2
t2
FOLLOW_UP MESSAGE CONTAINING VALUE OF t1
t1, t2
t3m
t3
t1, t2, t3
DELAY_REQ MESSAGE
t4
DELAY_RESP MESSAGE CONTAINING VALUE OF t4
TIME
t1, t2, t3, t4

Figure 31-15: IEEE 1588–2002 PTP Process

As shown in the figure, the PTP uses the following process:

1. The master broadcasts the PTP sync messages to all its nodes. The sync message contains the reference time information of the master. The time at which this message leaves the system of the master is t_1. The master must capture this time for Ethernet ports, at RMII.
2. The slave receives the Sync message and also captures the exact time, t_2, using its timing reference.
3. The master sends a Follow_up message to the slave, which contains t1 information for later use.
4. The slave sends a Delay_Req message to the master, noting the exact time, t_3, at which this frame leaves the RMII.
5. The master receives the message, capturing the exact time, t_4, at which it enters its system.
6. The master sends the t_4 information to the slave in the Delay_Resp message.
7. The slave uses the four values of t_1, t_2, t_3, and t_4 to synchronize its local timing reference to the timing reference of the master.

Most of the PTP implementation happens in the software above the UDP layer. However, the hardware support must capture the exact time when specific PTP packets enter or leave the Ethernet port at the RMII/RGMII. Hardware must capture this timing information and return it to the software for the proper implementation of PTP with high accuracy.

IEEE 1588–2008 Advanced Time Stamps

In addition to the basic time stamp features mentioned in IEEE 1588–2002 time stamps, the EMAC supports the following advanced time stamp features defined in the IEEE 1588–2008 standard.

- Support for the IEEE 1588–2008 (Version 2) time stamp format.
- Provides an option to take snapshot of all frames or only PTP type frames.
- Provides an option to take snapshot of only event messages.
- Provides an option to select the node to be a master or slave.
- Identifies the PTP message type, version, and PTP payload in frames sent directly over Ethernet and sends the status.
- Provides an option to run nanoseconds time in digital or binary format.

Peer-to-Peer (P2P) PTP Message Support

The IEEE 1588–2008 version supports Peer-to-Peer PTP (Pdelay) message in addition to SYNC, Delay Request, Follow-up, and Delay Response messages. Refer to the ***Propagation Delay Calculation between Nodes Supporting Peer-to-Peer Path Correction*** figure. The figure shows the method to calculate the propagation delay between nodes supporting peer-to-peer path correction.

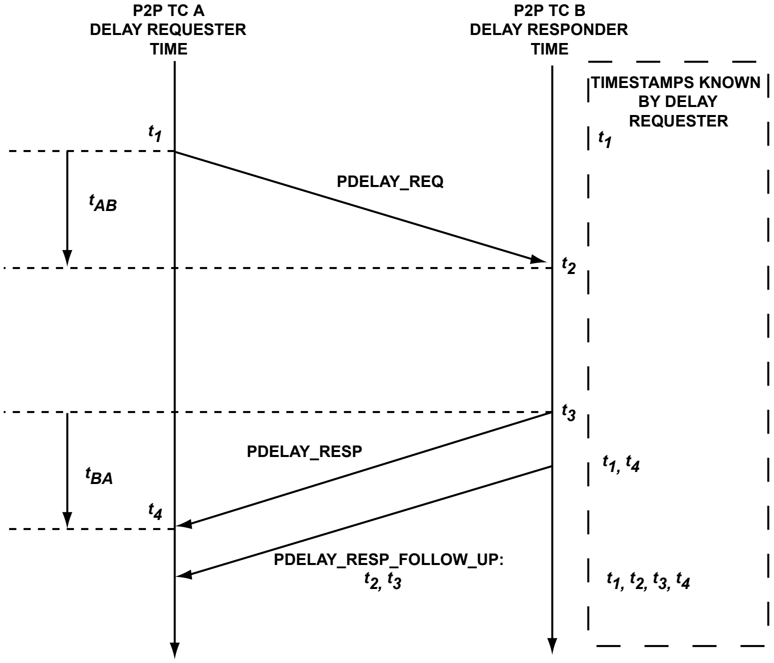

P2P TC A DELAY REQUESTER TIME
P2P TC B DELAY RESPONDER TIME
TIMESTAMPS KNOWN BY DELAY REQUESTER
t_1
t_{AB}
PDELAY_REQ
t_2
t_3
PDELAY_RESP
t_{BA}
t_4
t_1, t_4
PDELAY_RESP_FOLLOW_UP: t_2, t_3
t_1, t_2, t_3, t_4

Figure 31-16: Propagation Delay Calculation between Nodes Supporting Peer-to-Peer Path Correction

As shown in the figure, the propagation delay is calculated in the following way:

1. Port-1 issues a Pdelay_Req message and generates a time stamp, t1, for the Pdelay_Req message.
2. Port-2 receives the Pdelay_Req message and generates a time stamp, t2, for this message.
3. Port-2 returns a Pdelay_Resp message and generates a time stamp, t3, for this message. To minimize errors due to frequency offset between the two ports, Port-2 returns the Pdelay_Resp message as quickly as possible after the receipt of the Pdelay_Req message. The Port-2 returns any one of the following:
    - The difference between the time stamps t2 and t3 in the Pdelay_Resp message.
    - The difference between the time stamps t2 and t3 in the Pdelay_Resp_Follow_Up message.
    - The time stamps t2 and t3 in the Pdelay_Resp and Pdelay_Resp_Follow_Up messages respectively.
4. Port-1 generates a time stamp, t4, on receiving the Pdelay_Resp message.

Port-1 uses all four time stamps to compute the mean link delay.

Block Diagram

The ***PTP Block Diagram*** figure shows the functional block diagram of PTP module.

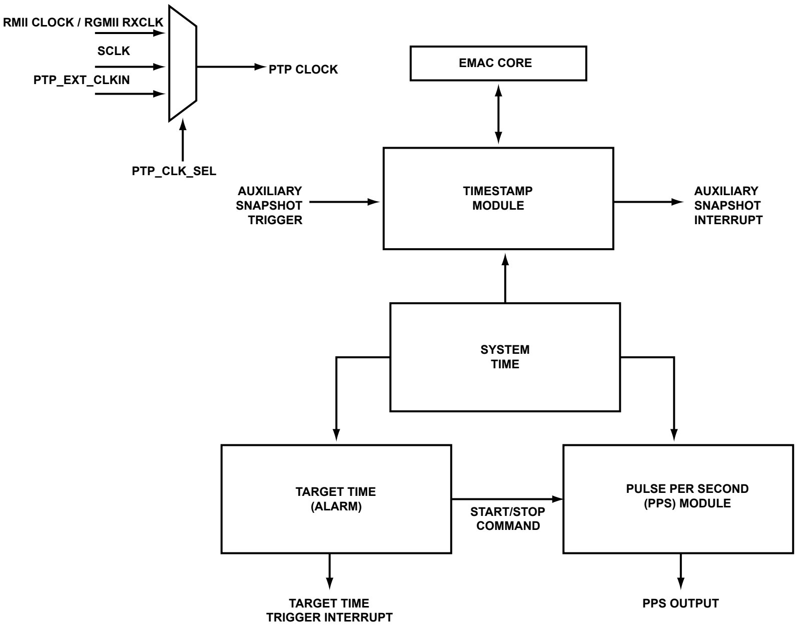

RMII CLOCK / RGMII RXCLK
SCLK
PTP_EXT_CLKIN
PTP CLOCK
PTP_CLK_SEL
EMAC CORE
AUXILIARY SNAPSHOT TRIGGER
TIMESTAMP MODULE
AUXILIARY SNAPSHOT INTERRUPT
SYSTEM TIME
TARGET TIME (ALARM)
START/STOP COMMAND
PULSE PER SECOND (PPS) MODULE
TARGET TIME TRIGGER INTERRUPT
PPS OUTPUT

Figure 31-17: PTP Block Diagram

A system time module is present which keeps the time of PTP module. It consists of hardware which can be programmed for time initialization, time correction, and clock correction.

The time stamp module captures the time (provided by the system time module) at various conditions. For example, when the EMAC sends or receives a frame or during the rising edge of the auxiliary snapshot trigger (`EMAC_PTPAUXIN[n]`) pins. When system time is captured after detection of a frame, the time stamp module automatically includes the time information in the frame descriptor. Time stamping on the detection of a frame can be programmed on a per frame basis.

The PTP clock drives the PTP module. This clock can be selected from three different clock sources.

The pulse per second (PPS) module generates a pulse or train of pulse on the PPS output pins, (`EMAC_PTPPPS[n]`). It is the physical representation of system time. PPS can be fixed (where only frequency varies) or flexible (where width, interval, start time, and stop time can be programmed).

The target time module acts as an alarm for the PTP module. Whenever system time reaches a value equal to programmed target time, the target time trigger interrupt is generated. By appropriate programming, The target time trigger can also start or stop flexible PPS output at specific time.

PTP Module Clock

The PTP module clock features include Clock Source Selection and Clock Frequency Range.

Clock Source Selection

The PTP module can take one of these clock sources as its input clock — SCLK0_0, RMII reference RGMII Rx clock, or PTP external clock.

As shown in the ***PTP Clock Source Selection*** table, the `PADS_PCFG0` register selects the PTP clock source.

Table 31-40: PTP Clock Source Selection

`PADS_PCFG0`	PTP Clock Source	Clock Description
00	EMAC_RMII	RMII reference clock
00	RGMII_RXCLK	RGMII Rx Clock
10	PTP external clock	Clock available on `EMAC_PTPCLKIN[n]` pin
X1	SCLK0_0	Processor system clock

Clock Frequency Range

The resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Therefore, a higher PTP clock frequency gives better system performance. The timing constraints achievable for logic operating on the selected PTP clock source limit the maximum PTP clock frequency.

The minimum PTP clock frequency depends on the time required between two consecutive frames. The IEEE specification determines the RMII clock frequency. The minimum PTP clock frequency required for proper operation depends on the operating mode and operating speed of the MAC. See the ***Minimum PTP Clock Frequency*** table.

A minimum delay required between two consecutive time stamp captures is 8 clock cycles of RMII or 4 clock cycles of RGMIIand 3 clock cycles of PTP clocks. If the delay between two time stamp captures is less than this delay, the EMAC does not take a time stamp snapshot for the second frame.

The following table assumes:

- Minimum Ethernet packet size = 64 bytes

- Miminum IFG = 96 bit times = 12 bytes
- Preamble = 8 bytes
- 3 PTP Clock + 4GMII/MII Clock Min gap between two SFDs (start of frame delimiters)
- 3 PTP Clock + 8 RMII Clock Min gap between two SFDs
- 3 PTP Clock + 4 RGMII Clock Min gap between two SFDs

Mode (Full Duplex in all cases)	Minimum Gap between two SFDs	PTP clock	Comments
10 Mbps RMII	64 bytes of data + 8 bytes of preamble + min IFG	3 PTP clock cycle + 8RMII clock cycle 3360 RMII clock cycles	In RMII @ 10 Mbps, 1 byte transmitted in 40 clock cycles
	(2560 + 320 + 480) RMII clock cycles	3 PTP clock cycle 3352 RMII clock cycles	RMII clock = 50 MHz
	3360 RMII clocks	PTP clock cycle 1117.33 20 ns = 22346 ns	1 RMII clock cycle = (1/50 MHz) = 20 ns
		PTP frequency min = 0.045 MHz	
10 Mbps RGMII	64 bytes of data + 8 bytes of preamble + min IFG	3 PTP clock cycle + 4RGMII clock cycle 168 RGMII clock cycles	In RGMII, 1 byte transmitted in 4 clock cycles
	(128 + 16 + 24) MII clock cycles	3 PTP clock cycle<= 164 RGMII clock cycles	RGMII clock at 100 Mbps = 2.5 MHz
	168 MII clock cycles	PTP clock cycle 54.67 400ns = 21860 ns	1 RGMII clock cycle = (1/2.5 MHz) = 400 ns
		PTP frequency min = 0.045 MHz	
100 Mbps RMII	64 bytes of data + 8 bytes of preamble + min IFG	3 PTP clock cycle + 8RMII clock cycle 336 RMII clock cycles	In RMII, 1 byte transmitted in 4 clock cycles
	(256 + 32 + 48) RMII clock cycles	3 PTP clock cycle 328 RMII clock cycles	RMII clock = 50 MHz
	336 RMII clocks	PTP clock cycle 109.33 20 ns = 2186 ns	1 RMII clock cycle = (1/50 MHz) = 20 ns
		PTP frequency min = 0.45 MHz	
100 Mbps RGMII	64 bytes of data + 8 bytes of preamble + min IFG	3 PTP clock cycle + 4RGMII clock cycle 168 RGMII clock cycles	In RGMII, 1 byte transmitted in 4 clock cycles
	(128 + 16 + 24) MII clock cycles	3 PTP clock cycle 164 RGMII clock cycles	RGMII clock at 100 Mbps = 25 MHz
	168 MII clock cycles	PTP clock cycle 54.67 40 ns = 2186 ns	1 RGMII clock cycle = (1/25 MHz) = 40 ns
		PTP frequency min = 0.45 MHz	

Mode (Full Duplex in all cases)	Minimum Gap between two SFDs	PTP clock	Comments
1000 Mbps RGMII	64 bytes of data + 8 bytes of preamble + min IFG	3 PTP clock cycle + 4 RGMII clock cycle 84 RGMII clock cycles	In RGMII @ 1000 Mbps, 1 byte transmitted in 1 clock cycles
	(64 + 8 + 12) RGMII clock cycles	3 PTP clock cycle 80 RGMII clock cycles	RGMII clock at 100 Mbps = 125 MHz
	84 RGMII clocks	PTP clock cycle 26.67 8 ns = 213 ns	1 RGMII clock cycle = (1/125 MHz) = 8 ns
		PTP frequency min = 4.6 MHz	

The minimum PTP clock frequency also depends on the maximum value of the `EMAC_TM_SUBSEC` register, so that even at the highest `EMAC_TM_SUBSEC.SSINC` value, the `EMAC_TM_SEC` register value can be incremented every second. Since the `EMAC_TM_SUBSEC.SSINC` is an 8-bit field, the minimum PTP clock frequency allowed is approximately 4 MHz.

Time Stamp Module

The time stamp module captures time in seconds and nanoseconds maintained as system time. The time stamp module also captures time when specific events occur. Events include detection of a frame transmitted or received over the EMAC and a rising edge on the `EMAC_PTPAUXIN[n]` pins. The time stamp module does not need to time stamp all of the transmitted or received frames over the EMAC. The PTP module can be programmed to detect specific kinds of frames for time stamping.

Frame Detection and Time Stamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If the module detects an event message, it takes a snapshot of the system time. The PTP module stores the value to the 64-bit fields in transmit or receive descriptor.

The time stamping occurs at the EMAC RMII/RGMII interface when the module sees the start of frame of an event message packet.

Transmit Path Time Stamping

The EMAC captures a time stamp when a frame transmits on the RMII/RGMII. Time stamp capture is controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a time stamp is captured for that frame or not.

Applications can extend the descriptor word length from 4 words to 8 words by setting the `EMAC_DMA0_BUSMODE.ATDS` bit. To enable the time stamp function, set the TTSE (transmit time stamp enable) bit in transmit descriptor word TDES0. When the PTP module captures a time stamp of a transmitted frame, it notifies the application by setting the TTSS (transmit time stamp status) in TDES0.

The EMAC returns the time stamp to the software inside the corresponding transmit descriptor, automatically connecting the time stamp to the specific frame. The 64-bit time stamp information is written to the TDES6 and TDES7 fields. The TDES6 field holds the 32-bit LSBs of the time stamp (system time nanoseconds), except as described in transmit time stamp field, and the TDES7 field holds the 32-bit MSBs (system time seconds). After the

PTP module time stamps the frame, the application can get the time stamp along with the transmit status from the EMAC.

NOTE: The PTP module time stamps all the transmitting frames having TTSE set in its TDES0. It does not distinguish according to the type of transmitting frame.

Receive Path Time Stamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If an event message is detected, the module takes a snapshot of the system time. The module stores its value to the 64-bit fields in transmit or receive descriptor. The time stamping is done at the EMAC RMII/RGMII interface when the module sees the start of frame of an event message packet.

PTP module captures the time stamp of received frames on the RMII/RGMII. Time stamp capture is controllable on a per-frame and per-type basis. In other words, each received frame is time stamped according to the frame type.

Applications can extend the descriptor word length from 4 words to 8 words by setting the `EMAC_DMA0_BUSMODE.ATDS` bit to store time stamp and received message status. The PTP notifies the application of receive time stamp availability when it sets bit 7 (time stamp available) in receive descriptor word RDES0.

When bit 0 (extended status available) is set in RDES0, it indicates that the extended status of the PTP frame is provided in the RDES4 word. Extended status includes PTP version, PTP frame type, and message type. The EMAC returns the time stamp to the software inside the corresponding receive descriptor. The 64-bit time stamp information is written back to the RDES6 and RDES7 fields in memory. The RDES6 holds the 32-bit LSBs of the time stamp (system time nanoseconds), except as mentioned in receive time stamp field, and the RDES7 field holds 32-bit MSBs (system time seconds).

The time stamp is written only to that receive descriptor for which the last descriptor status field has been set to 1. When the time stamp is not available (for example, because of an RxFIFO overflow), an all-ones pattern is written to the descriptors (RDES6 and RDES7). The write operation indicates that the time stamp is not correct. RDES0 [7] indicates whether the time stamp is updated in RDES6/7.

The PTP module processes received frames to identify valid PTP frames. Use the `EMAC_TM_CTL` register to control the snapshot of the time sent to the application.

The PTP module can be programmed to detect all received frames or only some types of PTP frames, according to bit settings in the `EMAC_TM_CTL` register. Refer to the ***PTP Frame Type Selections*** table.

Table 31-41: PTP Frame Type Selections

TSENALL (bit 8)	SNAPTYPSEL (bits [17:16])	TSMSTRENA (bit 15)	TSEVNTENA (bit 14)	Frames
1	X	X	X	All
0	00	X	0	Sync, Follow_Up, Delay_Req, Delay_Resp
0	00	0	1	Sync
0	00	1	1	Delay_Req

Table 31-41: PTP Frame Type Selections (Continued)

TSENALL (bit 8)	SNAPTYPSEL (bits [17:16])	TSMSTRENA (bit 15)	TSEVNTENA (bit 14)	Frames
0	01	X	0	Sync, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_ Follow_Up
0	01	0	1	Sync, Pdelay_Req, Pdelay_Resp
0	01	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
0	10	X	X	Sync, Delay_Req
0	11	X	X	Pdelay_Req, Pdelay_Resp

PTP Processing and Control

When the EMAC receives a frame, frame detection and time stamping are based on some of the PTP fields in the frame. The ***PTP Message Format (IEEE 1588–2008)*** table shows the common message header for the PTP messages. This format is derived from the IEEE standard 1588–2008. When the EMAC sends a PTP frame, the frame follows this format.

When a frame is received, the PTP module compares these fields with standard values and finds out the type of PTP frame and other information. For example, PTP version, IP version, and others. The module then updates the related fields in RDES4. When a frame is transmitted, programs must ensure that all the fields are appropriate. The PTP module on the other end of a communication must correctly detect and decode the frame.

Table 31-42: PTP Message Format (IEEE 1588–2008)

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
transportSpecific				messageType				1	0
Reserved				versionPTP				1	1
messageLength								2	2
domainNumber								1	4
Reserved								1	5
flagField								2	6
correctionField								8	8
Reserved								4	16
sourcePortIdentity								10	20
sequenceId								2	30

Table 31-42: PTP Message Format (IEEE 1588–2008) (Continued)

Bits	Octets	Offset
controlField (used in version 1. For version 2, messageType field is used for detecting different message types.)	1	32
logMessageInterva	1	33

There are some fields in the Ethernet payload that can be used to detect the PTP packet type and control the snapshot taken. These fields are different for the following PTP frames:

- PTP Frames Over IPv4
- PTP Frames Over IPv6
- PTP Frames Over Ethernet

For these PTP frames, EMAC does not consider the PTP version 1 messages as valid when the frame consists of peer delay multicast address as destination address (DA).

PTP Frame Over IPv4

The ***IPv4-UDP PTP Frame Fields Required for Control and Status*** table provides information about the fields that are matched to control snapshot for the PTP packets. The packets are sent over UDP over IPv4 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format. The format is defined in the ***PTP Message Format (IEEE 1588–2008)*** table in the PTP Processing and Control section.

Table 31-43: IPv4-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x0800	IPv4 datagram
IP Version and Header Length	14	0x45	IP version is IPv4
Layer 4 Protocol	23	0x11	UDP
IP Multicast Address (IEEE 1588 Version 1)	30, 31, 32, 33	0xE0,0x00, 0x01,0x81 (or 0x82 or 0x83 or 0x84)	Multicast IPv4 addresses allowed. 224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132
IP Multicast Address (IEEE 1588 Version 2)	30, 31, 32, 33	0xE0, 0x00, 0x01, 0x81 (Hex) 0xE0, 0x00, 0x00, 0x6B (Hex)	PTP-Primary multicast address: 224.0.1.129 PTP-Peer delay multicast address: 224.0.0.107

Table 31-43: IPv4-UDP PTP Frame Fields Required for Control and Status (Continued)

Field Matched	Octet Position	Matched Value	Description
UDP Destination Port	36, 37	0x013F 0x0140	0x013F - PTP Event Messages. These are SYNC, Delay_Req (IEEE 1588 version 1 and 2) or Pdelay_Req, Pdelay_Resp (IEEE 1588 version 2 only). 0x0140 - PTP general messages
PTP Control Field (IEEE version 1)	74	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	42 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	43 (nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over IPv6

The ***IPv6-UDP PTP Frame Fields Required for Control And Status*** table provides information about the fields that are matched to control the snapshots for the PTP packets. The packets are sent over UDP over IPv6 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format defined in PTP Message Format (IEEE 1588–2008).

Table 31-44: IPv6-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x86DD	IP datagram
IP Version	14 (bits [7:4])	0x06	IP version is IPv6
Layer 4 Protocol	20 (IPv6 extension header not defined for PTP packets)	0x11	UDP

Table 31-44: IPv6-UDP PTP Frame Fields Required for Control and Status (Continued)

Field Matched	Octet Position	Matched Value	Description
PTP Multicast Address	38–53	FF0x:0:0:0:0:0:0:0:0:181 (Hex) FF02:0:0:0:0:0:0:0:0:6B (Hex)	PTP - primary multicast address: FF0x: 0:0:0:0:0:0:0:0:181 (Hex) PTP - Peer delay multicast address: FF02:0:0:0:0:0:0:0:0:6B (Hex)
UDP Destination Port	56, 57 (IPv6 extension header not defined for PTP packets)	0x013F, 0x0140	0x013F - PTP event messages 0x0140 - PTP general messages
PTP Control Field (IEEE 1588 version 1)	93 (IPv6 extension header not defined for PTP packets)	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management (version1)
PTP Message Type Field (IEEE version 2)	74 (nibble) (IPv6 extension header not defined for PTP packets)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	75 (nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over Ethernet

Refer to the ***Ethernet PTP Frame Fields Required for Control and Status*** table. The table provides information about the fields that are matched to control the snapshots for the PTP packets sent over Ethernet for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. The positions are based on IEEE 1588–2008 standards and the message format defined in the table.

Table 31-45: Ethernet PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched value	Description
MAC Destination Multicast Address (The address match of destination address (DA) programmed in MAC address 0 is used when the EMAC_TM_CTL.TSENMACADDR bit is set)	0–5	01-1B-19-00-00-00 01-80-C2-00-00-0E	All PTP messages can use any of the following multicast addresses: 01-1B-19-00-00-00 01-80-C2-00-00-0E
MAC Frame Type	12, 13	0x88F7	PTP Ethernet frame
PTP control field (IEEE Version 1)	45	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	14 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	15(nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

Auxiliary Time Stamp Snapshot

The auxiliary snapshot feature stores snapshots of the system time whenever a rising edge is detected on the EMAC_PTPAUXIN[n] pins.

The PTP stores 64 bits of captured time stamp in a 4-deep FIFO. When a snapshot is stored, the PTP indicates this event to the EMAC with the auxiliary snapshot interrupt. The EMAC_TM_STMPSTAT.ATSTS bit is set. The value of the snapshot is read through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers. If the FIFO becomes full and an external trigger to take the snapshot is asserted, then the snapshot trigger-missed status is set in the EMAC_TM_STMPSTAT.ATSSTM bit. The latest snapshot is not written to the FIFO when it is full.

When a host reads the 64-bit time stamp from the FIFO through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers, the space becomes available to store the next snapshot.

NOTE: A space in the FIFO is created whenever the `EMAC_TM_AUXSTMP_SEC` register is read. Therefore, read the `EMAC_TM_AUXSTMP_NSEC` register before reading the `EMAC_TM_AUXSTMP_SEC` register.

The program can clear the FIFO by setting the `EMAC_TM_CTL.ATSFC` bit. When multiple snapshots are present in the FIFO, the `EMAC_TM_STMPSTAT.ATSNS` bits indicate the count.

NOTE: The minimum gap between two events on the `EMAC_PTPAUXIN[n]` pin must be 4 cycles of PTP_CLK + 3 cycles of `SCLK0_0`). Otherwise, the logic misses the rising-edge of the trigger.

System Time

To get a snapshot of the time, the EMAC requires a reference time in 64-bit format as defined in the IEEE 1588 specification. The PTP module maintains 80-bit time, known as system time. The PTP clock updates system time.

The 80-bit timing reference is split into the following three registers:

- `EMAC_TM_NSEC` – 32-bit nanoseconds register which provides time in nanoseconds
- `EMAC_TM_SEC` – 32-bit seconds register which provides time in seconds
- `EMAC_TM_HISEC` – 16-bit high seconds register which provides time beyond the seconds register. The IEEE 1588 standard does not include this register. Its use is application-specific.

The 64-bit system time (seconds and nanoseconds) is the source for taking time stamps for Ethernet frames being transmitted or received at the RMII.

Since the PTP clock frequency does not correspond to a 1ns period, the `EMAC_TM_NSEC` register is incremented with a value equal to the PTP clock period for every PTP clock cycle. The function uses the `EMAC_TM_SUBSEC` register. The `EMAC_TM_NSEC` value is incremented with the value programmed in `EMAC_TM_SUBSEC` register every PTP clock cycle.

Whenever the `EMAC_TM_SEC` register overflows from 0xFFFFFFFF to 0x00000000, the seconds overflow interrupt is triggered. The EMAC uses the `EMAC_TM_STMPSTAT.TSSOVF` bit to indicate the event. After a seconds overflow, the `EMAC_TM_HISEC` register increments by one.

The system time module supports the following two types of rollover modes for the `EMAC_TM_NSEC` register.

- Digital rollover mode. The maximum value in the nanoseconds field is 0x3B9AC9FF, that is, 10^9 nanoseconds. After it reaches this value, the `EMAC_TM_SEC` register increments and the `EMAC_TM_NSEC` register restarts counting from zero. Accuracy in digital rollover mode it is 1 ns per bit.
- Binary rollover mode. The nanoseconds field rolls over and increments the seconds field after the value reaches 0x7FFFFFFF. Accuracy in binary rollover mode is ~0.466 ns per bit.

System Time Adjustment

The following sections describe the process for system time adjustment.

System Time Initialization

System time can be initialized with 64-bit time when the PTP module is enabled. The initial value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT system time update registers. The system time counter is written with the value in the registers when the EMAC_TM_CTL.TSINIT bit is set.

Coarse Correction Method

If slave system time has an offset based on the system time of the master, then the coarse correction method can correct it. The time offset value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers. The offset value is then added to or subtracted from the system time when the EMAC_TM_CTL.TSUPDT bit is set. Use the EMAC_TM_NSECUPDT.ADDSUB bit to choose addition or subtraction. System time correction occurs in one clock cycle using the coarse correction method.

NOTE: During subtraction, the EMAC_TM_SECUPDT register value must be less than the value of the EMAC_TM_SEC register. Check the value prior to subtracting using coarse correction.

Fine Correction Method

If a slave PTP clock frequency has a drift based on the master PTP clock (as defined in IEEE 1588), it can be corrected using the fine correction method. Using this method, system time is corrected over a period (unlike coarse correction where it happens in one clock cycle). This correction helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP sync message intervals.

Using this method, an accumulator sums the contents of the EMAC_TM_ADDEND register. The ***System Time Update, Fine Correction Method*** figure shows the method. The arithmetic-carry that the accumulator generates acts as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency divider.

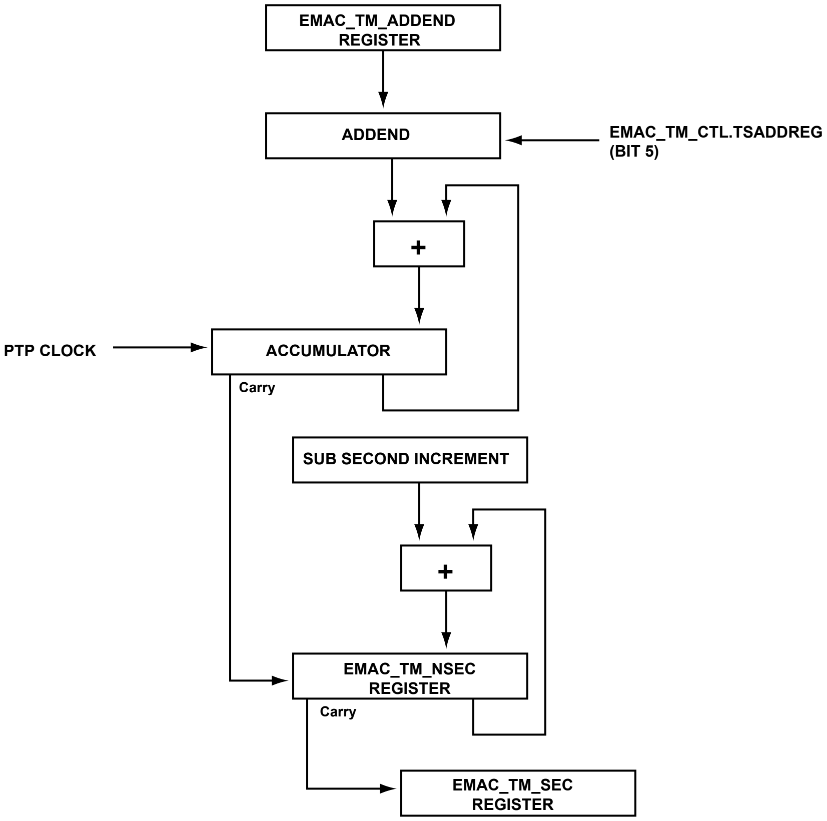

EMAC_TM_ADDEND REGISTER
ADDEND
EMAC_TM_CTL.TSADDREG (BIT 5)
+
PTP CLOCK
ACCUMULATOR
Carry
SUB SECOND INCREMENT
+
EMAC_TM_NSEC REGISTER
Carry
EMAC_TM_SEC REGISTER

Figure 31-18: System Time Update, Fine Correction Method

Calculating Addend Value

This section describes the fine correction process for system time.

In this example, the master clock runs at 50 MHz and the slave clock has drifted to 66 MHz. The goal is to adjust the slave system time to 50 MHz, so that the slave PTP module synchronizes with the master. Using the figure in Fine Correction Method, the nanoseconds increment signal runs at 50 MHz. The nanoseconds increment is the carry from accumulator register. The addend value increments the carry from accumulator at the rate of the slave clock (66 MHz).

The accumulator overflows and generates a carry every N addend value, so $N \times \text{Addend} = 2^{32}$.

The accumulator increments at 66 MHz. This addition brings the carry to 50 MHz $N = 66/50 = 1.32$.

Hence, the addend = $2^{32}/1.32$ = 0xC1F07C1F.

Therefore, if addend is programmed with 0xC1F07C1F, the slave system time runs at 50 MHz which synchronizes with the master.

In the Fine Correction Method figure, the subsecond increment is the value programmed in the `EMAC_TM_SUBSEC` register which increments the `EMAC_TM_NSEC` register according to the frequency of the nanoseconds increment signal.

In the example, the sub second increment is 20 (for digital rollover) or 43 (for binary rollover). This addition increments the `EMAC_TM_NSEC` register by 20 ns (1/50 MHz).

The software must calculate the drift in frequency and update the `EMAC_TM_ADDEND` register accordingly.

NOTE: The PTP reference clock is the clock at which the system time is updated. When the `EMAC_TM_CTL.TSCFUPDT` bit is set to 0, this clock equals the PTP clock. Using fine correction, the PTP reference clock is generated on the nanoseconds increment signal at which the system time updates.

Target Time Trigger (Alarm)

The PTP module provides an alarm function by triggering an alarm at a preset time. It sets the `EMAC_TM_STMPSTAT.TSTARGT0` bit when the system time matches the value of the `EMAC_TM_PPS0TGTM` and `EMAC_TM_PPS0NTGTM` registers. This trigger can generate an interrupt and command the flexible PPS module to start or stop PPS output, depending on value programmed in `EMAC_TM_PPSCTL.TRGTMODSEL0` bits.

The trigger is enabled by setting `EMAC_TM_CTL.TSTRIG` bit. Once an alarm has occurred, if the PTP needs another alarm, the software must:

- Clear the status bit
- Reprogram the `EMAC_TM_PPS0TGTM` and `EMAC_TM_PPS0NTGTM` registers to a future value, and
- Set the `EMAC_TM_CTL.TSTRIG` bit

If the time programmed in the target time registers has elapsed, then a target time programming error is indicated by setting the `EMAC_TM_STMPSTAT.TSTRGTERR0` bit.

The alarm time is represented in absolute units, not relative units. For example, if the software must generate an alarm after 10 seconds, it must read the current system time value. Then, the software must add the number corresponding to 10 seconds, and write the result back to the target time registers.

Pulse-Per-Second (PPS)

Pulse-per-second (PPS) is a physical representation of system time. It consists of a single pulse or train of pulses. The PTP uses PPS for extra synchronization or to monitor the synchronization performance between clocks. With proper configuration, the PTP module can generate PPS signals that are output on the `EMAC_PTPPPS[n]` pins. The PTP supports two kinds of PPS output, fixed and flexible.

Fixed Pulse-Per-Second Output

The EMAC supports fixed pulse-per-second (PPS) output that indicates 1-second intervals (default). Change the frequency of the PPS output by configuring the `EMAC_TM_PPSCTL.PPSCTL0` bits. The default value for these bits is 0000, which configures a 1Hz signal with a pulse width equal to the period of the PTP clock.

The ***PPS Output Frequencies*** table shows various PPS output frequencies.

Table 31-46: PPS Output Frequencies

PPSCTL Bit Setting	Binary Rollover	Digital Rollover
0001	2 Hz	1 Hz

Table 31-46: PPS Output Frequencies (Continued)

PPSCTL Bit Setting	Binary Rollover	Digital Rollover
0010	4 Hz	2 Hz
0011	8 Hz	4 Hz
...	...	...
1111	32.768 kHz	16.384 kHz

In binary rollover mode, the PPS output has a duty cycle of 50% with these frequencies.

In digital rollover mode, the PPS output frequency is an average number. The actual clock is a different frequency that is synchronized every second. PPS output pulses have different periods and duty cycles and this behavior is because of the non-linear toggling of the bits in digital rollover mode. For example:

- When `EMAC_TM_PPSCTL.PPSCTL0` = 0001, the PPS (1 Hz) has a low period of 537 ms and a high period of 463 ms.
- When `EMAC_TM_PPSCTL.PPSCTL0` = 0010, the PPS (2 Hz) is a sequence of:
  - One clock of 50-percent duty cycle and 537-ms period
  - Second clock of 463-ms period (268 ms low and 195 ms high)
- When `EMAC_TM_PPSCTL.PPSCTL0` = 0011, the PPS (4 Hz) is a sequence of:
  - Three clocks of 50-percent duty cycle and 268-ms period
  - Fourth clock of 195-ms period (134 ms low and 61 ms high)

Flexible Pulse-Per-Second Output

The EMAC also provides the flexibility to program the start or stop time, width, and interval of the pulse generated on the PPS output. Enable this feature, called flexible PPS, by setting the `EMAC_TM_PPSCTL.PPSEN` bit.

The flexible PPS output options are:

- Supports programming the start point of the single pulse and start and stop points of the pulse train in terms of system time. The target time registers program the start and stop time.
- Supports programming the stop time in advance. Programs can configure the stop time before the actual start time has elapsed.
- Supports programming the width, between the rising edge and corresponding falling edge of the PPS signal output, in terms of number of units of subsecond increment. This value is configured in the `EMAC_TM_SUBSEC` register.
- Supports programming the interval, between the rising edges of PPS signal, in terms of number of units of subsecond increment. This value is configured in the `EMAC_TM_SUBSEC` register.
- Provides the option to cancel the programmed PPS start or stop request.

- Indicates error if the start or stop time programmed has already elapsed.

PPS Start or Stop Time

Start time can initially be programmed in the target time registers. If necessary, the start or stop time can be programmed again but only after the earlier programmed value is synchronized to the PTP clock domain. The `EMAC_TM_PPS0NTGTM.TSTRBUSY` bit indicates the status of synchronization. Programs can configure the start or stop time in advance, even before the earlier stop or start time has elapsed.

Program the start or stop time with advanced system time to ensure proper PPS signal output. If the application programs a start or stop time that has already elapsed, then the EMAC sets the `EMAC_TM_STMPSTAT.TSTRGTERR0` bit, indicating the error. If enabled, the EMAC also sets the target time trigger (alarm) interrupt event. The application can cancel the start or stop request only if the corresponding start or stop time has not elapsed. If the time has elapsed, the cancel command has no effect.

PPS Width and Interval

The PPS width and interval are programmed in terms of granularity of system time, that is, the number of the units of subsecond increment value. For example, with the PTP reference clock of 50 MHz, a PPS pulse width of 40 ns, and an interval of 100 ns, program the width and interval to 2 and 5, respectively.

Use a faster PTP reference clock to achieve smaller granularity. Before commanding to trigger a pulse or pulse train on the PPS output, programs must configure or update the interval and width of the PPS signal output.

PPS Command

When the PPS module has a flexible PPS output configuration, the PTP can use the `EMAC_TM_PPSCTL.PPSCTL0` bits to command the PPS module to use any of the flexible PPS features.

Programming these bits with a non-zero value instructs the PPS module to initiate an event. Once the command transfers or synchronizes to the PTP clock domain, these bits clear automatically. Software must ensure that these bits are programmed only when they are all-zero.

The ***Flexible PPS Output Commands*** table explains the different commands and their function.

Table 31-47: Flexible PPS Output Commands

PPSCTL (Bits 3–0)	Command	Description
0000	No Command	
0001	Start Single Pulse	Generates single pulse rising at start point defined in target time registers and of duration defined in `EMAC_TM_PPS0WIDTH` register.
0010	Start Pulse Train	Generates train of pulses rising at the start time configured in the target time registers, of duration configured in the `EMAC_TM_PPS0WIDTH` register. The train of pulses repeats at the interval configured in the `EMAC_TM_PPS0INTVL` register. By default, the PPS pulse train is free-running unless stopped by stop pulse train at time or stop pulse train immediately commands.
0011	Cancel Start	Cancels the start single pulse and start pulse train commands when the system time has not crossed the programmed start time.

Table 31-47: Flexible PPS Output Commands (Continued)

PPSCTL (Bits 3–0)	Command	Description
0100	Stop Pulse Train at Time	Stops the train of pulses initiated by the command for start pulse train after the time programmed in the target time registers elapses.
0101	Stop Pulse Train Immediately	Immediately stops the train of pulses initiated by the command for start pulse train.
0110	Cancel Stop Pulse Train	Cancels the stop pulse train at time command when the programmed stop time has not elapsed. The PPS pulse train becomes free-running on the successful execution of this command.
0111-1111	Reserved	

PTP Interrupts

Set the `EMAC_IMSK.TS` bit to enable interrupts from the PTP module. The EMAC uses the `EMAC_ISTAT.TS` bit to indicate the status of the interrupt. The PTP supports the following three types of interrupts.

Auxiliary Snapshot Trigger

When an external event occurs on the `EMAC_PTPAUXIN[n]` pins and a time stamp snapshot occurs, an auxiliary snapshot interrupt is triggered. The EMAC uses the `EMAC_TM_STMPSTAT.ATSTS` bit to indicate the interrupt.

Target Time Reached

This interrupt is triggered when the system time becomes equal to the value written in the `EMAC_TM_PPS0NTGTM` and `EMAC_TM_PPS0NTGTM` registers. Enable or disable the interrupt using the `EMAC_TM_CTL.TSTRIG` and `EMAC_TM_PPSCTL.TRGTMODSEL0` bits. This interrupt can be used as an alarm and is indicated on the `EMAC_TM_STMPSTAT.TSTARGT0` bit.

System Time Seconds Register Overflow

This interrupt is triggered when the `EMAC_TM_SEC` register overflows from 0xFFFF FFFF to 0x0000 0000. This interrupt is indicated on the `EMAC_TM_STMPSTAT.TSSOVF` bit. As soon as `EMAC_TM_SEC` register overflows, the `EMAC_TM_HISEC` register increments by one.

Audio Video Data Transmission

The audio video (AV) feature enables transmission of time-sensitive traffic over bridged local area networks (LANs). The following standards define the various aspects of the AV feature implementation.

- IEEE 802.1Qav-2009: Allows the bridges to provide time-sensitive and loss-sensitive real-time audio video data transmission (AV traffic). It specifies the priority regeneration and controlled bandwidth queue draining algorithms that are used in bridges and AV traffic sources.
- IEEE 802.1Qat-2009: Allows reservation of the network resources for specific traffic streams traversing a bridged local area network.
- IEEE 802.1AS-2011: Specifies the protocol and procedures used to ensure that the synchronization requirements are met for time-sensitive applications. For example, audio and video and across bridged and virtual-bridged LANs. Virtual-bridged LANs consist of LAN media where the transmission delays are fixed and

symmetrical. For example, IEEE 802.3 full-duplex links include the maintenance of synchronized time during normal operation followed by addition, removal, or failure of network components and network reconfiguration.

As shown in the ***Transmit and Receive Path Block Diagram*** figure, one SCB master interface connects to three DMA channels (channel 0, channel 1, and channel 2). The DMA arbiter helps in arbitration of all the paths (transmit and receive) in channel 0, channel 1 and channel 2. Each channel has a separate control and status register (CSR) for managing the transmit and receive functions, descriptor handling, and interrupt handling.

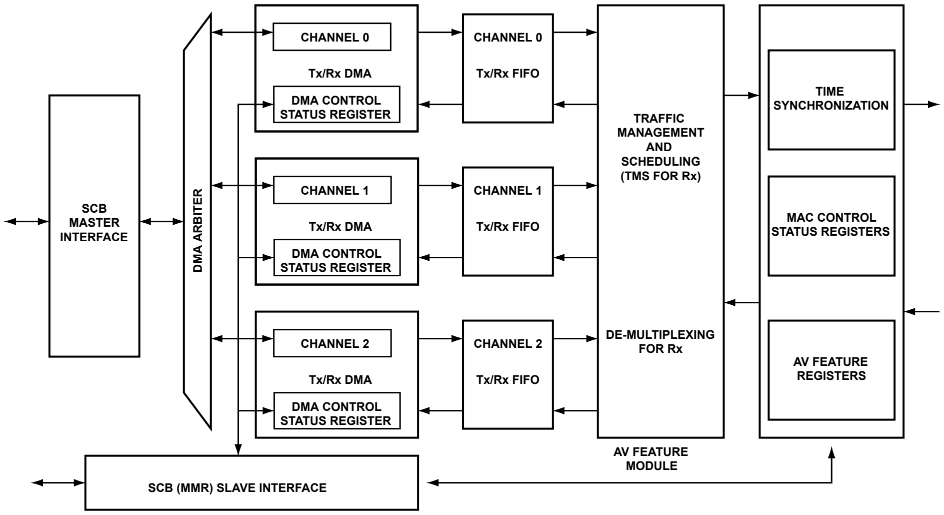

SCB MASTER INTERFACE
DMA ARBITER
CHANNEL 0
Tx/Rx DMA
DMA CONTROL STATUS REGISTER
CHANNEL 1
Tx/Rx DMA
DMA CONTROL STATUS REGISTER
CHANNEL 2
Tx/Rx DMA
DMA CONTROL STATUS REGISTER
CHANNEL 0
Tx/Rx FIFO
CHANNEL 1
Tx/Rx FIFO
CHANNEL 2
Tx/Rx FIFO
TRAFFIC MANAGEMENT AND SCHEDULING (TMS FOR Rx)
DE-MULTIPLEXING FOR Rx
TIME SYNCHRONIZATION
MAC CONTROL STATUS REGISTERS
AV FEATURE REGISTERS
AV FEATURE MODULE
SCB (MMR) SLAVE INTERFACE

Figure 31-19: Transmit and Receive Path Block Diagram

Transmit Path Functions

The transmit path of channel 0 supports strict-priority algorithm and is used for best-effort traffic. For a channel, the strict-priority algorithm determines that a frame is available for transmission if the channel contains one or more frames. When the threshold mode for EMAC MFL Tx FIFO is enabled, the strict-priority algorithm determines that a frame is available for transmission. The algorithm determines when the channel contains a partial frame of size equal to the programmed threshold limit.

The transmit paths of channel 1 and channel 2 support traffic management by using the credit-based shaper algorithm. For a channel, the credit-based shaper algorithm determines that a frame is available for transmission if the following conditions are true:

- The channel contains one or more frames.
- The credit for the channel is positive as per the algorithm.

Programs can disable the credit-based shaper algorithm for all channels or for lower-priority channels. The credit-based shaper algorithm can be disabled for channel 1 and channel 2 or for channel 1 only. When the credit-based shaper algorithm for a channel is disabled, the channel uses the default strict-priority algorithm.

Each transmit DMA has a separate descriptor chain for fetching the transmit data. The transmit channel that receives access to the system bus depends on the DMA arbiter. For information about the DMA arbiter, see DMA Arbiter.

The transmit path has separate FIFOs for each channel, as shown in the ***Transmit and Receive Path Block Diagram*** figure. The data fetched by the DMA is placed in the respective FIFO. The traffic management and scheduler unit (TMS) controls which FIFO data the MAC transmits. If the credit-based shaper algorithm is enabled for a channel (1 or 2), then the corresponding channel is selected for transmission when the following conditions are true:

- The frame is available in the channel and has a positive or zero credit.
- The higher priority channel has no frame waiting in the FIFO.

If the credit-based shaper algorithm is disabled for all channels, then the frame awaiting transmission from a channel is selected. The selection is made based on the priority scheme described in the ***Fixed Priority Scheme for DMA Channels*** table.

Receive Path Functions

To differentiate between the AV and non-AV traffic, the MAC provides a status. The status indicates if data is an AV packet and identifies its corresponding VLAN Priority tag value. This status is updated in the extended status field of the receive descriptor as explained in "Receive Descriptor". The `EMAC_MAC_AVCTL.AVT`bit field specifies a value that is compared with the EtherType field of the incoming Ethernet frame to detect an AV packet. The AV packets can be of the following two types:

- ***AV data packets.*** The AV data packets are always tagged. The tagged AV control packets are received based on the programmed priority value. The `EMAC_MAC_AVCTL.AVP` bit field specifies the channel to which an AV packet with a given priority must be sent.
- ***AV control packets.*** The AV control packets are either tagged or untagged. The untagged AV control packets are received on channel 0 by default. To receive these packets on channel 1 or channel 2, program the `EMAC_MAC_AVCTL.AVCH` bit field. Similar to the AV data packets, the tagged AV control packets are received based on the programmed priority value.

The following describes how tagged AV control packets and AV data packets are sent to a channel.

Table 31-48: AV Packets

Receive paths of channel 1 and channel 2 are enabled.	*Channel 2* The following packets, with priority value greater than or equal to the value programmed in `EMAC_MAC_AVCTL.AVP` bit field, are received on channel 2: AV packets Non-AV tagged packets (if `EMAC_MAC_AVCTL.VQE` bit is set)
	Channel 1 The following packets, with priority value less than the value programmed in AV MAC control register, are received on channel 1: AV packets Non-AV tagged packets (if `EMAC_MAC_AVCTL.VQE` bit is set)
	Channel 0 All other packets are received on channel 0.
For example, priority value of 3 is programmed in the `EMAC_MAC_AVCTL.AVP` bit field.	*Channel 2* The following packets with priority value from 3 to 7 are received on channel 2: AV packets Non-AV tagged packets (if the `EMAC_MAC_AVCTL.VQE` bit is set)
	Channel 1 The following packets with priority value 2 are received on channel 1: AV packets Non-AV tagged packets (if the `EMAC_MAC_AVCTL.VQE` bit is set)
	Channel 0 All other packets are received on channel 0.

DMA Arbiter

The DMA arbitrates between the transmit and receive paths of DMA channel 0, channel 1, and channel 2 for accessing descriptors and data buffers.

The fixed priority scheme is the default priority scheme for the DMA channels. In fixed priority scheme, the highest priority channel (channel 2) always wins the arbitration whenever it requests the bus. The ***Fixed Priority Scheme for DMA Channels*** table provides information about the priority levels of the DMA channels.

Table 31-49: Fixed Priority Scheme for DMA Channels

Priority Level	Channel
0 (low)	Channel 0
1	Channel 1
2 (high)	Channel 2

Slot Number Function

The slot number function schedules the data fetching by DMA from the system memory. This feature is useful when the source AV data must transmit at specific intervals. The transmit descriptor word 0 (TDES0) [6:3] bits program the slot number at which the DMA fetches the data from system memory. This 4-bit field allows the host to schedule data up to 16 slots of 125 micro-second each. This field is applicable only for the AV channels (channel 1 and channel 2).

When DMA fetches a transmit descriptor, it compares the slot number of the transmit descriptor with the internally generated reference slot interval. The slot interval is a counter that updates every 125 usec of the IEEE 1588 system time. In addition, the slot interval counter is initialized to zero when the value in the `EMAC_TM_SEC` register increments, that is, `EMAC_TM_NSEC` rolls over. The DMA fetches the data only if it matches the current slot or the next slot. The DMA remains in the descriptor fetch state until there is a match.

Programs can also set the `EMAC_DMA1_CHSFCS.ASC` bit to enable the DMA to fetch the data only if it matches the current slot or the next two slots.

NOTE: If the slot number in the descriptor is less than the reference slot number, the DMA takes it as a future slot.

Programs can enable the check for slot number by setting the `EMAC_DMA1_CHSFCS.ESC` bit. When this check is not enabled, the packets are fetched immediately after the descriptor is read. Programs can read the `EMAC_DMA1_CHSFCS.RSN` bit field to discover the value of the reference slot number in DMA.

Interrupts

The interrupts from different DMA channels are combined. Therefore, the software must read the interrupt status registers of all DMA channels to get the source of an interrupt. The MAC interrupt status (`EMAC_IMSK`) updates only in the interrupt status register of channel 0.

Credit-Based Shaper Algorithm Functions

The Traffic Manager and Scheduler (TMS) block (shown in the ***Transmit and Receive Path Block Diagram*** figure) uses the credit-based shaper algorithm to arbitrate the AV traffic in all channels and the legacy Ethernet traffic in channel 0. Channel 1 and channel 2 can be programmed to use the credit-based shaper algorithm.The following sections provide information about implementing the Credit-Based Shaper Algorithm:

Credit Value

The credit value is accumulated every transmit clock cycle, that is, 40 ns for 100 Mbps or 8 ns for 1000 Mbps. The credit to be added or subtracted per cycle can be fractional, based on the required idleSlope and sendSlope values as described in the following table.

Table 31-50: idleSlope and sendSlope Values

Mode	Values	Description
100 Mbps	portTransmitRate = 100 Mbps idleSlope = 70 Mbps (assuming 70% bandwidth reserved for a higher priority traffic class) sendSlope = 30 Mbps	Credit = 2.8 bits accumulates per cycle (40 ns) for the higher priority traffic class when besteffort frame is being transmitted. Credit = 1.2 bits drains per cycle (40 ns) when higher priority traffic class frame is being transmitted.

The DMA stores the channel traffic in the respective Tx FIFO based on the slot number in the transmit descriptor (if enabled) or depending on the bandwidth availability on the SCB.

The credit for a channel builds up only when the frame is available but it cannot be transmitted because the MAC is sending a frame from another channel. The EMAC supports another mode in which the credit can build up in advance for a channel in which no frame is available in its FIFO. This enables sending a burst of high priority traffic in a channel as soon as data is available. This can be enabled with the `EMAC_DMA1_CHCBSCTL.CC`/ `EMAC_DMA2_CHCBSCTL.CC` bit of the channel 1 and channel 2 CBS control registers.

When the `EMAC_DMA1_CHCBSCTL.CC`/`EMAC_DMA2_CHCBSCTL.CC` bit is reset, the accumulated credit parameter in the Credit-Based Shaper Algorithm is set to zero if there is positive credit and there is no frame to transmit in a channel. The credit does not accumulate when there is no frame waiting in a channel and other channels are transmitting. When the `EMAC_DMA1_CHCBSCTL.CC`/`EMAC_DMA2_CHCBSCTL.CC` bit is set, the accumulated credit parameter in the Credit-Based Shaper Algorithm is not reset to zero if there is positive credit and no frame to transmit in a channel. The credit accumulates even when there is no frame waiting in a channel and other channels are transmitting.

idleSlopeCredit and sendSlopeCredit Values

The software must program the idleSlopeCredit and sendSlopeCredit values. The programmed values should be the credit accumulated or drained per clock cycle scaled by 1024, such as, 2.8x1024=2867 and 1.2x1024=1229 respectively. In addition, the software must program the hiCredit and loCredit values, scaled by 1024, to adjust for scaling of the idleSlopeCredit and sendSlopeCredit values.

This means that if computed hiCredit and loCredit values are 12000 bits and 3036 bits respectively, then the values to be programmed in the `EMAC_DMA1_CHHIC`/`EMAC_DMA2_CHHIC` and `EMAC_DMA1_CHLOC`/ `EMAC_DMA2_CHLOC` registers are 12000x1024 bits and two's complement of 3036x1024 respectively.

Bandwidth Status

The hardware maintains the status of the actual bandwidth consumed by each higher priority channel (channel 1 and channel 2) in the CBS status registers (`EMAC_DMA1_CHCBSSTAT`/`EMAC_DMA2_CHCBSSTAT`). This allows the software to estimate the average bandwidth consumed by numerically higher traffic classes as compared to the reserved bandwidth.

The CBS status register gives the average number of bits transmitted during the previous programmed slot interval (1, 2, 4, 8, or 16 slots of 125 us) in a channel. The status register is updated even if the Credit-Based Shaper Algorithm is not enabled for a channel. The number of slots over which the average bits transmitted per slot are computed is programmed in the `EMAC_DMA1_CHCBSCTL.SLC`/`EMAC_DMA2_CHCBSCTL.SLC` bits. For example, if these bit fields are programmed for two slots, then the average bits are computed over slot numbers 0-1, 2-3, 4-5, and so on.

The value programmed in the `EMAC_DMA1_CHISC`/`EMAC_DMA2_CHISC` register of a channel is proportional to the bandwidth reserved for the channel. The software can allocate any bandwidth that is not used by the higher priority channel to the reserved bandwidth of the lower priority channel.

A lower priority channel, using the Credit-Based Shaper Algorithm, cannot use the unused reserved bandwidth of any higher priority channel that is using the Credit-Based Shaper Algorithm. However, a lower priority channel that is using the strict-priority algorithm can use the unused reserved bandwidth of any higher priority channel that uses the Credit-Based Shaper Algorithm. For example, channel 1 and channel 2 use the Credit-Based Shaper Algorithm (with reserved bandwidth of 50% and 25% respectively) and channel 0 uses the strict-priority algorithm. If channel 1 uses only 40% of the reserved bandwidth, then the remaining 10% is used by channel 0. The channel 2 cannot exceed the reserved bandwidth of 25%.

Energy-Efficient Ethernet

Energy-Efficient Ethernet (EEE) is an optional operational mode that enables the IEEE 802.3 Media Access Control (MAC) sublayer along with a family of Physical layers to operate in the Low-Power Idle (LPI) mode. The EEE operational mode supports the IEEE 802.3 MAC operation at 100 Mbps, 1000 Mbps, and 10 Gbps. The MAC supports the IEEE Standard 802.3az-2010 for EEE.

The LPI mode allows power saving by switching off parts of the communication device functionality when there is no data awaiting transmission and receipt. The systems on both sides of the link can disable some functionality and save power during the periods of low-link utilization. The MAC controls whether the system should enter or exit the LPI mode and communicates this to the PHY.

The EEE specifies the capabilities negotiation methods that the link partners can use to determine whether EEE is supported and then select the set of parameters that common to both devices.

NOTE: The EEE feature is not supported when the MAC is configured to use the RMII interface. You should activate the EEE mode only when the MAC is operating with RGMII interface.

NOTE: According to the Energy-Efficient Ethernet standard (IEEE 802.3az-2010), the LPI mode is supported only in the full-duplex mode. Therefore, you should not enable the LPI mode when the MAC Transmitter is configured for the half-duplex mode.

Transmit Path Functions

In the transmit path, the software must set the `EMAC_LPI_CTLSTAT.LPIEN` bit to indicate to the MAC to stop transmission and initiate the LPI protocol. The MAC completes the transmission in progress, generates its transmission status, and then starts transmitting the LPI pattern instead of the IDLE pattern during Interframe gap (IFG).

To make the PHY enter the LPI state, the MAC performs the following tasks:

1. De-asserts the `ETH0_TXEN` signal.
2. Asserts the TX_ER signal.
3. Sets `EMAC_TXD[n]`[3:0] to 0x1 (for 100 Mbps) or `EMAC_TXD[n]`[7:0] to 0x01 (for 1000 Mbps).

   NOTE: The MAC maintains the same state of the TX_EN, TX_ER, and `EMAC_TXD[n]` signals for the entire duration during which the PHY remains in the LPI state.

4. Updates the status using the `EMAC_LPI_CTLSTAT.TLPIEN` bit and generates an interrupt.

To bring the PHY out of the LPI state, that is, when the software resets the `EMAC_LPI_CTLSTAT.LPIEN` bit, the MAC performs the following tasks:

1. Stops transmitting the LPI pattern and starts transmitting the IDLE pattern.
2. Starts the LPI TW TIMER: The MAC cannot start the transmission until the wake-up time specified for the PHY expires. The auto-negotiated wake-up interval is programmed using the `EMAC_LPI_TMRSCTL.TWT` bit field.
3. Updates the LPI exit status using the `EMAC_LPI_CTLSTAT.TLPIEX` bit and generates an interrupt.

The ***LPI Transitions (Transmit)*** figure shows the behavior of TX_EN, TX_ER, and `EMAC_TXD[n]` [3:0] signals during the LPI mode transitions.

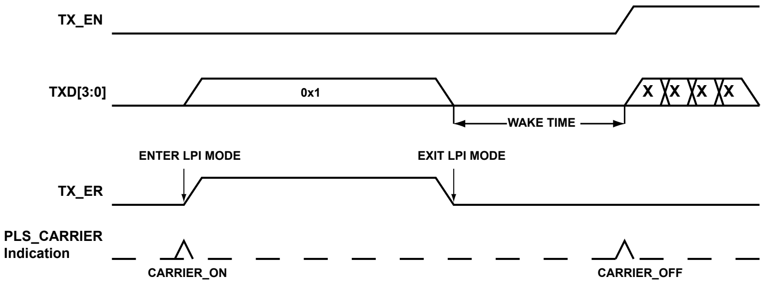


Figure 31-20: LPI Transitions (Transmit)

NOTE: The MAC does not stop the TX_CLK clock.

NOTE: If the MAC is in the Tx LPI mode and the Tx clock is stopped, the application should not write to CSR registers that are synchronized to Tx clock domain.

NOTE: If the MAC is in the LPI mode and the host issues a soft reset or hard reset, the MAC transmitter comes out of the LPI mode.

Receive Path Functions

In the receive path, when the PHY receives the signals from the link partner to enter into the LPI state, the PHY and MAC perform the following tasks:

1. The PHY asserts the `EMAC_RXERR`.
2. The PHY sets `EMAC_RXD[n]`[7:0] to 0x01.
3. The PHY de-asserts RX_DV.

   NOTE: The PHY maintains the same state of the `EMAC_RXERR`, `EMAC_RXD[n]`, and RX_DV signals for the entire duration during which it remains in the LPI state.

4. The MAC updates the `EMAC_LPI_CTLSTAT.RLPIEN` bit and generates an interrupt immediately.

NOTE: If the LPI pattern is detected for a very short duration (that is, less than 2 cycles of Rx clock), the MAC does not enter the Rx LPI mode.

NOTE: If the duration between end of the current Rx LPI pattern and start of the next Rx LPI pattern, is very short (that is, less than 2 cycles of Rx clock), then the MAC exits and again enters the Rx LPI mode. The MAC does not give the Rx LPI Exit and Entry interrupts.

When the PHY receives signals from the link partner to exit the LPI state, the PHY and MAC perform the following tasks:

1. The PHY de-asserts `EMAC_RXERR` and returns to a normal inter-frame state.
2. The MAC updates the `EMAC_LPI_CTLSTAT.RLPIEX` bit and generates an interrupt immediately.

The ***LPI Transitions (Receive)*** figure shows the behavior of `EMAC_RXERR`, RX_DV, and `EMAC_RXD[n]`[3:0] signals during the LPI mode transitions.

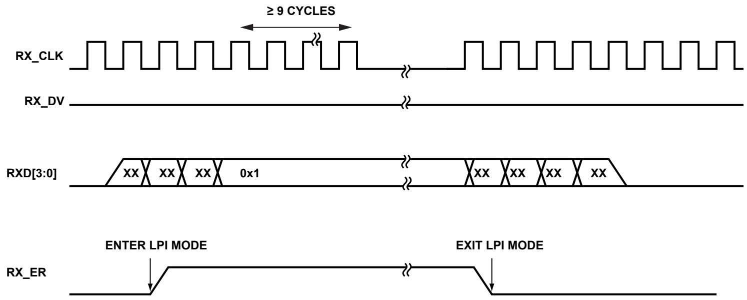


Figure 31-21: LPI Transitions (Receive)

NOTE: If the `RX_CLK_STOPPABLE` bit (in the PHY register written through MDIO) is asserted when the PHY is indicating LPI to the MAC, then the PHY may halt the RX_CLK at any time more than 9 clock cycles after the start of the LPI state as shown in the ***LPI Transitions (Receive)*** figure.

NOTE: If the MAC is in the LPI mode and the host issues a soft reset or hard reset, the MAC receiver comes out of the LPI mode during reset. If the LPI pattern is still received after the reset is de-asserted, the MAC receiver again enters the LPI state.

NOTE: If the RX clock is stopped in the RX LPI mode, the host should not write to the CSR registers that are being synchronized to the RX clock domain.

NOTE: When the PHY sends the LPI pattern, if EEE feature is enabled, the MAC automatically enters the LPI state. There is no software control to prevent the MAC from entering the LPI state.

LPI Timers

LPI LS TIMER

The LPI LS timer counts, in milliseconds, the time expired since the link status is up. Programs can enable the monitoring of Bit 3 `EMAC_ISTAT.RGMIIIS` (link status) of Register 54 (SGMII/RGMII/SMII Control and Status Register) by setting the `EMAC_LPI_CTLSTAT.PLSEN` bit.

The link status is indicated to the MAC by Bit 3 (link status) of Register 54 (RGMII Control and Status Register) or the value programmed by the software in the `EMAC_LPI_CTLSTAT.PLS` bit. If the link status is not available in Register 54 (RGMII Control and Status Register), the software should get the PHY link status by reading the PHY register and accordingly update the `EMAC_LPI_CTLSTAT.PLS` bit.

This timer is cleared every time the link goes down. It starts to increment when the link is up again and continues to increment until the value of the timer becomes equal to the terminal count. When the terminal count is reached, the timer remains at the same value as long as the link is up. The terminal count is the value programmed in the `EMAC_LPI_TMRSCTL.LST` bit field. The GMII interface does not assert the LPI pattern unless the terminal count is reached. This ensures a minimum time for which no LPI pattern is asserted after a link is established with the remote station. This period is defined as 1 second in the IEEE standard 802.3-az, version D2.0. The LPI LS timer is 10-bit wide. Therefore, the software can program up to 1023 milliseconds

LPI TW TIMER

The LPI TW timer counts, in microseconds, the time expired since the de-assertion of LPI. The terminal should be programmed using the `EMAC_LPI_TMRSCTL.TWT` bit. The terminal count of the timer is the value of resolved Transmit TW that is the auto-negotiated time after which the MAC can resume the normal transmit operation. After exiting the LPI mode, the MAC resumes its normal operation after the TW timer reaches the terminal count.

The MAC supports the LPI TW timer in units of microsecond. The LPI TW timer is 16-bit wide. Therefore, the software can program up to 65535 us.

NOTE: Program the `EMAC_LPI_CTLSTAT.PLS` bit to 1'b0 before switching between the GMII and MII modes. This resets the internal timers. If the mode is changed after the LPI LS timer or LPI TW timer starts, the change in the TX clock frequency can result in incorrect timeout.

LPI Interrupt

The MAC generates the LPI interrupt when the Tx or Rx side enters or exits the LPI state. The interrupt is asserted when the LPI interrupt status is set. The LPI interrupt can be cleared by reading the `EMAC_LPI_CTLSTAT` register.

EMAC Event Control

The EMAC has a dedicated interrupt signal registered with the system event controller (SEC) module. Various interrupt sources within the EMAC peripheral are shared through this interrupt line. Refer to the System Event

Controller (SEC) and Generic Interrupt Controller (GIC) chapter for details on how interrupts work in this product and how to configure them.

EMAC Interrupt Signals

Interrupts from the EMAC are triggered from the EMAC DMA layer or the EMAC CORE layer. Interrupts are triggered from EMAC DMA when a particular status bit is set in the `EMAC_DMA0_STAT` register. An interrupt line is asserted only when the corresponding bits are enabled in the DMA interrupt enable register. Similarly, interrupts are triggered from the EMAC CORE when a particular MMC status bit, RGMII Link status bit, or PTP status bit is set in the interrupt status register.

An interrupt line is asserted only when the corresponding bits are enabled in the MMC mask registers for MMC counters or the interrupt mask register for PTP. DMA status register also reflects the MMC interrupt status. The following lists show the two groups of interrupts in the DMA status register.

NIS – Normal Interrupt source summary:

- Transmit Interrupt
- Transmit Buffer Unavailable
- Receive Interrupt
- Early Receive Interrupt

AIS – Abnormal Interrupt source summary:

- Transmit Process Stopped
- Transmit Jabber Timeout
- Receive FIFO Overflow
- Transmit Underflow
- Receive Buffer Unavailable
- Receive Process Stopped
- Receive Watchdog Timeout
- Early Transmit Interrupt
- Fatal Bus Error

The EMAC generates an interrupt only once for simultaneous, multiple events. The driver must read the `EMAC_DMA0_STAT` register for the cause of the interrupt. It can generate a new interrupt once the driver has cleared the appropriate bit in DMA status register.

For example, the controller generates a receive interrupt (`EMAC_DMA0_STAT.RI` bit) and the driver begins reading the `EMAC_DMA0_STAT` register. Next, a receive buffer unavailable interrupt (`EMAC_DMA0_STAT.RU` bit) occurs. The driver clears the `EMAC_DMA0_STAT.RI` bit but the internal interrupt signal is not deasserted, because of the active or pending `EMAC_DMA0_STAT.RU` interrupt. The driver must scan all of the descriptors, from the last recorded position to the first one owned by the DMA, to know which descriptor has asserted the interrupt.

Interrupts are cleared by writing a 1 to the corresponding bit position in the `EMAC_DMA0_STAT` register. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared.

An interrupt delay timer provides (receive interrupt watchdog timer register) flexible control of the receive interrupt.

When the interrupt timer is programmed with a non-zero value, it is activated as soon as the RxDMA transfers a received frame to system memory. The transfer occurs without asserting the receive interrupt. This interrupt is not enabled in the corresponding receive descriptor (`RDES1`[31] in the receive DMA descriptors).

When this timer runs out (per the programmed value), the `EMAC_DMA0_STAT.RI` bit is set. The interrupt is asserted when the corresponding `EMAC_DMA0_STAT.RI` bit is enabled in the interrupt enable register. The timer is disabled before it runs out, when a frame is transferred to memory and when the `EMAC_DMA0_STAT.RI` bit is set because it is enabled for that descriptor.

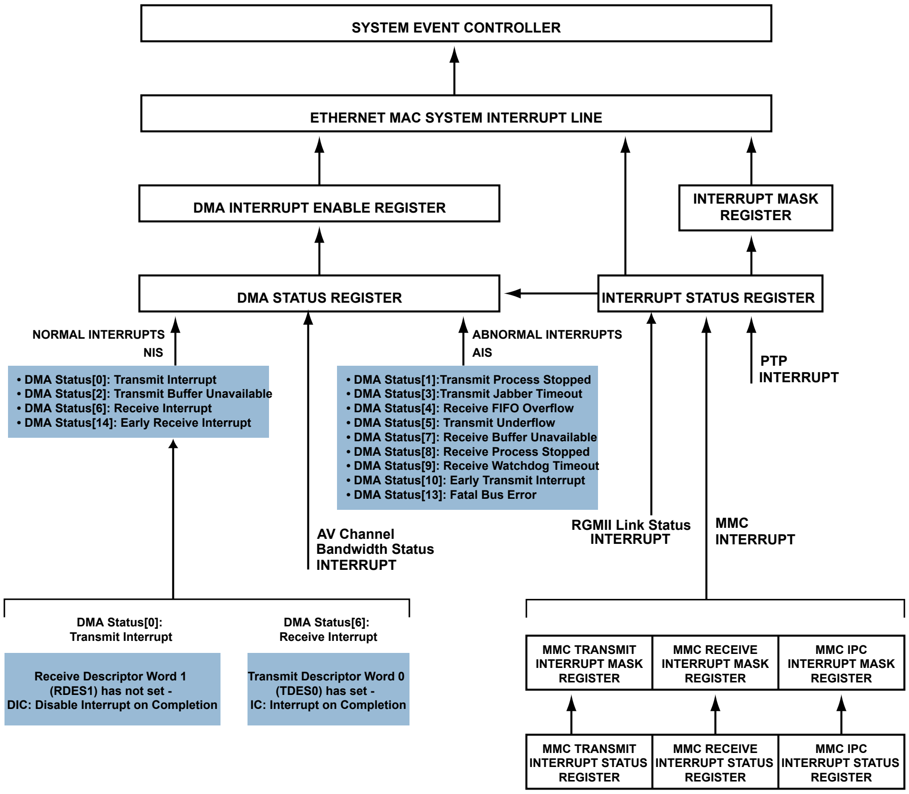

SYSTEM EVENT CONTROLLER
ETHERNET MAC SYSTEM INTERRUPT LINE
DMA INTERRUPT ENABLE REGISTER
INTERRUPT MASK REGISTER
DMA STATUS REGISTER
INTERRUPT STATUS REGISTER
NORMAL INTERRUPTS NIS
ABNORMAL INTERRUPTS AIS
PTP INTERRUPT
• DMA Status[0]: Transmit Interrupt
• DMA Status[2]: Transmit Buffer Unavailable
• DMA Status[6]: Receive Interrupt
• DMA Status[14]: Early Receive Interrupt
• DMA Status[1]:Transmit Process Stopped
• DMA Status[3]:Transmit Jabber Timeout
• DMA Status[4]: Receive FIFO Overflow
• DMA Status[5]: Transmit Underflow
• DMA Status[7]: Receive Buffer Unavailable
• DMA Status[8]: Receive Process Stopped
• DMA Status[9]: Receive Watchdog Timeout
• DMA Status[10]: Early Transmit Interrupt
• DMA Status[13]: Fatal Bus Error
AV Channel Bandwidth Status INTERRUPT
RGMII Link Status INTERRUPT
MMC INTERRUPT
DMA Status[0]: Transmit Interrupt
DMA Status[6]: Receive Interrupt
Receive Descriptor Word 1 (RDES1) has not set - DIC: Disable Interrupt on Completion
Transmit Descriptor Word 0 (TDES0) has set - IC: Interrupt on Completion
MMC TRANSMIT INTERRUPT MASK REGISTER
MMC RECEIVE INTERRUPT MASK REGISTER
MMC IPC INTERRUPT MASK REGISTER
MMC TRANSMIT INTERRUPT STATUS REGISTER
MMC RECEIVE INTERRUPT STATUS REGISTER
MMC IPC INTERRUPT STATUS REGISTER

Figure 31-22: EMAC Interrupt Flow Diagram

PHYINT Interrupt Signal

A PHY device can notify the EMAC when it detects changes to the link status, such as auto-negotiation or a duplex-mode change. The external PHY chip typically includes an interrupt generation pin to aid this status change notification to the MAC. This signal is typically called PHYINT. A failing or rising edge on this signal can detect a PHY interrupt at the EMAC.

In the processor, any of the GPIO pins can be a used as a PHYINT signal. Use the following procedure to configure a GPIO as a PHYINT signal.

1. Program the GPIO to detect a falling or rising edge sensitive interrupt.
2. Program the PHY to generate the interrupt on a signal status change.
3. If PHYINT is asserted, read the PHY status register through the station management interface.

NOTE: The PHYINT is not part of EMAC module. However, any GPIO pin can be configured to interrupt the processor when a rising edge generated by PHY is detected.

Refer to the PORT chapter for more info on configuring GPIO pins for input.

EMAC Programming Model

This section provides the programming model of Ethernet MAC peripheral for developers.

EMAC Programming Steps

The following sections provide some general programming information. The steps are written for EMAC DMA0, but apply to DMA1 and DMA2 as well.

DMA Initialization

Use the following procedure to initialize DMA. This procedure applies to all DMA channels.

1. Perform a software reset by setting the `EMAC_DMA0_BUSMODE.SWR` bit. This action resets all of the EMAC internal registers and logic.
2. Wait for the completion of the reset process by polling the `EMAC_DMA0_BUSMODE.SWR` bit which is only cleared (automatically) after the reset operation completes.
3. Poll the `EMAC_DMA0_BMSTAT.BUSRD` and `EMAC_DMA0_BMSTAT.BUSWR` bits to confirm that all previously initiated (before software-reset) or ongoing SCB transactions are complete.
4. Program the required fields in the `EMAC_DMA0_BMMODE` register:
    a. Address aligned bursts
    b. Fixed burst or undefined burst
    c. Burst length values and burst mode values
    d. Descriptor length (only valid when using ring mode)
5. Program the SCB interface options in the `EMAC_DMA0_BMMODE` register. If fixed burst-length is enabled, then select the maximum burst-length possible on the SCB bus (bits `EMAC_DMA0_BMMODE.BLEN4`, `EMAC_DMA0_BMMODE.BLEN8`, `EMAC_DMA0_BMMODE.BLEN16`).

6. Create a proper descriptor chain for transmit and receive. In addition, ensure that the DMA owns the receive descriptors (the `OWN` bit of the descriptor is set). For OSF mode, create at least two descriptors.
7. Ensure that the software creates three or more different transmit or receive descriptors in the chain before reusing any of the descriptors.
8. Initialize the `EMAC_DMA0_RXDSC_CUR` and `EMAC_DMA0_TXDSC_CUR` registers with the base address of the receive and transmit descriptors respectively.
9. Program the required fields in the `EMAC_DMA0_OPMODE` register to initialize the mode of operation as follows:
   a. Receive and transmit store and forward
   b. Receive and transmit threshold control
   c. Error Frame and undersized good frame forwarding enable
   d. OSF mode
10. Clear the interrupt requests by writing to those bits of the `EMAC_DMA0_STAT` register (interrupt bits only) that are set. For example, by writing 1 into bit 16, the normal interrupt summary clears this bit.
11. Enable the interrupts by programming the `EMAC_DMA0_IEN` register.
12. Start the receive and transmit DMA by setting the `EMAC_DMA0_OPMODE.SR` and `EMAC_DMA0_OPMODE.ST` bits.

EMAC CORE Initialization

Use the following procedure to initialize the EMAC CORE.

1. Program the EMAC management address register (`EMAC_SMI_ADDR`) for controlling the management cycles for external PHY. For example, physical layer address (`EMAC_SMI_ADDR.PA`). In addition, set the `EMAC_SMI_ADDR.SMIB` bit for writing into PHY and reading from PHY.
2. Read the 16-bit data of management address register (`EMAC_SMI_DATA`) from the PHY for link up, speed of operation, and mode of operation. Specify the appropriate address value in the `EMAC_SMI_ADDR.PA` bit field.
3. Program the MAC address in the `EMAC_ADDR0_HI` and `EMAC_ADDR0_LO` registers.
4. If using hash filtering, program the hash table high and low registers register (`EMAC_HASHTBL_HI`, `EMAC_HASHTBL_LO`).
5. Program the fields to set the appropriate filters for the incoming frames in the MAC frame filter register (`EMAC_MACFRMFILT`):
   a. Receive all.
   b. Promiscuous mode.

   c. Hash or perfect filter.
   d. Unicast, multicast, broadcast, and control frames filter settings.
6. Program the fields for proper flow control in flow control register (EMAC_FLOWCTL):
   a. Pause time and other pause frame control bits.
   b. Receive and transmit flow control bits.
   c. Flow control busy or backpressure activate.
7. Program the EMAC interrupt mask register bits (EMAC_IMSK), as needed.
8. Program the appropriate fields in MAC configuration register (EMAC_MACCFG). For example, inter-frame gap while transmission and jabber disable. Based on the auto-negotiation desired, set the duplex mode (EMAC_MACCFG.DM bit) or speed select (EMAC_MACCFG.FES bit).
9. Set the transmit enable (EMAC_MACCFG.TE) and receive enable (EMAC_MACCFG.RE) bits.

Performing Normal Transmit and Receive Operations

For normal transmit and receive interrupts, the program must first read the interrupt status.

1. Poll the descriptors, reading the status of the descriptor owned by the application (either transmit or receive).
2. Set the appropriate values for the descriptors, ensuring that transmit and receive descriptors are DMA descriptors to resume the transmission and reception of data.

   ADDITIONAL INFORMATION: If the descriptors are not DMA (or no descriptor is available), the DMA goes into SUSPEND state.
3. Write a 0 into the Tx/Rx poll demand registers (EMAC_DMA0_TXPOLL and EMAC_DMA0_RXPOLL).

   Transmit or receive operations are resumed by freeing the descriptors and issuing a poll demand.
4. Read (for the debug process), the values of the current host transmitter or receiver descriptor address pointer (EMAC_DMA0_TXDSC_CUR, EMAC_DMA0_RXDSC_CUR) registers.
5. Read (for the debug process), the values of the current host transmit buffer address pointer and receive buffer address pointer (EMAC_DMA0_TXBUF_CUR, EMAC_DMA0_TXBUF_CUR) registers.

Stopping and Starting Transfers

Use the following procedure to stop and start EMAC transfers.

1. Disable the transmit DMA (if applicable), by clearing the EMAC_DMA0_OPMODE.ST bit.
2. Wait for any previous frame transmissions to complete. Check for completion by reading the appropriate bits of the debug register (EMAC_DBG).

3. Disable the MAC transmitter and MAC receiver by clearing the `EMAC_MACCFG.TE` and `EMAC_MACCFG.RE` bits.
4. Disable the receive DMA (if applicable), after ensuring that the data in the receive FIFO transfers to the system memory by reading the `EMAC_DBG` register.
5. Make sure that both the transmit and receive FIFOs are empty.
6. To restart the operation, first start the DMA, and then enable the MAC transmitter and receiver.

Interrupts and Interrupt Service Routines

The following procedure describes specific steps for enabling interrupts and using their ISRs.

This procedure is typically performed with EMAC and DMA initialization and operations.

1. Receive interrupts are enabled for descriptors by default. Enable transmit interrupts for individual descriptors by setting the IC bit (bit 30) in the TDES0 word of the transmit descriptor.
2. Enable the required bits in the DMA interrupt enable register (`EMAC_DMA0_IEN.NIE`).

   ADDITIONAL INFORMATION: Setting the `EMAC_DMA0_IEN.NIE` or `EMAC_DMA0_IEN.AIE` bits can turn on the occurrence of all normal or abnormal interrupt conditions. Individual conditions can also be enabled on using individual bits in the `EMAC_DMA0_IEN` register.
3. Enable MMC overflow interrupts by setting appropriate bits in the `EMAC_MMC_RXIMSK` and `EMAC_MMC_TXIMSK` registers.
4. Enable PTP interrupts by setting the `EMAC_IMSK.TS` bit.
5. Once an EMAC interrupt is asserted and the SEC branches execution to the EMAC ISR, perform the following software program sequence.
   a. Read DMA status from the `EMAC_DMA0_STAT` register.
   b. Clear the interrupt source by writing 1 (W1C) to the bits that are set in the `EMAC_DMA0_STAT` register.
   c. Check for normal/abnormal/mmc/ptp interrupts by parsing the status bits read earlier, and call the appropriate service function.

      ADDITIONAL INFORMATION: Normal interrupt assertions include the transmit and receive interrupt. Abnormal interrupt assertions include the receive underflow.
6. The MMC handler functions use the following sequence.
   a. Read the `EMAC_ISTAT` register and parse for the `EMAC_ISTAT.MMCTX` and `EMAC_ISTAT.MMCRX` bits to determine if the interrupt is a transmit counter or receive counter-interrupt.
   b. Read the `EMAC_MMC_RXINT` or `EMAC_MMC_TXINT` registers to determine which of the counters have triggered the interrupt.
   c. Read the respective MMC counter that caused the interrupt to clear it.

7. PTP handler functions use the following sequence:
    a. Read the `EMAC_ISTAT.TS` bit to determine if a PTP interrupt occurred.
    b. Read `EMAC_TM_STMPSTAT` register to determine the interrupt source by parsing the `EMAC_TM_STMPSTAT.ATSTS`, `EMAC_TM_STMPSTAT.TSTARGT0` - `EMAC_TM_STMPSTAT.TSTARGT3`, and `EMAC_TM_STMPSTAT.TSSOVF` bits.
    c. Clear the interrupt source by reading the `EMAC_TM_STMPSTAT` register.

Enabling Checksum for Transmit and Receive

Use the following steps to enable transmit and receive checksums.

Enabling receive and transmit checksums is typically performed with EMAC and DMA initialization and operations. Transmit and receive checksum features are independent of each other.

1. To enable transmit checksum insertion:
    a. Enable store-and-forward mode in the FIFO by setting the `EMAC_DMA0_OPMODE.TSF` bit.
    b. Ensure that the transmit frame can be contained within the 256 byte Tx FIFO conforming to the size rule: FIFO Depth – PBL – 3 FIFO locations, where PBL is burst length.
    c. Program the following necessary parameters for transmit checksum, by programming (CIC) checksum insertion control in TDES0: IP header checksum, IP header checksum and payload checksum, IP header checksum, payload checksum, and pseudo header checksum.

   A higher layer such as the IP stack sends out the packet to the EMAC which inserts the checksum as configured.
2. To enable receive checksum verification:
    a. Enable receive checksum offload engine by setting the `EMAC_MACCFG.IPC` bit.
    b. Enable 8 word descriptor (32 bytes), by setting the `EMAC_DMA0_BUSMODE.ATDS` bit.
    c. Provide a total of 8 x 32-bit word space for the receive descriptor.
    d. Wait for the receive interrupt and check for extended status availability by parsing bit 0 in the RDES0 word.
    e. If extended status available, read RDES4 and pass to a higher layer such as the IP stack.

       The higher software layer can check for IPv4/IPv6/payload type and checksum payload or header errors.

Programming the System Time Module

Use the following procedure to configure the PTP module

1. Enable the PTP module by setting the `EMAC_TM_CTL.TSENA` bit 0.

2. System Time Initialization
    a. The time (seconds and nanoseconds) at which system time is initialized. Write the time into the `EMAC_TM_SECUPDT` and `EMAC_TM_NSECUPDT` registers.
    b. Set `EMAC_TM_CTL.TSINIT` bit. System time is initialized and this bit is cleared automatically.
    c. Configure binary or digital rollover of the `EMAC_TM_NSEC` register using the `EMAC_TM_CTL.TSCTRLSSR` bit.
3. System Time Coarse Correction
    a. Write the offset time (seconds and nanoseconds) to add to or subtract from the system time using the `EMAC_TM_SECUPDT` and `EMAC_TM_NSECUPDT` registers.
    b. Choose between add or subtract offset time using the `EMAC_TM_NSECUPDT.ADDSUB` bit.
    c. Set the `EMAC_TM_CTL.TSUPDT` bit to correct system time with offset time. This bit is cleared automatically.
4. System Time Fine Correction
    a. Calculate the addend value based on the input PTP clock frequency and the frequency requirement. See Fine Correction Method.
    b. Write the calculated addend value in `EMAC_TM_ADDEND` register and set the `EMAC_TM_CTL.TSADDREG` bit to update the addend value. This bit is cleared automatically.
    c. Configure the `EMAC_TM_SUBSEC` register based on new PTP frequency.
5. Target Time Trigger (Alarm)
    a. Set the `EMAC_IMSK.TS` bit to enable PTP interrupts.
    b. Program the `EMAC_TM_PPSCTL.TRGTMODSEL0` bit with 00 or 10 (for PPS start or stop time programming).
    c. Program the time when the interrupt must trigger using the `EMAC_TM_PPS0TGTM` and `EMAC_TM_PPS0NTGTM` registers. The programmed time must be greater than the current system time.

       ADDITIONAL INFORMATION: If the programmed time is not greater than the target time, a programming error occurs. The EMAC uses the `EMAC_TM_STMPSTAT.TSTRGTERR0` bit to indicate the error.
    d. Set the `EMAC_TM_CTL.TSTRIG` bit to enable the target time trigger interrupt.

       After the system time reaches the programmed target time (in step 2), the target time trigger interrupt occurs. The `EMAC_TM_STMPSTAT.TSTRGTERR0` and `EMAC_ISTAT.TS` bits indicate the error. The `EMAC_TM_CTL.TSTRIG` bit is cleared automatically.

Programming the PTP for Frame Detection and Time Stamping

Use the following procedure to configure the PTP module.

1. For time stamping a transmitting frame, set the TTSE bit in the TDES0 register of the corresponding frame.
2. Extend the descriptor word length from 4 words to 8 words by setting the `EMAC_DMA0_BUSMODE.ATDS` bit.
3. Configure bits 18–10 in the `EMAC_TM_CTL` register so that the PTP module detects and time stamps only specific types of received frames. Refer to the `EMAC_TM_CTL` register description for more information.
4. Enable the PTP module by setting the `EMAC_TM_CTL.TSENA` bit.
5. Initialize the system time.
6. Verify the RDES4 register for the status of the received frame and the RDES6 and RDES7 registers for time stamp nanoseconds and seconds value.

Programming for Auxiliary Time Stamps

1. Set the `EMAC_IMSK.TS` bit to enable PTP interrupts.
2. Set the `EMAC_TM_CTL.TSENA` bit to enable the PTP module.
3. Initialize system time.

   ADDITIONAL INFORMATION: Whenever a rising edge on the auxiliary time stamp trigger pin is detected, system time seconds and nanoseconds are captured and stored into 4-deep auxiliary time stamp FIFO. An auxiliary time stamp trigger interrupt occurs. The EMAC uses the `EMAC_TM_STMPSTAT.ATSTS` and the `EMAC_IMSK.TS` bits to indicate the interrupt.
4. Read the contents of the FIFO one-by-one through `EMAC_TM_AUXSTMP_SEC` and `EMAC_TM_AUXSTMP_NSEC` registers. One level of the FIFO is cleared when the `EMAC_TM_AUXSTMP_SEC` register is read. Therefore, read the `EMAC_TM_AUXSTMP_NSEC` register before the `EMAC_TM_AUXSTMP_SEC` register.
5. Set the `EMAC_TM_CTL.ATSFC` bit to clear the FIFO.

Programming Fixed Pulse-Per-Second Output

Use the following procedure to program the fixed PPS output.

1. Enable the PTP module by setting the `EMAC_TM_CTL.TSENA` bit.
2. Output the PPS waveform by configuring the `EMAC_TM_PPSCTL.PPSCTL0` bits and binary or digital rollover using the `EMAC_TM_CTL.TSCTRLSSR` bit. See Fixed Pulse-Per-Second Output.

Programming Flexible Pulse-Per-Second Output

Use the following procedure to program flexible PPS output.

1. Enable the PTP module by setting the `EMAC_TM_CTL.TSENA` bit.
2. Set the `EMAC_TM_PPSCTL.PPSEN` bit to enable flexible PPS output.

3. Program the `EMAC_TM_PPSCTL.TRGTMODSEL0` bits with 11 or 10 (for target time trigger interrupt).
4. Program the start time value when the PPS output starts using the `EMAC_TM_PPS0TGTM` and `EMAC_TM_PPS0NTGTM` registers. Ensure that the `EMAC_TM_PPS0NTGTM.TSTRBUSY` bit is reset before programming the target time registers again.
5. Program the period of the PPS signal output using the `EMAC_TM_PPS0INTVL` register for pulse train output. Program the width of the PPS signal output in the `EMAC_TM_PPS0WIDTH` register for single pulse or pulse train output.
6. Ensure that the `EMAC_TM_PPSCTL.PPSCTL0` bits are cleared. Then, program the bits to 0001 to start single pulse, or to 0010 to start pulse train at programmed start time (Step 4).

   ADDITIONAL INFORMATION: The PPS pulse train is free-running unless stopped by a STOP pulse train at time command (`EMAC_TM_PPSCTL.PPSCTL0` = 0100) or STOP pulse train immediately command `EMAC_TM_PPSCTL.PPSCTL0` = 0101).
7. The start of pulse generation can be canceled by giving the cancel start command (`EMAC_TM_PPSCTL.PPSCTL0` = 0011) before the programmed start time (Step 4) elapses.
8. Program the stop time value when the PPS output must stop using the`EMAC_TM_PPS0TGTM` and `EMAC_TM_PPS0NTGTM` registers. Ensure that the `EMAC_TM_PPS0NTGTM.TSTRBUSY` bit is reset before programming the target time registers again.
9. Ensure that the `EMAC_TM_PPSCTL.PPSCTL0` bits are cleared. Then, program the bits to 0100. This programming stops the train of pulses on PPS signal output after the programmed stop time (Step 8) elapses.

   ADDITIONAL INFORMATION: The pulse train can be stopped immediately by giving the STOP pulse train immediately command (`EMAC_TM_PPSCTL.PPSCTL0` = 0101). Program the `EMAC_TM_PPSCTL.PPSCTL0` bits to 0110 before the programmed stop time (Step 8) elapses to cancel the stop pulse train command (given in Step 9).

EMAC Programming Concepts

The following sections provide basic information and guidelines to help with programming the EMAC module.

IEEE 802.3 Ethernet Packet Structure

The ***IEEE 802.3 Frame Structure*** table provides typical frame format of an Ethernet packet. Refer to the IEEE standards for detailed information on Ethernet packets and their format.

Table 31-51: IEEE 802.3 Frame Structure

Parameter	Description	Position in Ethernet Packet	Total Bytes
PREAMBLE	This parameter is a 56-bit (7-byte) pattern of alternating 1 and 0 bits (#10101010), which allows devices on the network to detect a new incoming frame for synchronization.	1	7
SFD	The SFD (#10101011) is a 1-byte pattern designed to break the preamble pattern, and signal the start of the actual frame.	2	1
DA	48-bit destination address. This parameter can be a unicast, multicast, or broadcast address.	3	6
SA	48-bit long source address, typically a unicast, multicast, or broadcast address.	4	6
LT	Typically this field is the length, in terms of the number of bytes, and can be anywhere between 0–1500. When the value is greater than or equal to 0x0600, this field also indicates the type of special payload carried by the frame. Examples include 0x8808 for flow control and 0x0800 for IPv4.	5	2
DATA	Actual application data payload, usually between 0–1500.	6	0–1500
PAD	This field compensates for data frames that are shorter than 64 bytes long, not including the preamble.	7	0–46
FCS	The frame check sequence is a 32-bit cyclic redundancy check that detects corrupted data within the entire frame. This parameter is generated from a CRC-32 polynomial code (CRC-32-IEEE): G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.	8	4

Frame Size Statistics for Application Software

Table 31-52: Ethernet Frame Size Statistics

Frame size statistics	VLAN-specific change Comments	
Information bytes/Header	4 byte 802.1Q header inserted after source address and before Type/LAN in 802.3 packets = 22 bytes.	6 x 2 + 2 + 4 = 18 bytes (DA+SA+LT+FCS)
Minimum Frame Size (typical)	If DATA is NULL, 42-byte padding makes 64 bytes (42 +22)	64 bytes. If DATA is NULL, 46-byte padding makes 64 bytes (46 +18)
Maximum Frame Size (typical)	1522 bytes	1518 bytes (1500 bytes DATA and 18-bytes header)
Jumbo Frame Size	9022 bytes	Typical industry standard. Ethernet jumbo frame size treated as 9018 bytes.

Software Visualization of Programmable Packet Size

The *Visualization of Programmable Packet Size* table provides the byte sizes of packets with various configurations.

Table 31-53: Visualization of Programmable Packet Size

Size in Bytes	Comments
16384	Receive watchdog and transmit jabber disabled, jumbo frames enabled.
10240	Receive watchdog and transmit jabber disabled, jumbo frames disabled.
2048	Receive watchdog and transmit jabber enabled.
1518	Typical max size of Ethernet frame. Receive watchdog and transmit jabber enabled.
64	Typical minimum size of Ethernet frame.
< 64	Runt frames requiring Zero-PAD.

Ethernet Packet Structure in C

The following is an example for Ethernet packet structure in the C language.

```
typedef struct ETHER_PACKET
{
    char  dst_addr[6];              //destination address
    char  src_addr[6];              //source address
    char  length[2];                //length of actual data
    char  data[DATA_SIZE];          //application data
    char  fdlimit[DELIMIT_SIZE];    //32-bit delimit (if manual appending)
    char  fcs[4];                   //crc frame checksum, used by RX buffer.
} ETHER_PACKET;
```

DMA Descriptor Implementation in C

The following code is a simple implementation of descriptors in ring and chain model in C language. Typically 4 WORDs (32-bit) are used for descriptors. Using checksum offload or the PTP engine requires 8 WORDs. Only high-level common functions across transmit and receive descriptors are considered here.

```
/* DMA Ring Descriptor */
typedef struct EMAC_DMADESC_RING
{
    unsigned int        Status;     //TDES0 OR RDES0
    unsigned int        ControlDesc;  //TDES1 OR RDES1
    unsigned int        StartAddr1;   //TDES2 OR RDES2
    unsigned int        StartAddr2;   //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT      ExtendedStat;
    #endif
} EMAC_DMADESC_RING;
/* DMA Chain Descriptor */
typedef struct EMAC_DMADESC_CHAIN
{
    unsigned int        Status;        //TDES0 OR RDES0
    unsigned int        ControlDesc;      //TDES1 OR RDES1
    unsigned int        StartAddr;     //TDES2 OR RDES2
    struct EMAC_DMADESC_CHAIN     *pNextDesc;  //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT     ExtendedStat;
    #endif
} EMAC_DMADESC_CHAIN;
/* Extended Status Descriptor with PTP not enabled*/
typedef struct EMAC_EXT_STAT
{
    #ifdef RX_DESC
    unsigned int        CheckSumStat;    //RDES4
    #ifdef TX_DESC
    unsigned int        Reserved;     //TDES4
    #endif
    unsigned int        Reserved;        //RDES5 OR TDES5
    unsigned int        Reserved;     //RDES6 OR TDES6
    unsigned int        Reserved;     //RDES7 OR TDES7
} EMAC_EXT_STAT;
```

PTP Header Structure in C

The following code is an example of the PTP message format.

```
/* PTP Message Format (Refer to PTP Frame Over IPv4)*/
    typedef struct EMAC_PTP_HEADER
    {
    unsigned char    messageType:4,    //PTP Version 2 message type
    transportSpecific:4;
    unsigned char    versionPTP;       //PTP Version (1 or 2)
    unsigned short    messageLength;
    unsigned char    domainNumber;
    unsigned char    RESERVED1;
    unsigned short    flagField;
    unsigned char    correctionField[8];
    unsigned char    RESERVED2[4];
    unsigned char    sourcePortIdentity[10];
    unsigned short    sequenceid;
    unsigned char    controlField;     //PTP Version 1 message type
    unsigned char    logMessageInterval;
    }EMAC_PTP_HEADER;
```

ADSP-SC58x EMAC Register Descriptions

Ethernet MAC (EMAC) contains the following registers.

Table 31-54: ADSP-SC58x EMAC Register List

Name	Description
`EMAC_ADDR0_HI`	MAC Address 0 High Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_ADDR1_HI	MAC Address 1 High Register
EMAC_ADDR1_LO	MAC Address 1 Low Register
EMAC_DBG	Debug Register
EMAC_DMA0_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA0_BMSTAT	DMA SCB Status Register
EMAC_DMA0_BUSMODE	DMA Bus Mode Register
EMAC_DMA0_IEN	DMA Interrupt Enable Register
EMAC_DMA0_MISS_FRM	DMA Missed Frame Register
EMAC_DMA0_OPMODE	DMA Operation Mode Register
EMAC_DMA0_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA0_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA0_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA0_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA0_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA0_STAT	DMA Status Register
EMAC_DMA0_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA0_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA0_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA0_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA1_BUSMODE	DMA Bus Mode Register
EMAC_DMA1_CHCBSCTL	Channel 1 Credit Shaping Control Register
EMAC_DMA1_CHCBSSTAT	Channel 1 Average Traffic Transmitted Register
EMAC_DMA1_CHHIC	Channel 1 High Credit Value Register
EMAC_DMA1_CHISC	Channel 1 Idle Slope Credit Value Register
EMAC_DMA1_CHLOC	Channel 1 Low Credit Value Register
EMAC_DMA1_CHSFCS	Channel 1 Control Bits for Slot Function Register
EMAC_DMA1_CHSSC	Channel 1 Send Slope Credit Value Register
EMAC_DMA1_IEN	DMA Interrupt Enable Register
EMAC_DMA1_MISS_FRM	DMA Missed Frame Register
EMAC_DMA1_OPMODE	DMA Operation Mode Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_DMA1_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA1_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA1_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA1_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA1_RXPOLL	DMA Rx Poll Demand Register
EMAC_DMA1_STAT	DMA Status Register
EMAC_DMA1_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA1_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA1_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA1_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA2_BUSMODE	DMA Bus Mode Register
EMAC_DMA2_CHCBSCTL	Channel 2 Credit Shaping Control Register
EMAC_DMA2_CHCBSSTAT	Channel 2 Avg Traffic Transmitted Status Register
EMAC_DMA2_CHHIC	Channel 2 High Credit Value Register
EMAC_DMA2_CHISC	Channel 2 Idle Slope Credit Value Register
EMAC_DMA2_CHLOC	Channel 2 Low Credit Value Register
EMAC_DMA2_CHSFCS	Channel 2 Control Bits for Slot Function Register
EMAC_DMA2_CHSSC	Channel 2 Send Slope Credit Value Register
EMAC_DMA2_IEN	DMA Interrupt Enable Register
EMAC_DMA2_MISS_FRM	DMA Missed Frame Register
EMAC_DMA2_OPMODE	DMA Operation Mode Register
EMAC_DMA2_RXBUF_CUR	DMA Rx Buffer Current Register
EMAC_DMA2_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA2_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA2_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA2_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA2_STAT	DMA Status Register
EMAC_DMA2_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA2_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA2_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA2_TXPOLL	DMA Tx Poll Demand Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_FLOWCTL	FLow Control Register
EMAC_GIGE_CTLSTAT	RGMII Control and Status Register
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_IMSK	Interrupt Mask Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_ISTAT	Interrupt Status Register
EMAC_L3L4_CTL	Layer3 and Layer4 Control Register
EMAC_L3_ADDR0	Layer 3 Address0 Register
EMAC_L3_ADDR1	Layer 3 Address1 Register
EMAC_L3_ADDR2	Layer 3 Address2 Register
EMAC_L3_ADDR3	Layer 3 Address3 Register
EMAC_L4_ADDR	Layer 4 Address Register
EMAC_LPI_CTLSTAT	Low Power Idle Control and Status Register
EMAC_LPI_TMRSCTL	Low Power Idle Timeout Register
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register
EMAC_MAC_AVCTL	AV MAC Control Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RXALIGN_ERR	Rx alignment Error Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXCTLFRM_G	Rx Good Control Frames Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXRCV_ERR	Rx Error Frames Received Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_CTL	Time Stamp Control Register
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_PPS0INTVL	Time Stamp PPS Interval Register
EMAC_TM_PPS0NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_PPS0TGTM	Time Stamp Target Time Seconds Register
EMAC_TM_PPS0WIDTH	PPS Width Register
EMAC_TM_PPS1INTVL	PPS 1 Interval Register
EMAC_TM_PPS1NTGTM	PPS 1 Target Time Nanoseconds Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
EMAC_TM_PPS1TGTM	PPS 1 Target Time Seconds Register
EMAC_TM_PPS1WIDTH	PPS 1 Width Register
EMAC_TM_PPS2INTVL	PPS 2 Interval Register
EMAC_TM_PPS2NTGTM	PPS 2 Target Time Nanoseconds Register
EMAC_TM_PPS2TGTM	PPS 2 Target Time Seconds Register
EMAC_TM_PPS2WIDTH	PPS 2 Width Register
EMAC_TM_PPS3INTVL	PPS 3 Interval Register
EMAC_TM_PPS3NTGTM	PPS 3 Target Time Nanoseconds Register
EMAC_TM_PPS3TGTM	PPS 3 Target Time Seconds Register
EMAC_TM_PPS3WIDTH	PPS 3 Width Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register

Table 31-54: ADSP-SC58x EMAC Register List (Continued)

Name	Description
`EMAC_TXMCASTFRM_GB`	Tx Multicast Frames (Good/Bad) Register
`EMAC_TXMULTCOL_G`	Tx Multiple Collision (Good) Register
`EMAC_TXOCTCNT_G`	Tx Octet Count (Good) Register
`EMAC_TXOCTCNT_GB`	Tx OCT Count (Good/Bad) Register
`EMAC_TXOVRSIZE_G`	Number of Tx Frames (Good) greater than maxsize
`EMAC_TXPAUSEFRM`	Tx Pause Frame Register
`EMAC_TXSNGCOL_G`	Tx Single Collision (Good) Register
`EMAC_TXUCASTFRM_GB`	Tx Unicast Frames (Good/Bad) Register
`EMAC_TXUNDR_ERR`	Tx Underflow Error Register
`EMAC_TXVLANFRM_G`	Tx VLAN Frames (Good) Register
`EMAC_VLANTAG`	VLAN Tag Register
`EMAC_VLAN_HSHTBL`	VLAN Hash Table Register
`EMAC_VLAN_INCL`	VLAN Tag Inclusion or Replacement Register
`EMAC_WDOG_TIMOUT`	Watchdog Timeout Register

MAC Address 0 High Register

The EMAC_ADDR0_HI register holds the address 0 high bits.

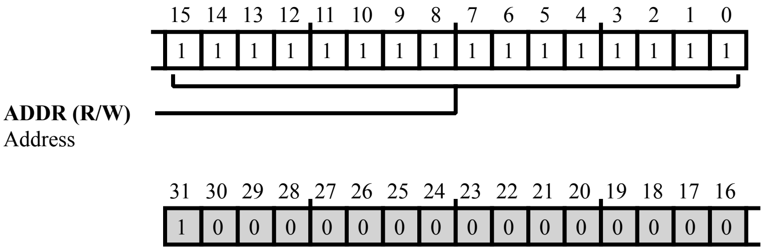

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
ADDR (R/W)
Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-23: EMAC_ADDR0_HI Register Diagram

Table 31-55: EMAC_ADDR0_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	ADDR	Address. The EMAC_ADDR0_HI.ADDR bits contain the upper 16 bits (47:32) of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

MAC Address 0 Low Register

The `EMAC_ADDR0_LO` register holds the address 0 low bits.

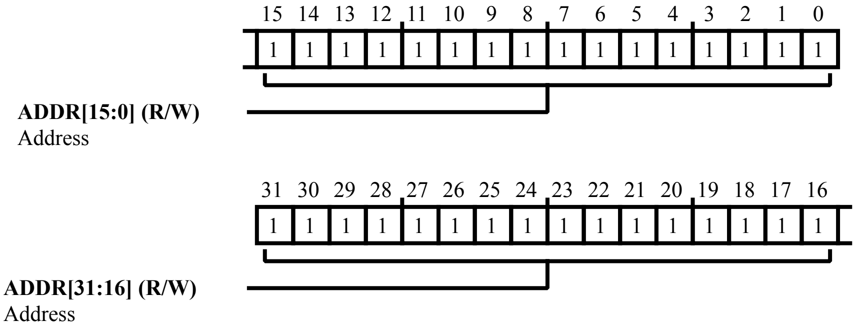

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
ADDR[15:0] (R/W)
Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
ADDR[31:16] (R/W)
Address

Figure 31-24: EMAC_ADDR0_LO Register Diagram

Table 31-56: EMAC_ADDR0_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address. The `EMAC_ADDR0_LO.ADDR` bits contain the lower 32 bits of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

MAC Address 1 High Register

The `EMAC_ADDR1_HI` register Contains the higher 16 bits of the second MAC address.

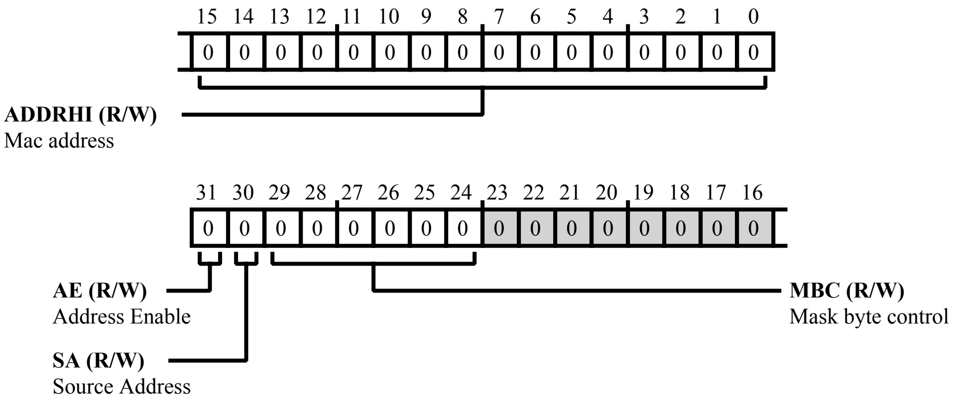

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRHI (R/W)
Mac address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AE (R/W)
Address Enable
SA (R/W)
Source Address
MBC (R/W)
Mask byte control

Figure 31-25: EMAC_ADDR1_HI Register Diagram

Table 31-57: EMAC_ADDR1_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	AE	Address Enable. The `EMAC_ADDR1_HI.AE` bit, When this bit is set, the address filter module uses the second MAC address for perfect filtering.
30 (R/W)	SA	Source Address. The `EMAC_ADDR1_HI.SA` bit, When this bit is set, the MAC Address1[47:0] is used to compare with the SA fields of the received frame.
29:24 (R/W)	MBC	Mask byte control. The `EMAC_ADDR1_HI.MBC` bit, are mask control bits for comparison of each of the MAC Address bytes.
15:0 (R/W)	ADDRHI	Mac address. The `EMAC_ADDR1_HI.ADDRHI` bit, contains the upper 16 bits (47:32) of the second 6-byte MAC address.

MAC Address 1 Low Register

The `EMAC_ADDR1_LO` register Contains the lower 32 bits of the second MAC address.

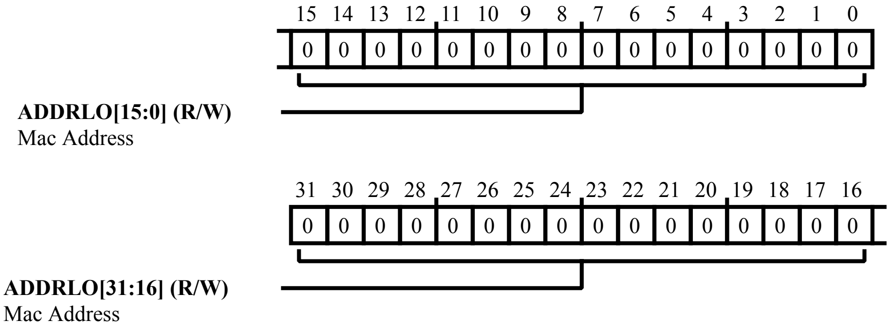

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRLO[15:0] (R/W)
Mac Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRLO[31:16] (R/W)
Mac Address

Figure 31-26: EMAC_ADDR1_LO Register Diagram

Table 31-58: EMAC_ADDR1_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDRLO	Mac Address. The `EMAC_ADDR1_LO.ADDRLO` bit, contains the lower 32 bits of the first 6-byte MAC address. This is used by the MAC for filtering the received frames and inserting the MAC address in the Transmit Flow Control (Pause) Frames.

Debug Register

The EMAC_DBG register contains EMAC debug status information.

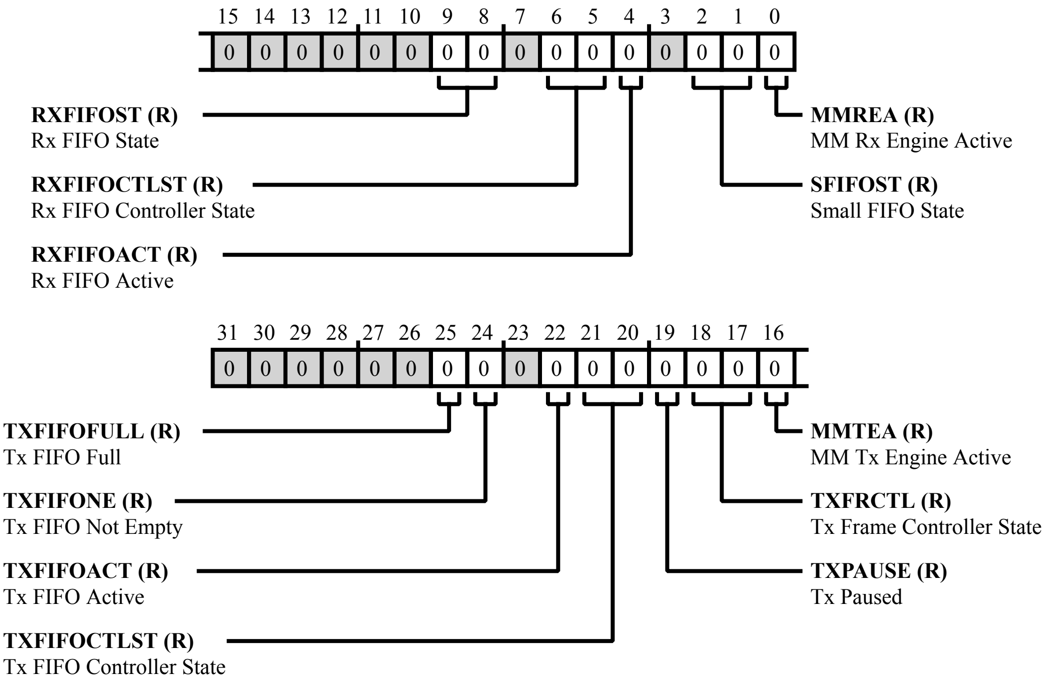

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXFIFOST (R)
Rx FIFO State
RXFIFOCTLST (R)
Rx FIFO Controller State
RXFIFOACT (R)
Rx FIFO Active
MMREA (R)
MM Rx Engine Active
SFIFOST (R)
Small FIFO State
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TXFIFOFULL (R)
Tx FIFO Full
TXFIFONE (R)
Tx FIFO Not Empty
TXFIFOACT (R)
Tx FIFO Active
TXFIFOCTLST (R)
Tx FIFO Controller State
MMTEA (R)
MM Tx Engine Active
TXFRCTL (R)
Tx Frame Controller State
TXPAUSE (R)
Tx Paused

Figure 31-27: EMAC_DBG Register Diagram

Table 31-59: EMAC_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	TXFIFOFULL	Tx FIFO Full. The `EMAC_DBG.TXFIFOFULL` bit, when high, indicates that the MFL TxStatus FIFO is full, and the MFL cannot accept any more frames for transmission.
24 (R/NW)	TXFIFONE	Tx FIFO Not Empty. The `EMAC_DBG.TXFIFONE` bit, when high, indicates that the MFL TxFIFO is not empty and has some data left for transmission.
22 (R/NW)	TXFIFOACT	Tx FIFO Active. The `EMAC_DBG.TXFIFOACT` bit, when high, indicates that the MFL TxFIFO write controller is active and transferring data to the TxFIFO.
21:20 (R/NW)	TXFIFOCTLST	Tx FIFO Controller State. The `EMAC_DBG.TXFIFOCTLST` bits indicate the state of the TxFIFO read controller as: 00=IDLE state, 01=READ state (transferring data to MAC transmitter), 10=Waiting for TxStatus from MAC transmitter, and 11=Writing the received TxStatus or flushing the TxFIFO

Table 31-59: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/NW)	TXPAUSE	Tx Paused. The `EMAC_DBG.TXPAUSE` bit, when high, indicates that the MAC transmitter is in PAUSE condition (in full-duplex only) and does not schedule any frame for transmission.	
18:17 (R/NW)	TXFRCTL	Tx Frame Controller State. The `EMAC_DBG.TXFRCTL` bits indicate the state of the MAC transmit frame controller module.	
		0	Idle Frame controller is in idle state.
		1	Wait Frame controller is waiting for status of previous frame or IFG/backoff period end.
		2	Pause Frame controller is generating and transmitting a PAUSE control frame (in full duplex mode).
		3	Transmit Frame controller is transferring input frame for transmission.
16 (R/NW)	MMTEA	MM Tx Engine Active. The `EMAC_DBG.MMTEA` bit, when high, indicates that the MAC core transmit protocol engine is actively transmitting data and is not in IDLE state.	
9:8 (R/NW)	RXFIFOST	Rx FIFO State. The `EMAC_DBG.RXFIFOST` bits give the status of the RxFIFO fill level and indicate the relationship to the flow-control activation threshold.	
		0	Rx FIFO Empty
		1	Rx FIFO Below De-activate FCT
		2	Rx FIFO Above De-activate FCT
		3	Rx FIFO Full
6:5 (R/NW)	RXFIFOCTLST	Rx FIFO Controller State. The `EMAC_DBG.RXFIFOCTLST` bits give the state of the RxFIFO read controller.	
		0	Idle Read controller is in idle state.
		1	Read Data Read controller is reading frame data.
		2	Read Status Read controller is reading frame status or time-stamp.
		3	Flush Read controller is flushing the frame data and status.
4 (R/NW)	RXFIFOACT	Rx FIFO Active. The `EMAC_DBG.RXFIFOACT` bit, when high, indicates that the MFL RxFIFO write controller is active and is transferring a received frame to the FIFO.	

Table 31-59: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:1 (R/NW)	SFIFOST	Small FIFO State. The `EMAC_DBG.SFIFOST` bit, when high, indicates the active state of the small FIFO read and write controllers respectively of the MAC receive frame controller module.
0 (R/NW)	MMREA	MM Rx Engine Active. The `EMAC_DBG.MMREA` bit, when high, indicates that the MAC core receive protocol engine is actively receiving data and is not in IDLE state.

DMA SCB Bus Mode Register

The EMAC_DMA0_BMMODE register selects EMAC DMA system cross bar bus mode features.

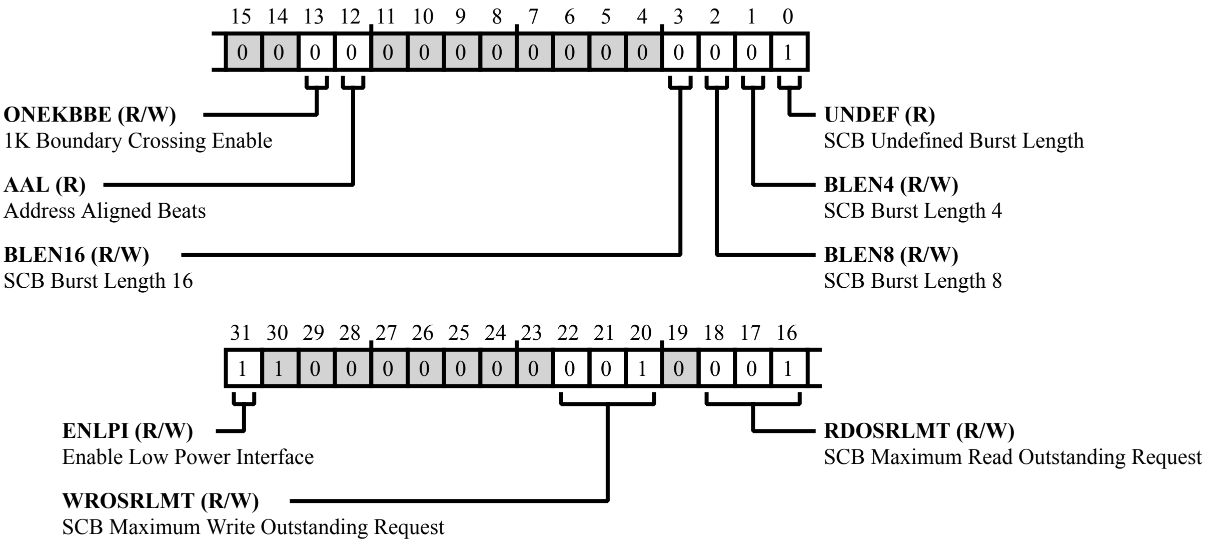

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
ONEKBBE (R/W)
1K Boundary Crossing Enable
AAL (R)
Address Aligned Beats
BLEN16 (R/W)
SCB Burst Length 16
UNDEF (R)
SCB Undefined Burst Length
BLEN4 (R/W)
SCB Burst Length 4
BLEN8 (R/W)
SCB Burst Length 8
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1
ENLPI (R/W)
Enable Low Power Interface
WROSRLMT (R/W)
SCB Maximum Write Outstanding Request
RDOSRLMT (R/W)
SCB Maximum Read Outstanding Request

Figure 31-28: EMAC_DMA0_BMMODE Register Diagram

Table 31-60: EMAC_DMA0_BMMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ENLPI	Enable Low Power Interface. The EMAC_DMA0_BMMODE.ENLPI bit field's when set to 1, it enable LPI mode supported by the GMAC configuration and accepts the LPI request from the system Clock controller.
22:20 (R/W)	WROSRLMT	SCB Maximum Write Outstanding Request. The EMAC_DMA0_BMMODE.WROSRLMT bit field's value limits the maximum outstanding request on the SCB write interface. Maximum outstanding requests = WR_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding write requests.
18:16 (R/W)	RDOSRLMT	SCB Maximum Read Outstanding Request. The EMAC_DMA0_BMMODE.RDOSRLMT bit field's value limits the maximum outstanding request on the SCB read interface. Maximum outstanding requests = RD_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding read requests.
13 (R/W)	ONEKBBE	1K Boundary Crossing Enable. When the EMAC_DMA0_BMMODE.ONEKBBE bit is set, the GMAC extensible bus master performs burst transfers that do not cross 1 KB boundary. When reset, the GMAC extensible bus master performs burst transfers that do not cross 4 KB boundary.

Table 31-60: EMAC_DMA0_BMMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/NW)	AAL	Address Aligned Beats. The `EMAC_DMA0_BMMODE.AAL` bit (read-only) reflects the state of the `EMAC_DMA0_BUSMODE.AAL` bit. When this bit is set to 1, EMAC-SCB performs address-aligned burst transfers on both read and write channels.
3 (R/W)	BLEN16	SCB Burst Length 16. The `EMAC_DMA0_BMMODE.BLEN16` bit, when set (or when `EMAC_DMA0_BMMODE.UNDEF` is set to 1), directs the EMAC-SCB to select a burst length of 16 on the SCB master interface.
2 (R/W)	BLEN8	SCB Burst Length 8. The `EMAC_DMA0_BMMODE.BLEN8` bit, when set (or when `EMAC_DMA0_BMMODE.UNDEF` is set to 1), directs the EMAC-SCB to select a burst length of 8 on the SCB master interface.
1 (R/W)	BLEN4	SCB Burst Length 4. The `EMAC_DMA0_BMMODE.BLEN4` bit, when set (or when `EMAC_DMA0_BMMODE.UNDEF` is set to 1), directs the EMAC-SCB to select a burst length of 4 on the SCB master interface.
0 (R/NW)	UNDEF	SCB Undefined Burst Length. The `EMAC_DMA0_BMMODE.UNDEF` bit (read-only) indicates the complement (invert) value of `EMAC_DMA0_BUSMODE.FB` bit . When this bit is set to 1, the EMAC-SCB is allowed to perform any burst length equal to or below the maximum allowed burst length as programmed in bits[3:1]. When this bit is set to 0, the EMAC-SCB is allowed to perform only fixed burst lengths as indicated by 16/8/4, or a burst length of 1.

DMA SCB Status Register

The `EMAC_DMA0_BMSTAT` register indicates EMAC DMA system cross bar status.

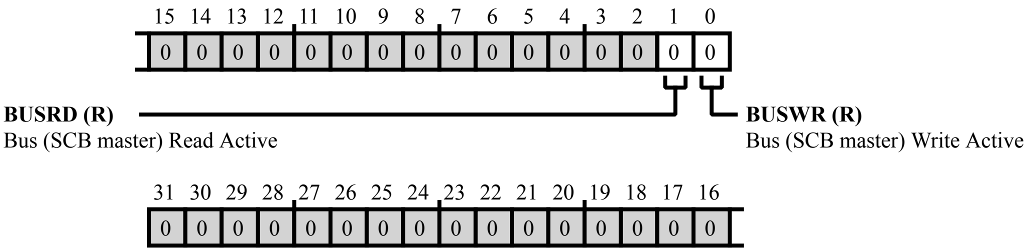

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BUSRD (R)
Bus (SCB master) Read Active
BUSWR (R)
Bus (SCB master) Write Active
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-29: EMAC_DMA0_BMSTAT Register Diagram

Table 31-61: EMAC_DMA0_BMSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	BUSRD	Bus (SCB master) Read Active. The `EMAC_DMA0_BMSTAT.BUSRD` bit, when high, indicates that SCB Master's read channel is active and transferring data.
0 (R/NW)	BUSWR	Bus (SCB master) Write Active. The `EMAC_DMA0_BMSTAT.BUSWR` bit, when high, indicates that SCB Master's write channel is active and transferring data.

DMA Bus Mode Register

The `EMAC_DMA0_BUSMODE` register selects the DMA bus operating modes for EMAC DMA.

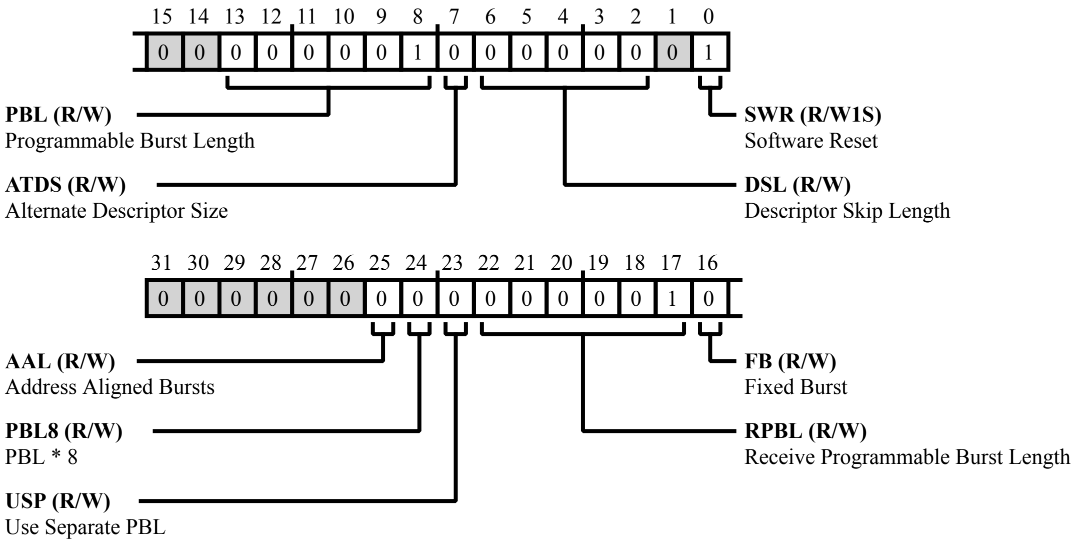

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
PBL (R/W)
Programmable Burst Length
ATDS (R/W)
Alternate Descriptor Size
SWR (R/W1S)
Software Reset
DSL (R/W)
Descriptor Skip Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
AAL (R/W)
Address Aligned Bursts
PBL8 (R/W)
PBL * 8
USP (R/W)
Use Separate PBL
FB (R/W)
Fixed Burst
RPBL (R/W)
Receive Programmable Burst Length

Figure 31-30: EMAC_DMA0_BUSMODE Register Diagram

Table 31-62: EMAC_DMA0_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	AAL	Address Aligned Bursts. The `EMAC_DMA0_BUSMODE.AAL` bit, when set high and the FB bit equals 1, directs the SCB interface to generate all bursts aligned to the start address LS bits. If the FB bit is equal to 0, the first burst (accessing the data buffers start address) is not aligned, but subsequent bursts are aligned to the address.
24 (R/W)	PBL8	PBL * 8. The `EMAC_DMA0_BUSMODE.PBL8` bit, when set high, multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, and 32 beats depending on the PBL value.
23 (R/W)	USP	Use Separate PBL. The `EMAC_DMA0_BUSMODE.USP` bit, when set high, configures the Rx DMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to Tx DMA operations only.

Table 31-62: EMAC_DMA0_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22:17 (R/W)	RPBL	Receive Programmable Burst Length. The `EMAC_DMA0_BUSMODE.RPBL` bits indicate the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read/Write. The Rx DMA always attempts to burst as specified in RPBL every time it starts a Burst transfer on the host bus. RPBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. These bits are valid and applicable only when USP is set high.
16 (R/W)	FB	Fixed Burst. The `EMAC_DMA0_BUSMODE.FB` bit controls whether the SCB Master interface performs fixed burst transfers or not. See the `EMAC_DMA0_BMMODE.UNDEF` bit description for more information.
13:8 (R/W)	PBL	Programmable Burst Length. The `EMAC_DMA0_BUSMODE.PBL` bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read/Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the host bus. Any other value results in undefined behavior. When USP is set high, this PBL value is applicable for Tx DMA transactions only. PBL-max limit = (FIFO size / 2) / 4. PBL-max limit (transmit) = 256 bytes / 2 /4 = 32. PBL-max limit (receive) = 128 bytes / 2 /4 = 16. Note that this PBL is at the DMA end. If PBL= 32 and if BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If `EMAC_DMA0_BUSMODE.PBL` =8, and if `EMAC_DMA0_BMMODE.BLEN16` is enabled, the max burst is limited to `EMAC_DMA0_BMMODE.BLEN8`. If `EMAC_DMA0_BUSMODE.PBL8` bit is set, the programmed PBL value is multiplied by 8 times internally. However, the result cannot be more than the above maximum limits specified above.
7 (R/W)	ATDS	Alternate Descriptor Size. The `EMAC_DMA0_BUSMODE.ATDS` bit, when set, increases the size of the alternate descriptor to 32 bytes (8 DWORDS). This is required when the Advanced Time Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). The enhanced descriptor is not required if the Advanced Time Stamp and IPC Full Checksum Offload features are not enabled. In such case, you can use the 16 bytes descriptor to save 4 bytes of memory.

Table 31-62: EMAC_DMA0_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:2 (R/W)	DSL	Descriptor Skip Length. The `EMAC_DMA0_BUSMODE.DSL` bit specifies the number of 32-bit words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.
0 (R/W1S)	SWR	Software Reset. The `EMAC_DMA0_BUSMODE.SWR` bit, when set, directs the MAC DMA Controller to reset all MAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in this bit before re-programming any register of the core. Note: The reset operation is completed only when all the resets in all the active clock domains are de-asserted. Therefore, it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for software reset completion. This field cleared to 1b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.

DMA Interrupt Enable Register

The EMAC_DMA0_IEN register enables (unmasks) EMAC DMA interrupts.

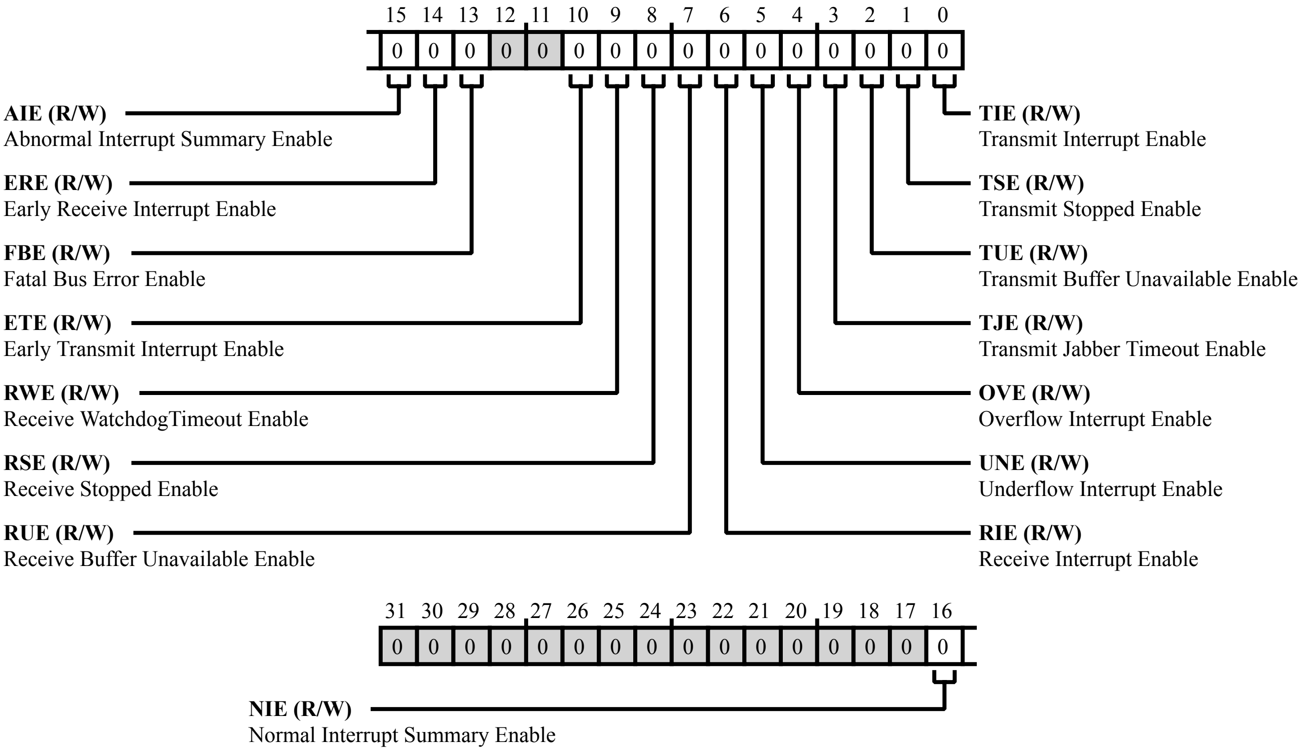

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIE (R/W)
Abnormal Interrupt Summary Enable
ERE (R/W)
Early Receive Interrupt Enable
FBE (R/W)
Fatal Bus Error Enable
ETE (R/W)
Early Transmit Interrupt Enable
RWE (R/W)
Receive WatchdogTimeout Enable
RSE (R/W)
Receive Stopped Enable
RUE (R/W)
Receive Buffer Unavailable Enable
TIE (R/W)
Transmit Interrupt Enable
TSE (R/W)
Transmit Stopped Enable
TUE (R/W)
Transmit Buffer Unavailable Enable
TJE (R/W)
Transmit Jabber Timeout Enable
OVE (R/W)
Overflow Interrupt Enable
UNE (R/W)
Underflow Interrupt Enable
RIE (R/W)
Receive Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NIE (R/W)
Normal Interrupt Summary Enable

Figure 31-31: EMAC_DMA0_IEN Register Diagram

Table 31-63: EMAC_DMA0_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	NIE	Normal Interrupt Summary Enable. The EMAC_DMA0_IEN.NIE bit, when set, enables a normal interrupt. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: EMAC_DMA0_STAT.TI, EMAC_DMA0_STAT.TU, EMAC_DMA0_STAT.RI, and EMAC_DMA0_STAT.ERI.
15 (R/W)	AIE	Abnormal Interrupt Summary Enable. The EMAC_DMA0_IEN.AIE bit, when set, enables an abnormal interrupt. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits: EMAC_DMA0_STAT.TPS, EMAC_DMA0_STAT.TJT, EMAC_DMA0_STAT.OVF, EMAC_DMA0_STAT.RU, EMAC_DMA0_STAT.RPS, EMAC_DMA0_STAT.RWT, EMAC_DMA0_STAT.ETI, and EMAC_DMA0_STAT.FBI.

Table 31-63: EMAC_DMA0_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	ERE	Early Receive Interrupt Enable. The `EMAC_DMA0_IEN.ERE` bit, when set (and with `EMAC_DMA0_IEN.NIE` =1), enables the Early Receive Interrupt. When this bit is reset, Early Receive Interrupt is disabled.
13 (R/W)	FBE	Fatal Bus Error Enable. The `EMAC_DMA0_IEN.FBE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Fatal Bus Error Interrupt. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.
10 (R/W)	ETE	Early Transmit Interrupt Enable. The `EMAC_DMA0_IEN.ETE` bit, when this bit is set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Early Transmit Interrupt. When this bit is reset, Early Transmit Interrupt is disabled.
9 (R/W)	RWE	Receive WatchdogTimeout Enable. The `EMAC_DMA0_IEN.RWE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Receive Watchdog Timeout Interrupt. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.
8 (R/W)	RSE	Receive Stopped Enable. The `EMAC_DMA0_IEN.RSE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.
7 (R/W)	RUE	Receive Buffer Unavailable Enable. The `EMAC_DMA0_IEN.RUE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Receive Buffer Unavailable Interrupt. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
6 (R/W)	RIE	Receive Interrupt Enable. The `EMAC_DMA0_IEN.RIE` bit, when set (and with `EMAC_DMA0_IEN.NIE` =1), enables the Receive Interrupt. When this bit is reset, Receive Interrupt is disabled.
5 (R/W)	UNE	Underflow Interrupt Enable. The `EMAC_DMA0_IEN.UNE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Transmit Underflow Interrupt. When this bit is reset, Underflow Interrupt is disabled.
4 (R/W)	OVE	Overflow Interrupt Enable. The `EMAC_DMA0_IEN.OVE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Receive Overflow Interrupt. When this bit is reset, Overflow Interrupt is disabled.

Table 31-63: EMAC_DMA0_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TJE	Transmit Jabber Timeout Enable. The `EMAC_DMA0_IEN.TJE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Transmit Jabber Timeout Interrupt. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.
2 (R/W)	TUE	Transmit Buffer Unavailable Enable. The `EMAC_DMA0_IEN.TUE` bit, when set (and with `EMAC_DMA0_IEN.NIE` =1), enables the Transmit Buffer Unavailable Interrupt. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.
1 (R/W)	TSE	Transmit Stopped Enable. The `EMAC_DMA0_IEN.TSE` bit, when set (and with `EMAC_DMA0_IEN.AIE` =1), enables the Transmission Stopped Interrupt. When this bit is reset, Transmission Stopped Interrupt is disabled.
0 (R/W)	TIE	Transmit Interrupt Enable. The `EMAC_DMA0_IEN.TIE` bit, when set (and with `EMAC_DMA0_IEN.NIE` =1), enables the Transmit Interrupt. When this bit is reset, Transmit Interrupt is disabled.

DMA Missed Frame Register

The `EMAC_DMA0_MISS_FRM` register contains counters for EMAC DMA missed frames and buffer overflows.

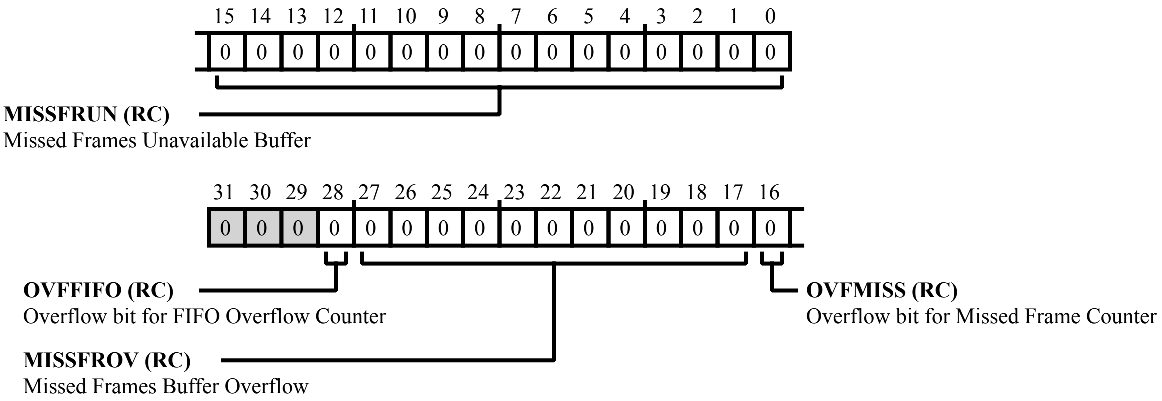

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISSFRUN (RC)
Missed Frames Unavailable Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OVFFIFO (RC)
Overflow bit for FIFO Overflow Counter
MISSFROV (RC)
Missed Frames Buffer Overflow
OVFMISS (RC)
Overflow bit for Missed Frame Counter

Figure 31-32: EMAC_DMA0_MISS_FRM Register Diagram

Table 31-64: EMAC_DMA0_MISS_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (RC/NW)	OVFFIFO	Overflow bit for FIFO Overflow Counter. The `EMAC_DMA0_MISS_FRM.OVFFIFO` bit holds the overflow bit for FIFO Overflow Counter.
27:17 (RC/NW)	MISSFROV	Missed Frames Buffer Overflow. The `EMAC_DMA0_MISS_FRM.MISSFROV` bits indicate the number of frames missed by the application due to buffer overflow.
16 (RC/NW)	OVFMISS	Overflow bit for Missed Frame Counter. The `EMAC_DMA0_MISS_FRM.OVFMISS` bit holds the overflow bit for the Missed Frame Counter.
15:0 (RC/NW)	MISSFRUN	Missed Frames Unavailable Buffer. The `EMAC_DMA0_MISS_FRM.MISSFRUN` bits indicate the number of frames missed by the controller because of the Application Receive Buffer being unavailable.

DMA Operation Mode Register

The EMAC_DMA0_OPMODE register selects receive and transmit DMA operating modes.

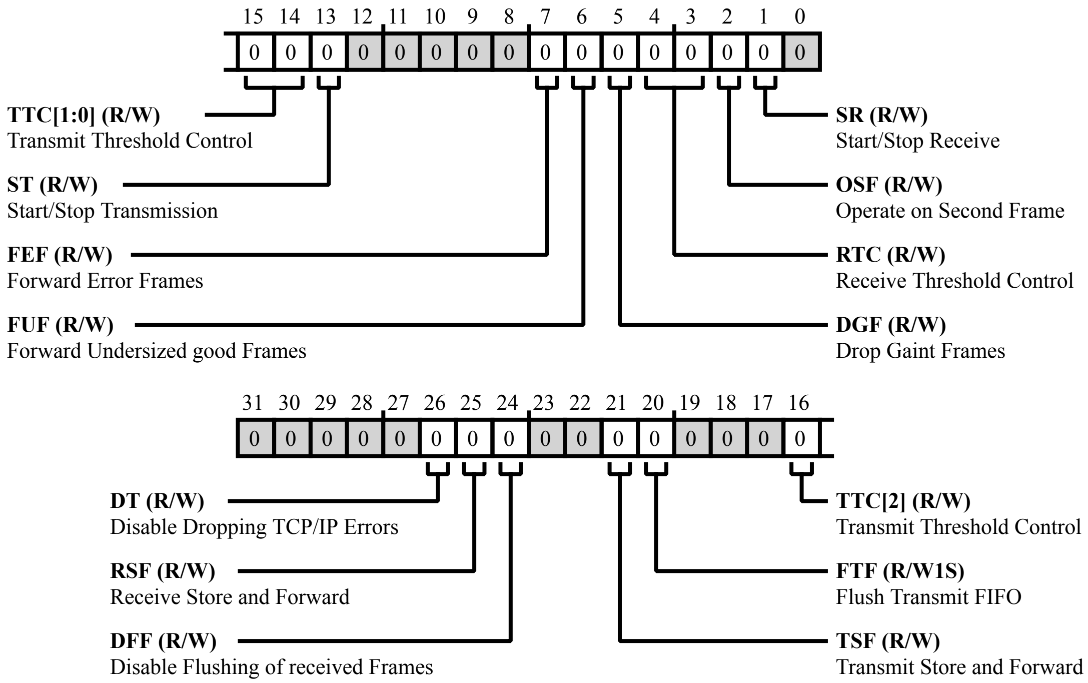

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TTC[1:0] (R/W)
Transmit Threshold Control
ST (R/W)
Start/Stop Transmission
FEF (R/W)
Forward Error Frames
FUF (R/W)
Forward Undersized good Frames
SR (R/W)
Start/Stop Receive
OSF (R/W)
Operate on Second Frame
RTC (R/W)
Receive Threshold Control
DGF (R/W)
Drop Gaint Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DT (R/W)
Disable Dropping TCP/IP Errors
RSF (R/W)
Receive Store and Forward
DFF (R/W)
Disable Flushing of received Frames
TTC[2] (R/W)
Transmit Threshold Control
FTF (R/W1S)
Flush Transmit FIFO
TSF (R/W)
Transmit Store and Forward

Figure 31-33: EMAC_DMA0_OPMODE Register Diagram

Table 31-65: EMAC_DMA0_OPMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	DT	Disable Dropping TCP/IP Errors. The EMAC_DMA0_OPMODE.DT bit, when set, directs the core not to drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the EMAC_DMA0_OPMODE.FEF bit is reset.
25 (R/W)	RSF	Receive Store and Forward. The EMAC_DMA0_OPMODE.RSF bit, when set, directs the MFL only to read a frame from the Rx FIFO after the complete frame has been written to it, ignoring the EMAC_DMA0_OPMODE.RTC bits. When this bit is reset, the Rx FIFO operates in threshold mode, subject to the threshold specified by the EMAC_DMA0_OPMODE.RTC bits.
24 (R/W)	DFF	Disable Flushing of received Frames. The EMAC_DMA0_OPMODE.DFF bit, when set, directs the Rx DMA not to flush any frames because of the unavailability of receive descriptors/buffers as it does normally when this bit is reset.

Table 31-65: EMAC_DMA0_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	TSF	Transmit Store and Forward. The `EMAC_DMA0_OPMODE.TSF` bit, when set, starts transmission when a full frame resides in the MFL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.	
20 (R/W1S)	FTF	Flush Transmit FIFO. The `EMAC_DMA0_OPMODE.FTF` bit, when set, directs the transmit FIFO controller logic to reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation completes only after emptying the Tx FIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock is required to be active. This field cleared to 1b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.	
16:14 (R/W)	TTC	Transmit Threshold Control. The `EMAC_DMA0_OPMODE.TTC` bits control the threshold level of the MFL Transmit FIFO. Transmission starts when the frame size within the MFL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the `EMAC_DMA0_OPMODE.TSF` bit is reset. The value =011 is not used.	
		0	64
		1	128
		2	192
		3	256
		4	40
		5	32
		6	24
		7	16

Table 31-65: EMAC_DMA0_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	ST	Start/Stop Transmission. The `EMAC_DMA0_OPMODE.ST` bit, when set, places transmission in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Transmit Descriptor List Address, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state, and the `EMAC_DMA0_STAT.TU` bit is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the `EMAC_DMA0_TXDSC_CUR` address register, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.
7 (R/W)	FEF	Forward Error Frames. The `EMAC_DMA0_OPMODE.FEF` bit, when reset, directs the Rx FIFO to drop frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frames start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. When `EMAC_DMA0_OPMODE.FEF` bit is set, all frames except runt error frames are forwarded to the DMA. But when Rx FIFO overflows when a partial frame is written, then such frames are dropped even when `EMAC_DMA0_OPMODE.FEF` is set.
6 (R/W)	FUF	Forward Undersized good Frames. The `EMAC_DMA0_OPMODE.FUF` bit, when set, directs the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO drops all frames of less than 64 bytes, unless it is already transferred because of lower value of Receive Threshold (for example, `EMAC_DMA0_OPMODE.RTC` =01).
5 (R/W)	DGF	Drop Gaint Frames. The `EMAC_DMA0_OPMODE.DGF` bit, when set, the MAC drops the received giant frames in the Rx FIFO, that is, frames that are larger than the computed giant frame limit. When reset, the MAC does not drop the giant frames in the Rx FIFO.

Table 31-65: EMAC_DMA0_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	RTC	Receive Threshold Control. The `EMAC_DMA0_OPMODE.RTC` bits control the threshold level of the MFL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MFL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. These bits are valid only when the `EMAC_DMA0_OPMODE.RSF` bit is zero, and are ignored when the `EMAC_DMA0_OPMODE.RSF` bit is set to 1. The value =11 is not used.	
		0	64
		1	32
		2	96
		3	128
2 (R/W)	OSF	Operate on Second Frame. The `EMAC_DMA0_OPMODE.OSF` bit, when set, instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.	
1 (R/W)	SR	Start/Stop Receive. The `EMAC_DMA0_OPMODE.SR` bit, when set, places the Receive process in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Receive Descriptor List Address or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended, and the `EMAC_DMA0_STAT.RU` bit is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting `EMAC_DMA0_RXDSC_CUR` address register, DMA behavior is unpredictable. When this bit is cleared, Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.	

DMA Rx Buffer Current Register

The `EMAC_DMA0_RXBUF_CUR` register holds the pointer to the current receive DMA buffer.

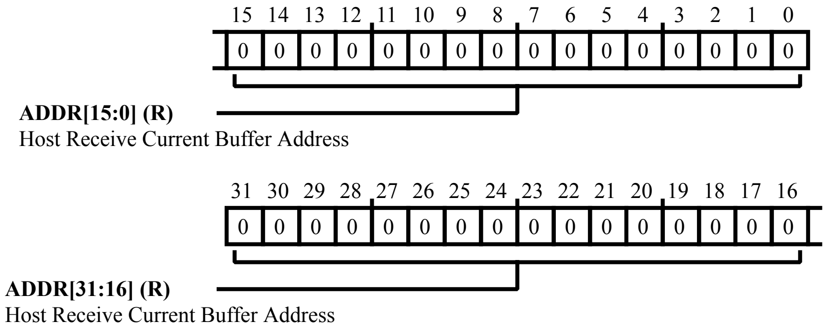

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Buffer Address

Figure 31-34: EMAC_DMA0_RXBUF_CUR Register Diagram

Table 31-66: EMAC_DMA0_RXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Buffer Address. The `EMAC_DMA0_RXBUF_CUR.ADDR` bit field points to the current Receive Buffer address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Descriptor List Address Register

The `EMAC_DMA0_RXDSC_ADDR` register holds the address for the DMA receive descriptor list. Writing to this Register is permitted only when reception is stopped. When stopped, this must be written to before the receive Start command is given. The processor can write to `EMAC_DMA0_RXDSC_ADDR` only when Rx DMA has stopped (`EMAC_DMA0_OPMODE.SR` bit =0). When stopped, it can be written with a new descriptor list address. When the processor sets the `EMAC_DMA0_OPMODE.SR` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA0_OPMODE.SR` bit is cleared to 0, the DMA takes the descriptor address where it was stopped earlier.

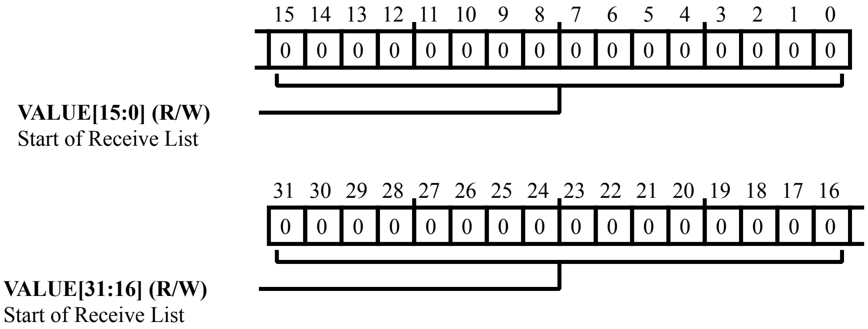

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Receive List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Receive List

Figure 31-35: EMAC_DMA0_RXDSC_ADDR Register Diagram

Table 31-67: EMAC_DMA0_RXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Receive List. The `EMAC_DMA0_RXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1:0] for the 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Rx Descriptor Current Register

The EMAC_DMA0_RXDSC_CUR register contains the current DMA receive descriptor.

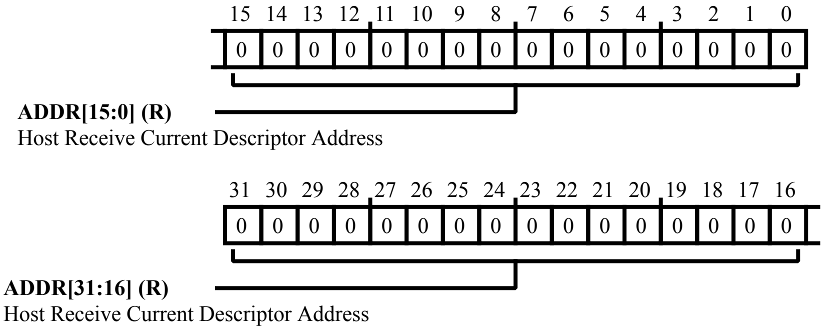

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Descriptor Address

Figure 31-36: EMAC_DMA0_RXDSC_CUR Register Diagram

Table 31-68: EMAC_DMA0_RXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Descriptor Address. The `EMAC_DMA0_RXDSC_CUR.ADDR` bit field points to the start address of the current Receive Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Interrupt Watch Dog Register

The `EMAC_DMA0_RXIWDOG` register contains the timeout value for the EMAC DMA receive interrupt watch dog timer.

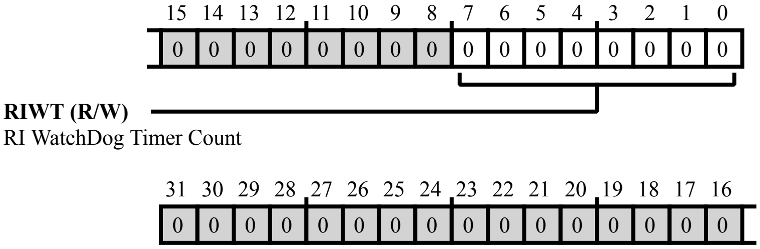

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RIWT (R/W)
RI WatchDog Timer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-37: EMAC_DMA0_RXIWDOG Register Diagram

Table 31-69: EMAC_DMA0_RXIWDOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	RIWT	RI WatchDog Timer Count. The `EMAC_DMA0_RXIWDOG.RIWT` bit field indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watch-dog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when `EMAC_DMA0_STAT.RI` bit is set high because of automatic setting of `EMAC_DMA0_STAT.RI` as per RDES1[31] of any received frame.

DMA Rx Poll Demand register

The `EMAC_DMA0_RXPOLL` register directs the EMAC to poll the receive descriptor list.

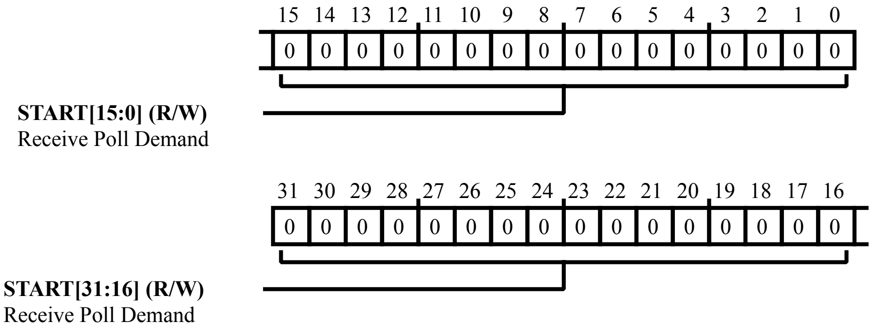

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Receive Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Receive Poll Demand

Figure 31-38: EMAC_DMA0_RXPOLL Register Diagram

Table 31-70: EMAC_DMA0_RXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Receive Poll Demand. The `EMAC_DMA0_RXPOLL.START` bits, when written with any value, cause the DMA to read the current descriptor pointed to by the `EMAC_DMA0_RXDSC_CUR` register. If that descriptor is not available (owned by application), reception returns to the Suspended state, and the `EMAC_DMA0_STAT.RU` bit is asserted. If the descriptor is available, the Receive DMA returns to the active state.

DMA Status Register

The `EMAC_DMA0_STAT` register indicates EMAC DMA status.

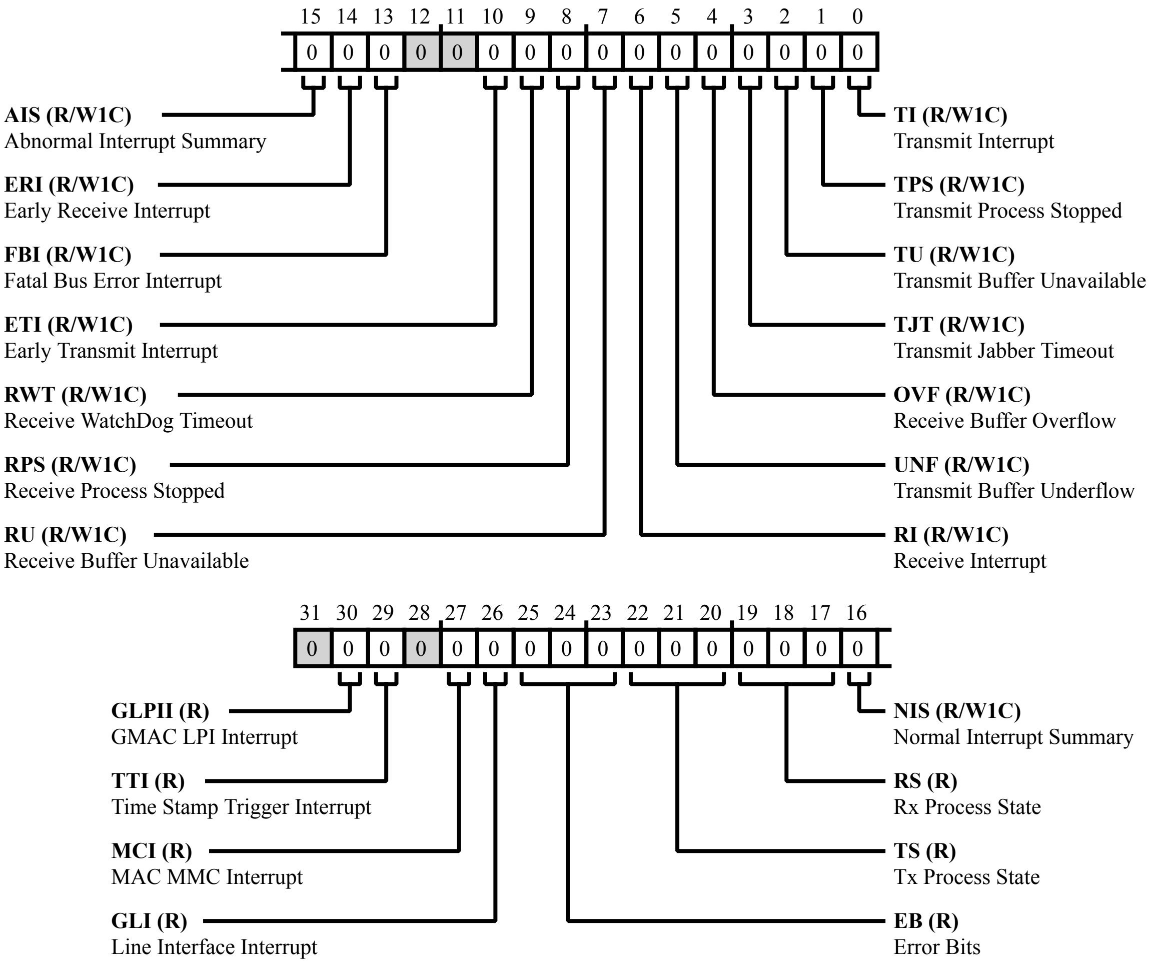

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIS (R/W1C)
Abnormal Interrupt Summary
ERI (R/W1C)
Early Receive Interrupt
FBI (R/W1C)
Fatal Bus Error Interrupt
ETI (R/W1C)
Early Transmit Interrupt
RWT (R/W1C)
Receive WatchDog Timeout
RPS (R/W1C)
Receive Process Stopped
RU (R/W1C)
Receive Buffer Unavailable
TI (R/W1C)
Transmit Interrupt
TPS (R/W1C)
Transmit Process Stopped
TU (R/W1C)
Transmit Buffer Unavailable
TJT (R/W1C)
Transmit Jabber Timeout
OVF (R/W1C)
Receive Buffer Overflow
UNF (R/W1C)
Transmit Buffer Underflow
RI (R/W1C)
Receive Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GLPII (R)
GMAC LPI Interrupt
TTI (R)
Time Stamp Trigger Interrupt
MCI (R)
MAC MMC Interrupt
GLI (R)
Line Interface Interrupt
NIS (R/W1C)
Normal Interrupt Summary
RS (R)
Rx Process State
TS (R)
Tx Process State
EB (R)
Error Bits

Figure 31-39: EMAC_DMA0_STAT Register Diagram

Table 31-71: EMAC_DMA0_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	GLPII	GMAC LPI Interrupt. The `EMAC_DMA0_STAT.GLPII` bit indicates an interrupt event in the LPI logic of the MAC. To reset this bit to 1'b0, the software must read the corresponding registers in the DWC_gmac to get the exact cause of the interrupt and clear its source.

Table 31-71: EMAC_DMA0_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/NW)	TTI	Time Stamp Trigger Interrupt. The `EMAC_DMA0_STAT.TTI` bit indicates an interrupt event in the MAC core's Time Stamp Generator block. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to =0. When this bit is high, the interrupt signal from the MAC is high.	
27 (R/NW)	MCI	MAC MMC Interrupt. The `EMAC_DMA0_STAT.MCI` bit reflects an interrupt event in the MMC module of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as =0. The interrupt signal from the MAC is high when this bit is high.	
26 (R/NW)	GLI	Line Interface Interrupt. The `EMAC_DMA0_STAT.GLI` bit When set, this bit reflects any of the following interrupt events in the DWC_gmac interfaces	
25:23 (R/NW)	EB	Error Bits. The `EMAC_DMA0_STAT.EB` bits indicate the type of error that caused a Bus Error (for example, error response on the SCB interface). These bits are valid only when the `EMAC_DMA0_STAT.FBI` bit is set. This field does not generate an interrupt.	
		0	Error during data buffer access, write transfer, Rx DMA
		1	Error during data buffer access, write transfer, Tx DMA
		2	Error during data buffer access, read transfer, Rx DMA
		3	Error during data buffer access, read transfer, Tx DMA
		4	Error during descriptor access, write transfer, Rx DMA
		5	Error during descriptor access, write transfer, Tx DMA
		6	Error during descriptor access, read transfer, Rx DMA
		7	Error during descriptor access, read transfer, Tx DMA
22:20 (R/NW)	TS	Tx Process State. The `EMAC_DMA0_STAT.TS` bits indicate the transmit DMA state. This field does not generate an interrupt.	
		0	Stopped; Reset or Stop Tx Command Issued
		1	Running; Fetching Tx Transfer Descriptor
		2	Running; Waiting for Status
		3	Reading Data from Host Memory Buffer and Queuing It to Tx Buffer
		4	TIME_STAMP Write State

Table 31-71: EMAC_DMA0_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Reserved
		6	Suspended; Tx Descriptor Unavailable or Tx Buffer Underflow
		7	Closing Tx Descriptor
19:17 (R/NW)	RS	Rx Process State. The `EMAC_DMA0_STAT.RS` bits indicate the receive DMA state. This field does not generate an interrupt.	
		0	Stopped: Reset or Stop Rx Command Issued.
		1	Running: Fetching Rx Transfer Descriptor.
		2	Reserved
		3	Running: Waiting for Rx Packet
		4	Suspended: Rx Descriptor Unavailable
		5	Running: Closing Rx Descriptor
		6	TIME_STAMP Write State
		7	Running: Transferring Rx Packet Data from Rx Buffer to Host Memory
16 (R/W1C)	NIS	Normal Interrupt Summary. The value of the `EMAC_DMA0_STAT.NIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA0_STAT.TI`, `EMAC_DMA0_STAT.TU`, `EMAC_DMA0_STAT.RI`, and `EMAC_DMA0_STAT.ERI`. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes `EMAC_DMA0_STAT.NIS` to be set is cleared.	
15 (R/W1C)	AIS	Abnormal Interrupt Summary. The value of the `EMAC_DMA0_STAT.AIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA0_IEN.TSE`, `EMAC_DMA0_IEN.TJE`, `EMAC_DMA0_IEN.OVE`, `EMAC_DMA0_IEN.UNE`, `EMAC_DMA0_IEN.RUE`, `EMAC_DMA0_IEN.RSE`, `EMAC_DMA0_IEN.RWE`, `EMAC_DMA0_IEN.ETE`, and `EMAC_DMA0_IEN.FBE`. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit that causes `EMAC_DMA0_STAT.AIS` to be set is cleared.	
14 (R/W1C)	ERI	Early Receive Interrupt. The `EMAC_DMA0_STAT.ERI` bit indicates that the DMA had filled the first data buffer of the packet. The `EMAC_DMA0_STAT.RI` bit automatically clears this bit.	

Table 31-71: EMAC_DMA0_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	FBI	Fatal Bus Error Interrupt. The `EMAC_DMA0_STAT.FBI` bit indicates that a bus error occurred, as detailed in the `EMAC_DMA0_STAT.EB` field. When this bit is set, the corresponding DMA engine disables all its bus accesses.
10 (R/W1C)	ETI	Early Transmit Interrupt. The `EMAC_DMA0_STAT.ETI` bit indicates that the frame to be transmitted was fully transferred to the MFL Transmit FIFO.
9 (R/W1C)	RWT	Receive WatchDog Timeout. The `EMAC_DMA0_STAT.RWT` bit is asserted when a frame with a length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).
8 (R/W1C)	RPS	Receive Process Stopped. The `EMAC_DMA0_STAT.RPS` bit is asserted when the Receive Process enters the Stopped state.
7 (R/W1C)	RU	Receive Buffer Unavailable. The `EMAC_DMA0_STAT.RU` bit indicates that the Next Descriptor in the Receive List is owned by the application and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the application should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor was owned by the DMA.
6 (R/W1C)	RI	Receive Interrupt. The `EMAC_DMA0_STAT.RI` bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.
5 (R/W1C)	UNF	Transmit Buffer Underflow. The `EMAC_DMA0_STAT.UNF` bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
4 (R/W1C)	OVF	Receive Buffer Overflow. The `EMAC_DMA0_STAT.OVF` bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].
3 (R/W1C)	TJT	Transmit Jabber Timeout. The `EMAC_DMA0_STAT.TJT` bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.

Table 31-71: EMAC_DMA0_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	TU	Transmit Buffer Unavailable. The `EMAC_DMA0_STAT.TU` bit indicates that the Next Descriptor in the Transmit List is owned by the application and cannot be acquired by the DMA. Transmission is suspended. The value in the `EMAC_DMA0_STAT.TS` bits explain the Transmit Process state transitions. To resume processing transmit descriptors, the application should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.
1 (R/W1C)	TPS	Transmit Process Stopped. The `EMAC_DMA0_STAT.TPS` bit is set when the transmission is stopped.
0 (R/W1C)	TI	Transmit Interrupt. The `EMAC_DMA0_STAT.TI` bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.

DMA Tx Buffer Current Register

The `EMAC_DMA0_TXBUF_CUR` register holds the pointer to the current transmit DMA buffer.

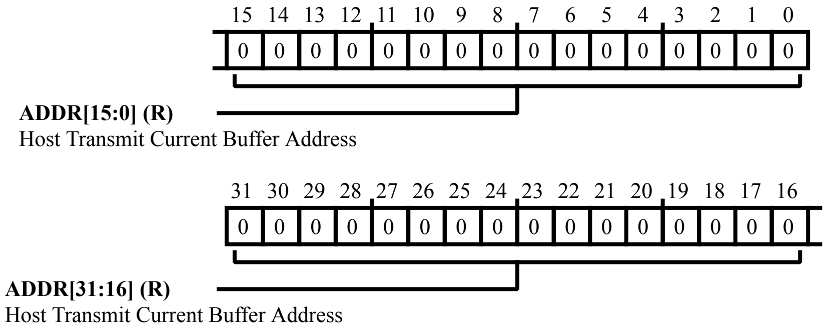

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Current Buffer Address

Figure 31-40: EMAC_DMA0_TXBUF_CUR Register Diagram

Table 31-72: EMAC_DMA0_TXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Current Buffer Address. The `EMAC_DMA0_TXBUF_CUR.ADDR` bit field points to the current Transmit Buffer Address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Descriptor List Address Register

The `EMAC_DMA0_TXDSC_ADDR` register holds the address for the DMA transmit descriptor list. The processor can write to this Register only when Tx DMA has stopped (`EMAC_DMA0_OPMODE.ST` bit =0). When stopped, this can be written with a new descriptor list address. When the processor sets the `EMAC_DMA0_OPMODE.ST` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA0_OPMODE.ST` bit is cleared to 0, then the DMA takes the descriptor address where it was stopped earlier.

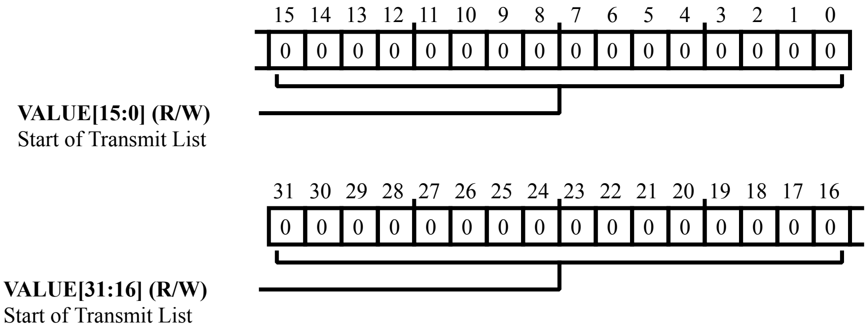

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Transmit List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Transmit List

Figure 31-41: EMAC_DMA0_TXDSC_ADDR Register Diagram

Table 31-73: EMAC_DMA0_TXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Transmit List. The `EMAC_DMA0_TXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1:0] for 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Tx Descriptor Current Register

The `EMAC_DMA0_TXDSC_CUR` register contains the current DMA transmit descriptor.

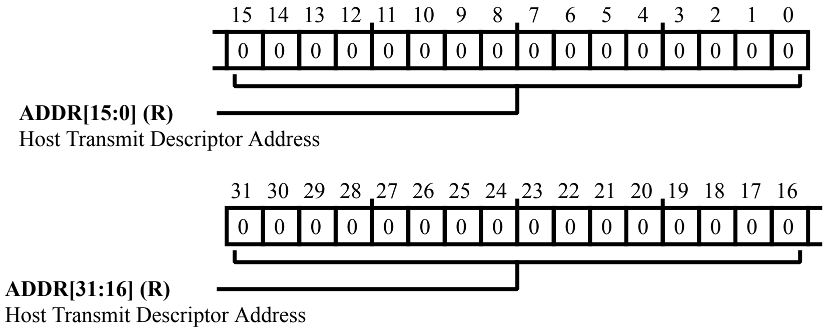

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Descriptor Address

Figure 31-42: EMAC_DMA0_TXDSC_CUR Register Diagram

Table 31-74: EMAC_DMA0_TXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Descriptor Address. The `EMAC_DMA0_TXDSC_CUR.ADDR` bit field points to the start address of the current Transmit Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Poll Demand Register

The EMAC_DMA0_TXPOLL register directs the EMAC to poll the transmit descriptor list.

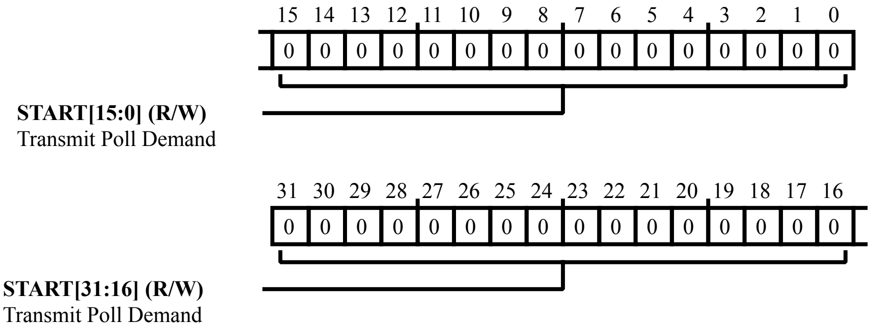

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Transmit Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Transmit Poll Demand

Figure 31-43: EMAC_DMA0_TXPOLL Register Diagram

Table 31-75: EMAC_DMA0_TXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Transmit Poll Demand. The EMAC_DMA0_TXPOLL.START bits, when written with any value, cause the DMA to read the current descriptor pointed to by EMAC_DMA0_TXDSC_CUR register. If that descriptor is not available (owned by application), transmission returns to the Suspend state, and the EMAC_DMA0_STAT.TU bit is asserted. If the descriptor is available, transmission resumes.

DMA Bus Mode Register

The `EMAC_DMA1_BUSMODE` register selects the DMA bus operating modes for EMAC DMA.

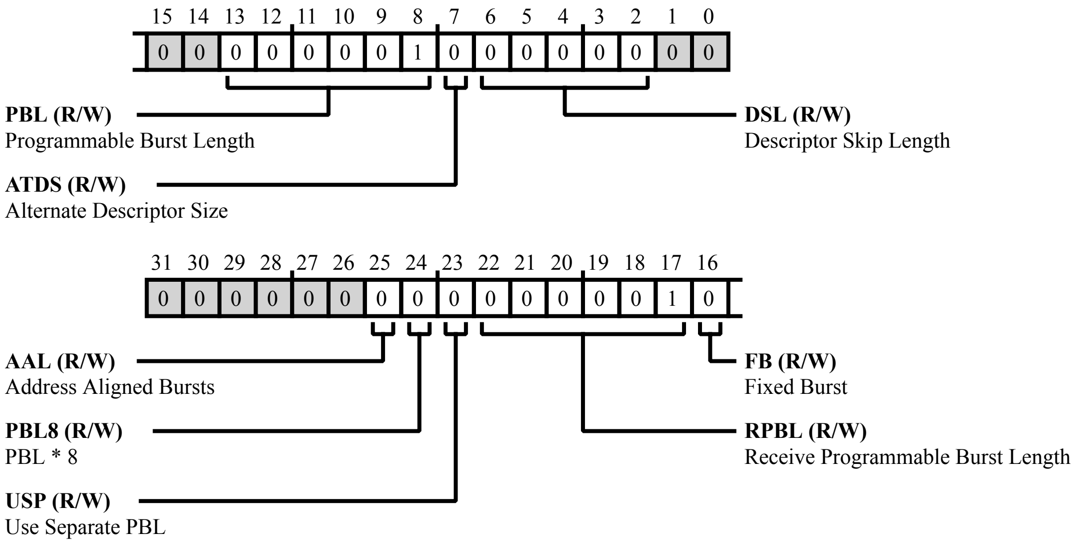

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
PBL (R/W)
Programmable Burst Length
ATDS (R/W)
Alternate Descriptor Size
DSL (R/W)
Descriptor Skip Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
AAL (R/W)
Address Aligned Bursts
PBL8 (R/W)
PBL * 8
USP (R/W)
Use Separate PBL
FB (R/W)
Fixed Burst
RPBL (R/W)
Receive Programmable Burst Length

Figure 31-44: EMAC_DMA1_BUSMODE Register Diagram

Table 31-76: EMAC_DMA1_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	AAL	Address Aligned Bursts. The `EMAC_DMA1_BUSMODE.AAL` bit, when set high and the FB bit equals 1, directs the SCB interface to generate all bursts aligned to the start address LS bits. If the FB bit is equal to 0, the first burst (accessing the data buffers start address) is not aligned, but subsequent bursts are aligned to the address.
24 (R/W)	PBL8	PBL * 8. The `EMAC_DMA1_BUSMODE.PBL8` bit, when set high, multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, and 32 beats depending on the PBL value.
23 (R/W)	USP	Use Separate PBL. The `EMAC_DMA1_BUSMODE.USP` bit, when set high, configures the Rx DMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to Tx DMA operations only.

Table 31-76: EMAC_DMA1_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22:17 (R/W)	RPBL	Receive Programmable Burst Length. The `EMAC_DMA1_BUSMODE.RPBL` bits indicate the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read/Write. The Rx DMA always attempts to burst as specified in RPBL every time it starts a Burst transfer on the host bus. RPBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. These bits are valid and applicable only when USP is set high.
16 (R/W)	FB	Fixed Burst. The `EMAC_DMA1_BUSMODE.FB` bit controls whether the SCB Master interface performs fixed burst transfers or not. See the `EMAC_DMA0_BMMODE.UNDEF` bit description for more information.
13:8 (R/W)	PBL	Programmable Burst Length. The `EMAC_DMA1_BUSMODE.PBL` bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read/Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the host bus. Any other value results in undefined behavior. When USP is set high, this PBL value is applicable for Tx DMA transactions only. PBL-max limit = (FIFO size / 2) / 4. PBL-max limit (transmit) = 256 bytes / 2 /4 = 32. PBL-max limit (receive) = 128 bytes / 2 /4 = 16. Note that this PBL is at the DMA end. If PBL= 32 and if BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If `EMAC_DMA1_BUSMODE.PBL` =8, and if `EMAC_DMA0_BMMODE.BLEN16` is enabled, the max burst is limited to `EMAC_DMA0_BMMODE.BLEN8`. If `EMAC_DMA1_BUSMODE.PBL8` bit is set, the programmed PBL value is multiplied by 8 times internally. However, the result cannot be more than the above maximum limits specified above.
7 (R/W)	ATDS	Alternate Descriptor Size. The `EMAC_DMA1_BUSMODE.ATDS` bit, when set, increases the size of the alternate descriptor to 32 bytes (8 DWORDS). This is required when the Advanced Time Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). The enhanced descriptor is not required if the Advanced Time Stamp and IPC Full Checksum Offload features are not enabled. In such case, you can use the 16 bytes descriptor to save 4 bytes of memory.

Table 31-76: EMAC_DMA1_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:2 (R/W)	DSL	Descriptor Skip Length. The `EMAC_DMA1_BUSMODE.DSL` bit specifies the number of 32-bit words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.

Channel 1 Credit Shaping Control Register

The `EMAC_DMA1_CHCBSCTL` register controls the credit-based shaper algorithm in the Traffic Manager for scheduling the frames for transmission. This register is present only when you select the Transmit Channel 1 in the AV mode.

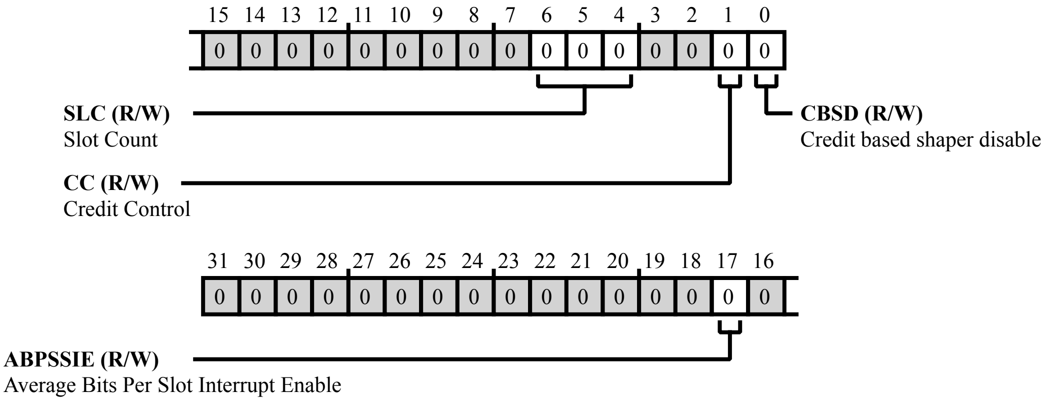

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLC (R/W)
Slot Count
CC (R/W)
Credit Control
CBSD (R/W)
Credit based shaper disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABPSSIE (R/W)
Average Bits Per Slot Interrupt Enable

Figure 31-45: EMAC_DMA1_CHCBSCTL Register Diagram

Table 31-77: EMAC_DMA1_CHCBSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	ABPSSIE	Average Bits Per Slot Interrupt Enable. When the `EMAC_DMA1_CHCBSCTL.ABPSSIE` bit is set, the MAC asserts an interrupt (sbd_intr_o or mci_intr_o) when the average bits per slot status is updated for Channel 1. When this bit is cleared, an interrupt is not asserted for such an event.
6:4 (R/W)	SLC	Slot Count. The `EMAC_DMA1_CHCBSCTL.SLC` bit field programs the number of slots (of duration 125 micro-sec) over which the average transmitted bits per slot (provided in the CBS Status register) are computed for Channel 1 when the credit-based shaper algorithm is enabled.
1 (R/W)	CC	Credit Control. The `EMAC_DMA1_CHCBSCTL.CC` bit, when reset, sets the accumulated credit parameter in the credit-based shaper algorithm logic to zero when there is positive credit and no frame to transmit in Channel 1.
0 (R/W)	CBSD	Credit based shaper disable. The `EMAC_DMA1_CHCBSCTL.CBSD` bit disables the credit-based shaper algorithm for Channel 1 traffic and makes the traffic management algorithm to strict priority for Channel 1 over Channel 0. When reset, the credit-based shaper algorithm schedules the traffic in Channel 1 for transmission.

Channel 1 Average Traffic Transmitted Register

The `EMAC_DMA1_CHCBSSTAT` register provides the average traffic transmitted in Channel 1. This register is present only when you select the Transmit Channel 1 in the AV mode.

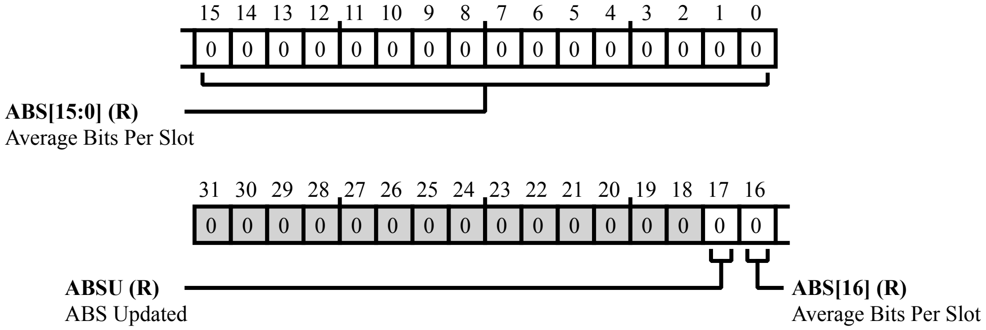

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABS[15:0] (R)
Average Bits Per Slot
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABSU (R)
ABS Updated
ABS[16] (R)
Average Bits Per Slot

Figure 31-46: EMAC_DMA1_CHCBSSTAT Register Diagram

Table 31-78: EMAC_DMA1_CHCBSSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	ABSU	ABS Updated. The `EMAC_DMA1_CHCBSSTAT.ABSU` bit indicates that the MAC has updated the ABS value. This bit is cleared when the application reads the ABS value.
16:0 (R/NW)	ABS	Average Bits Per Slot. The `EMAC_DMA1_CHCBSSTAT.ABS` bit field contains the average transmitted bits per slot. This field is computed over programmed number of slots (`EMAC_DMA1_CHCBSCTL.SLC` bit field) for Channel 1 traffic. The maximum value is 0x30D4 for 100 Mbps and 0x1E848 for 1000 Mbps.

Channel 1 High Credit Value Register

The `EMAC_DMA1_CHHIC` register provides the maximum value that can be accumulated for Channel 1 in the credit parameter of the credit-based shaper algorithm. This register is present only when you select the Transmit Channel 1 in the AV mode.

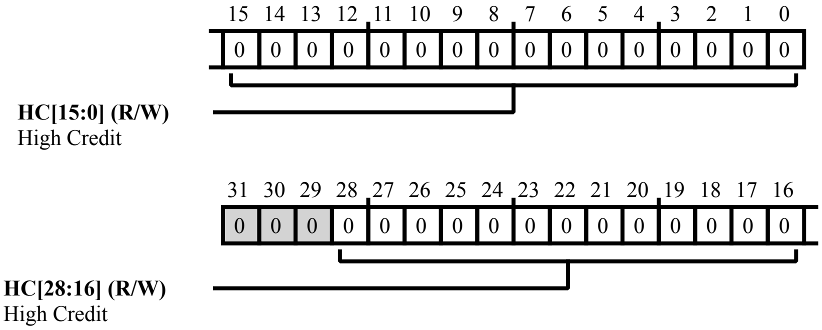

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HC[15:0] (R/W)
High Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HC[28:16] (R/W)
High Credit

Figure 31-47: EMAC_DMA1_CHHIC Register Diagram

Table 31-79: EMAC_DMA1_CHHIC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28:0 (R/W)	HC	High Credit. The `EMAC_DMA1_CHHIC.HC` bit field contains the hiCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 1 Idle Slope Credit Value Register

The `EMAC_DMA1_CHISC` register provides the bandwidth allocated for the AV traffic on Channel 1. This register is present only when you select the Transmit Channel 1 in the AV mode.

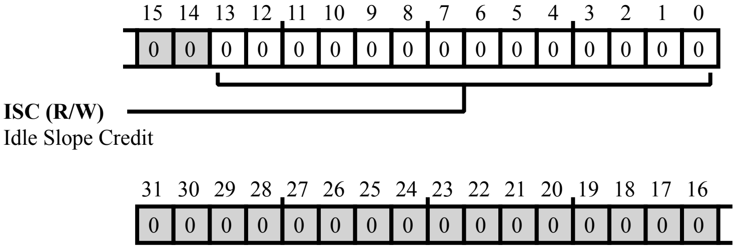

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ISC (R/W)
Idle Slope Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-48: EMAC_DMA1_CHISC Register Diagram

Table 31-80: EMAC_DMA1_CHISC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/W)	ISC	Idle Slope Credit. The `EMAC_DMA1_CHISC.ISC` bit field contains the idleSlopeCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 1 Low Credit Value Register

The `EMAC_DMA1_CHLOC` register provides the minimum value that can be accumulated for Channel 1 in the credit parameter of the credit-based shaper algorithm. This register is present only when you select Transmit Channel 1 in the AV mode.

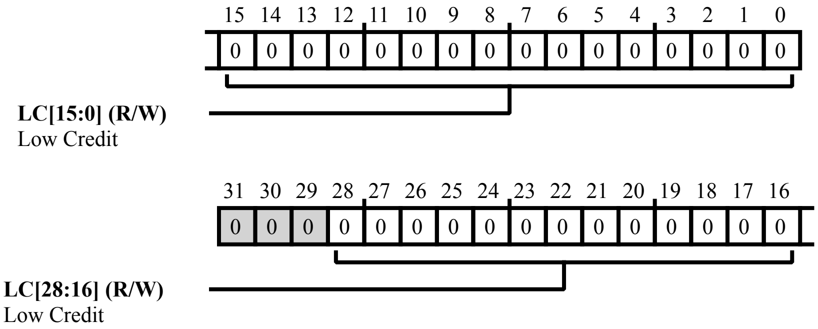

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LC[15:0] (R/W)
Low Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LC[28:16] (R/W)
Low Credit

Figure 31-49: EMAC_DMA1_CHLOC Register Diagram

Table 31-81: EMAC_DMA1_CHLOC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28:0 (R/W)	LC	Low Credit. The `EMAC_DMA1_CHLOC.LC` bit field contains the loCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 1 Control Bits for Slot Function Register

The `EMAC_DMA1_CHSFCS` register controls the slot comparison feature that the Channel 1 transmit DMA uses to fetch the buffer data from system memory.

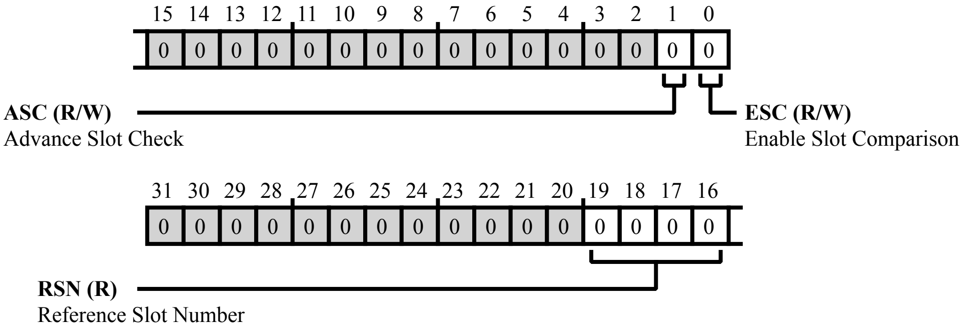

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ASC (R/W)
Advance Slot Check
ESC (R/W)
Enable Slot Comparison
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RSN (R)
Reference Slot Number

Figure 31-50: EMAC_DMA1_CHSFCS Register Diagram

Table 31-82: EMAC_DMA1_CHSFCS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/NW)	RSN	Reference Slot Number. The `EMAC_DMA1_CHSFCS.RSN` bits, gives the current value of the reference slot number in DMA used for comparison checking.
1 (R/W)	ASC	Advance Slot Check. The `EMAC_DMA1_CHSFCS.ASC` bits, When set, this bit enables the DMA to fetch the data from the buffer when the slot number (SLOTNUM) programmed in the transmit descriptor is ---equal to the reference slot number given in Bits [19:16] -or- ahead of the reference slot number by up to two slots. This bit is applicable only when Bit 0 (ESC) is set.
0 (R/W)	ESC	Enable Slot Comparison. The `EMAC_DMA1_CHSFCS.ESC` bits,When set, this bit enables the checking of the slot numbers, programmed in the transmit descriptor, with the current reference given in Bits [19:16]. The DMA fetches the data from the corresponding buffer only when the slot number is equal to the reference slot number or is ahead of the reference slot number by one slot. When reset, this bit disables the checking of the slot numbers. The DMA fetches the data immediately after the descriptor is processed.

Channel 1 Send Slope Credit Value Register

The EMAC_DMA1_CHSSC register provides the bandwidth that is available for the AV traffic on other channels. This register is present only when you select the Transmit Channel 1 in the AV mode.

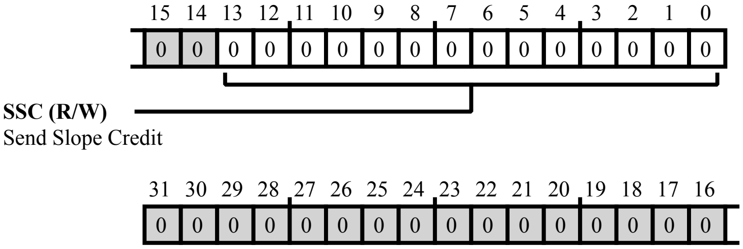

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SSC (R/W)
Send Slope Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-51: EMAC_DMA1_CHSSC Register Diagram

Table 31-83: EMAC_DMA1_CHSSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/W)	SSC	Send Slope Credit. The EMAC_DMA1_CHSSC.SSC bit field contains the sendSlopeCredit value required for credit-based shaper algorithm for Channel 1.

DMA Interrupt Enable Register

The EMAC_DMA1_IEN register enables (unmasks) EMAC DMA interrupts.

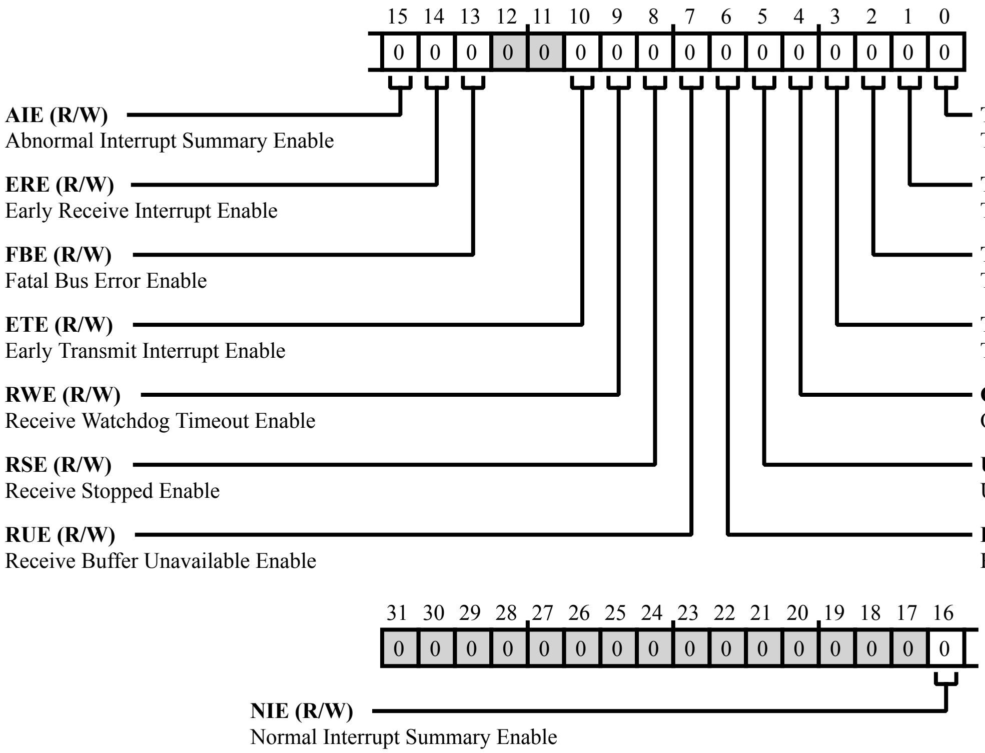

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIE (R/W)
Abnormal Interrupt Summary Enable
ERE (R/W)
Early Receive Interrupt Enable
FBE (R/W)
Fatal Bus Error Enable
ETE (R/W)
Early Transmit Interrupt Enable
RWE (R/W)
Receive Watchdog Timeout Enable
RSE (R/W)
Receive Stopped Enable
RUE (R/W)
Receive Buffer Unavailable Enable
TIE (R/W)
Transmit Interrupt Enable
TSE (R/W)
Transmit Stopped Enable
TUE (R/W)
Transmit Buffer Unavailable Enable
TJE (R/W)
Transmit Jabber Timeout Enable
OVE (R/W)
Overflow Interrupt Enable
UNE (R/W)
Underflow Interrupt Enable
RIE (R/W)
Receive Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NIE (R/W)
Normal Interrupt Summary Enable

Figure 31-52: EMAC_DMA1_IEN Register Diagram

Table 31-84: EMAC_DMA1_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	NIE	Normal Interrupt Summary Enable. The `EMAC_DMA1_IEN.NIE` bit, when set, enables a normal interrupt. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: `EMAC_DMA1_STAT.TI`, `EMAC_DMA1_STAT.TU`, `EMAC_DMA1_STAT.RI`, and `EMAC_DMA1_STAT.ERI`.
15 (R/W)	AIE	Abnormal Interrupt Summary Enable. The `EMAC_DMA1_IEN.AIE` bit, when set, enables an abnormal interrupt. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits: `EMAC_DMA1_STAT.TPS`, `EMAC_DMA1_STAT.TJT`, `EMAC_DMA1_STAT.OVF`, `EMAC_DMA1_STAT.RU`, `EMAC_DMA1_STAT.RPS`, `EMAC_DMA1_STAT.RWT`, `EMAC_DMA1_STAT.ETI`, and `EMAC_DMA1_STAT.FBI`.

Table 31-84: EMAC_DMA1_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	ERE	Early Receive Interrupt Enable. The `EMAC_DMA1_IEN.ERE` bit, when set (and with `EMAC_DMA1_IEN.NIE` =1), enables the Early Receive Interrupt. When this bit is reset, Early Receive Interrupt is disabled.
13 (R/W)	FBE	Fatal Bus Error Enable. The `EMAC_DMA1_IEN.FBE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Fatal Bus Error Interrupt. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.
10 (R/W)	ETE	Early Transmit Interrupt Enable. The `EMAC_DMA1_IEN.ETE` bit, when this bit is set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Early Transmit Interrupt. When this bit is reset, Early Transmit Interrupt is disabled.
9 (R/W)	RWE	Receive Watchdog Timeout Enable. The `EMAC_DMA1_IEN.RWE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Receive Watchdog Timeout Interrupt. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.
8 (R/W)	RSE	Receive Stopped Enable. The `EMAC_DMA1_IEN.RSE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.
7 (R/W)	RUE	Receive Buffer Unavailable Enable. The `EMAC_DMA1_IEN.RUE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Receive Buffer Unavailable Interrupt. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
6 (R/W)	RIE	Receive Interrupt Enable. The `EMAC_DMA1_IEN.RIE` bit, when set (and with `EMAC_DMA1_IEN.NIE` =1), enables the Receive Interrupt. When this bit is reset, Receive Interrupt is disabled.
5 (R/W)	UNE	Underflow Interrupt Enable. The `EMAC_DMA1_IEN.UNE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Transmit Underflow Interrupt. When this bit is reset, Underflow Interrupt is disabled.
4 (R/W)	OVE	Overflow Interrupt Enable. The `EMAC_DMA1_IEN.OVE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Receive Overflow Interrupt. When this bit is reset, Overflow Interrupt is disabled.

Table 31-84: EMAC_DMA1_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TJE	Transmit Jabber Timeout Enable. The `EMAC_DMA1_IEN.TJE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Transmit Jabber Timeout Interrupt. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.
2 (R/W)	TUE	Transmit Buffer Unavailable Enable. The `EMAC_DMA1_IEN.TUE` bit, when set (and with `EMAC_DMA1_IEN.NIE` =1), enables the Transmit Buffer Unavailable Interrupt. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.
1 (R/W)	TSE	Transmit Stopped Enable. The `EMAC_DMA1_IEN.TSE` bit, when set (and with `EMAC_DMA1_IEN.AIE` =1), enables the Transmission Stopped Interrupt. When this bit is reset, Transmission Stopped Interrupt is disabled.
0 (R/W)	TIE	Transmit Interrupt Enable. The `EMAC_DMA1_IEN.TIE` bit, when set (and with `EMAC_DMA1_IEN.NIE` =1), enables the Transmit Interrupt. When this bit is reset, Transmit Interrupt is disabled.

DMA Missed Frame Register

The EMAC_DMA1_MISS_FRM register contains counters for EMAC DMA missed frames and buffer overflows.

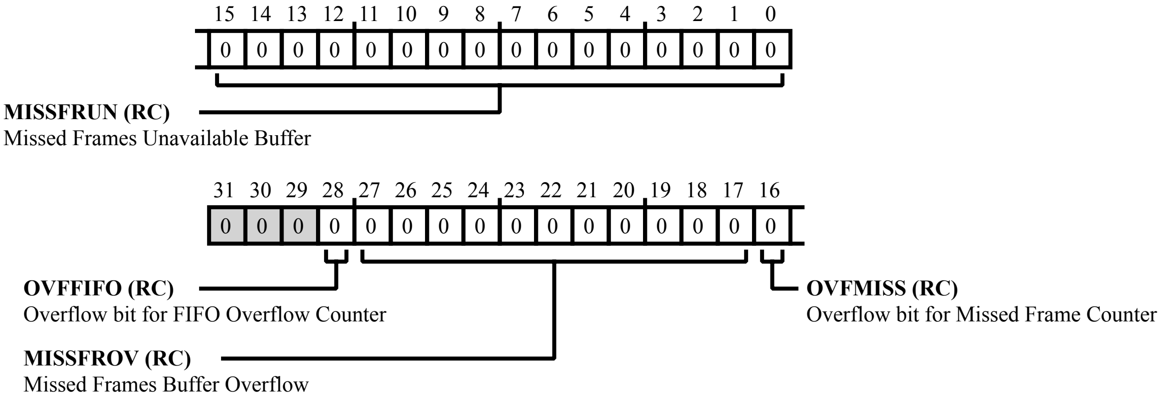

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISSFRUN (RC)
Missed Frames Unavailable Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OVFFIFO (RC)
Overflow bit for FIFO Overflow Counter
MISSFROV (RC)
Missed Frames Buffer Overflow
OVFMISS (RC)
Overflow bit for Missed Frame Counter

Figure 31-53: EMAC_DMA1_MISS_FRM Register Diagram

Table 31-85: EMAC_DMA1_MISS_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (RC/NW)	OVFFIFO	Overflow bit for FIFO Overflow Counter. The EMAC_DMA1_MISS_FRM.OVFFIFO bit holds the overflow bit for FIFO Overflow Counter.
27:17 (RC/NW)	MISSFROV	Missed Frames Buffer Overflow. The EMAC_DMA1_MISS_FRM.MISSFROV bits indicate the number of frames missed by the application due to buffer overflow.
16 (RC/NW)	OVFMISS	Overflow bit for Missed Frame Counter. The EMAC_DMA1_MISS_FRM.OVFMISS bit holds the overflow bit for the Missed Frame Counter.
15:0 (RC/NW)	MISSFRUN	Missed Frames Unavailable Buffer. The EMAC_DMA1_MISS_FRM.MISSFRUN bits indicate the number of frames missed by the controller because of the Application Receive Buffer being unavailable.

DMA Operation Mode Register

The `EMAC_DMA1_OPMODE` register selects receive and transmit DMA operating modes.

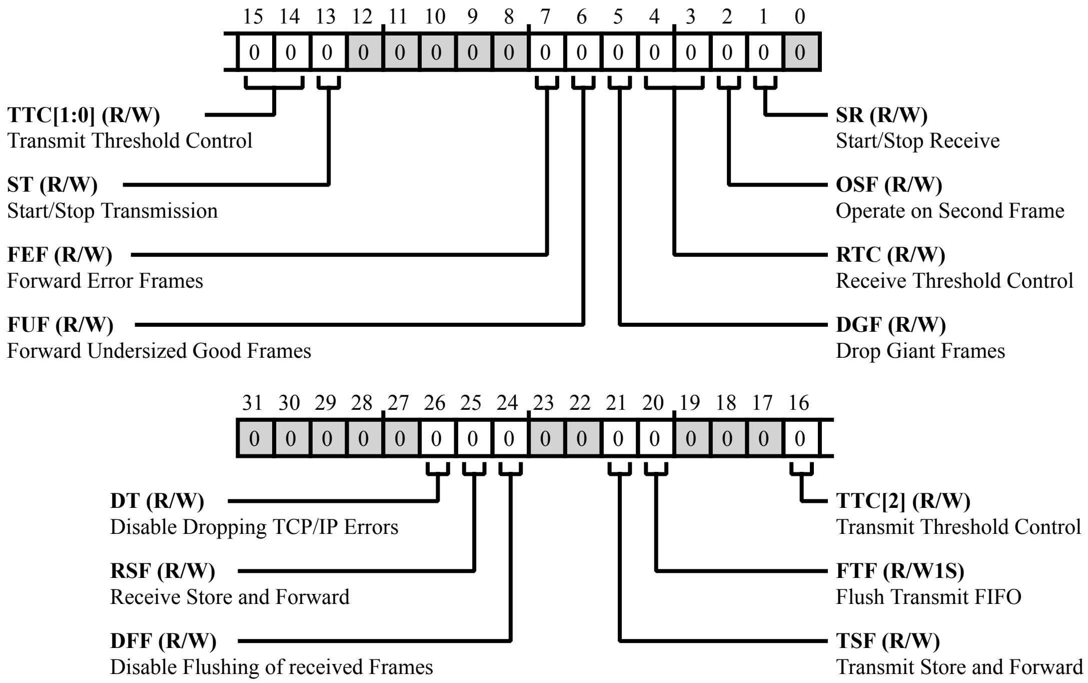

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TTC[1:0] (R/W)
Transmit Threshold Control
ST (R/W)
Start/Stop Transmission
FEF (R/W)
Forward Error Frames
FUF (R/W)
Forward Undersized Good Frames
SR (R/W)
Start/Stop Receive
OSF (R/W)
Operate on Second Frame
RTC (R/W)
Receive Threshold Control
DGF (R/W)
Drop Giant Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DT (R/W)
Disable Dropping TCP/IP Errors
RSF (R/W)
Receive Store and Forward
DFF (R/W)
Disable Flushing of received Frames
TTC[2] (R/W)
Transmit Threshold Control
FTF (R/W1S)
Flush Transmit FIFO
TSF (R/W)
Transmit Store and Forward

Figure 31-54: EMAC_DMA1_OPMODE Register Diagram

Table 31-86: EMAC_DMA1_OPMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	DT	Disable Dropping TCP/IP Errors. The `EMAC_DMA1_OPMODE.DT` bit, when set, directs the core not to drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the `EMAC_DMA1_OPMODE.FEF` bit is reset.
25 (R/W)	RSF	Receive Store and Forward. The `EMAC_DMA1_OPMODE.RSF` bit, when set, directs the MFL only to read a frame from the Rx FIFO after the complete frame has been written to it, ignoring the `EMAC_DMA1_OPMODE.RTC` bits. When this bit is reset, the Rx FIFO operates in threshold mode, subject to the threshold specified by the `EMAC_DMA1_OPMODE.RTC` bits.
24 (R/W)	DFF	Disable Flushing of received Frames. The `EMAC_DMA1_OPMODE.DFF` bit, when set, directs the Rx DMA not to flush any frames because of the unavailability of receive descriptors/buffers as it does normally when this bit is reset.

Table 31-86: EMAC_DMA1_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	TSF	Transmit Store and Forward. The `EMAC_DMA1_OPMODE.TSF` bit, when set, starts transmission when a full frame resides in the MFL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.	
20 (R/W1S)	FTF	Flush Transmit FIFO. The `EMAC_DMA1_OPMODE.FTF` bit, when set, directs the transmit FIFO controller logic to reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation completes only after emptying the Tx FIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock is required to be active. This field cleared to 1b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.	
16:14 (R/W)	TTC	Transmit Threshold Control. The `EMAC_DMA1_OPMODE.TTC` bits control the threshold level of the MFL Transmit FIFO. Transmission starts when the frame size within the MFL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the `EMAC_DMA1_OPMODE.TSF` bit is reset. The value =011 is not used.	
		0	64
		1	128
		2	192
		3	256
		4	40
		5	32
		6	24
		7	16

Table 31-86: EMAC_DMA1_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	ST	Start/Stop Transmission. The `EMAC_DMA1_OPMODE.ST` bit, when set, places transmission in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Transmit Descriptor List Address, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state, and the `EMAC_DMA1_STAT.TU` bit is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the `EMAC_DMA1_TXDSC_CUR` address register, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.
7 (R/W)	FEF	Forward Error Frames. The `EMAC_DMA1_OPMODE.FEF` bit, when reset, directs the Rx FIFO to drop frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frames start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. When `EMAC_DMA1_OPMODE.FEF` bit is set, all frames except runt error frames are forwarded to the DMA. But when Rx FIFO overflows when a partial frame is written, then such frames are dropped even when `EMAC_DMA1_OPMODE.FEF` is set.
6 (R/W)	FUF	Forward Undersized Good Frames. The `EMAC_DMA1_OPMODE.FUF` bit, when set, directs the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO drops all frames of less than 64 bytes, unless it is already transferred because of lower value of Receive Threshold (for example, `EMAC_DMA1_OPMODE.RTC` =01).
5 (R/W)	DGF	Drop Giant Frames. The `EMAC_DMA1_OPMODE.DGF` bit, when set, the MAC drops the received giant frames in the Rx FIFO, that is, frames that are larger than the computed giant frame limit. When reset, the MAC does not drop the giant frames in the Rx FIFO.

Table 31-86: EMAC_DMA1_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	RTC	Receive Threshold Control. The `EMAC_DMA1_OPMODE.RTC` bits control the threshold level of the MFL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MFL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. These bits are valid only when the `EMAC_DMA1_OPMODE.RSF` bit is zero, and are ignored when the `EMAC_DMA1_OPMODE.RSF` bit is set to 1. The value =11 is not used.	
		0	64
		1	32
		2	96
		3	128
2 (R/W)	OSF	Operate on Second Frame. The `EMAC_DMA1_OPMODE.OSF` bit, when set, instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.	
1 (R/W)	SR	Start/Stop Receive. The `EMAC_DMA1_OPMODE.SR` bit, when set, places the Receive process in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Receive Descriptor List Address or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended, and the `EMAC_DMA1_STAT.RU` bit is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting `EMAC_DMA1_RXDSC_CUR` address register, DMA behavior is unpredictable. When this bit is cleared, Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.	

DMA Rx Buffer Current Register

The EMAC_DMA1_RXBUF_CUR register holds the pointer to the current receive DMA buffer.

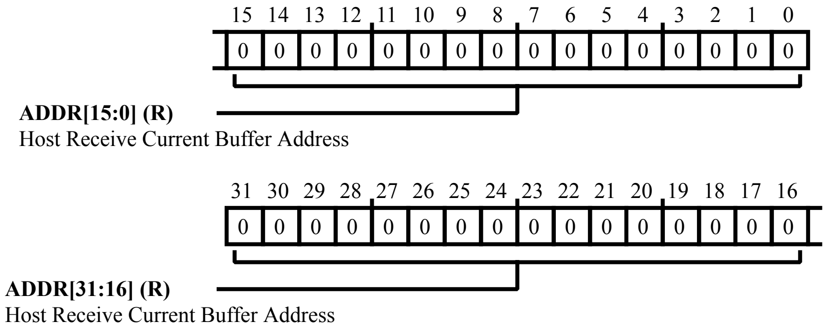

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Buffer Address

Figure 31-55: EMAC_DMA1_RXBUF_CUR Register Diagram

Table 31-87: EMAC_DMA1_RXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Buffer Address. The `EMAC_DMA1_RXBUF_CUR.ADDR` bit field points to the current Receive Buffer address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Descriptor List Address Register

The `EMAC_DMA1_RXDSC_ADDR` register holds the address for the DMA receive descriptor list. Writing to this Register is permitted only when reception is stopped. When stopped, this must be written to before the receive Start command is given. The processor can write to `EMAC_DMA1_RXDSC_ADDR` only when Rx DMA has stopped (`EMAC_DMA1_OPMODE.SR` bit =0). When stopped, it can be written with a new descriptor list address. When the processor sets the `EMAC_DMA1_OPMODE.SR` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA1_OPMODE.SR` bit is cleared to 0, the DMA takes the descriptor address where it was stopped earlier.

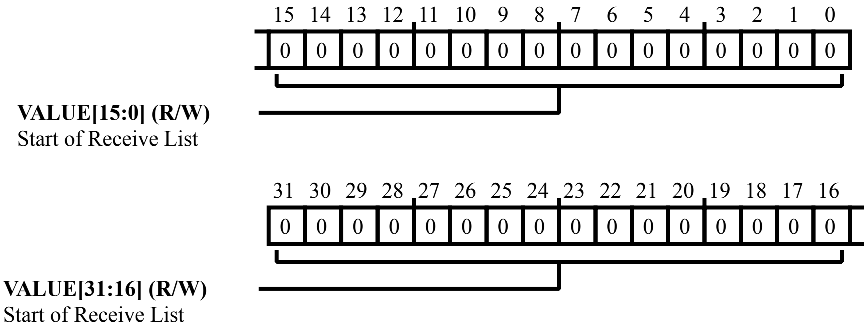

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Receive List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Receive List

Figure 31-56: EMAC_DMA1_RXDSC_ADDR Register Diagram

Table 31-88: EMAC_DMA1_RXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Receive List. The `EMAC_DMA1_RXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1:0] for the 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Rx Descriptor Current Register

The `EMAC_DMA1_RXDSC_CUR` register contains the current DMA receive descriptor.

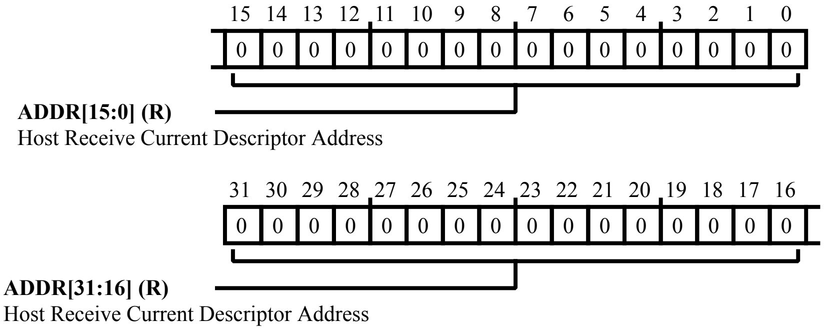

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Descriptor Address

Figure 31-57: EMAC_DMA1_RXDSC_CUR Register Diagram

Table 31-89: EMAC_DMA1_RXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Descriptor Address. The `EMAC_DMA1_RXDSC_CUR.ADDR` bit field points to the start address of the current Receive Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Interrupt Watch Dog Register

The `EMAC_DMA1_RXIWDOG` register contains the timeout value for the EMAC DMA receive interrupt watch dog timer.

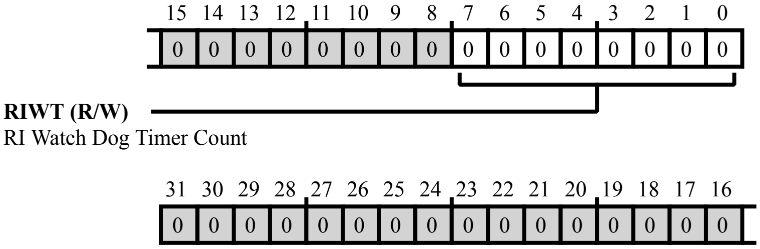

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RIWT (R/W)
RI Watch Dog Timer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-58: EMAC_DMA1_RXIWDOG Register Diagram

Table 31-90: EMAC_DMA1_RXIWDOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	RIWT	RI Watch Dog Timer Count. The `EMAC_DMA1_RXIWDOG.RIWT` bit field indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watchdog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when `EMAC_DMA1_STAT.RI` bit is set high because of automatic setting of `EMAC_DMA1_STAT.RI` as per RDES1[31] of any received frame.

DMA Rx Poll Demand Register

The EMAC_DMA1_RXPOLL register directs the EMAC to poll the receive descriptor list.

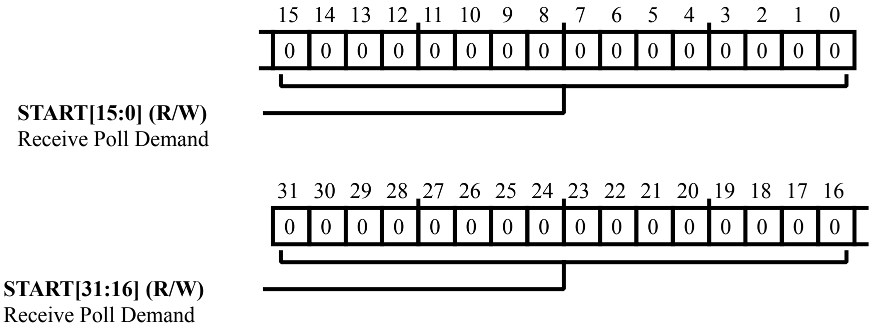

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Receive Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Receive Poll Demand

Figure 31-59: EMAC_DMA1_RXPOLL Register Diagram

Table 31-91: EMAC_DMA1_RXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Receive Poll Demand. The `EMAC_DMA1_RXPOLL.START` bits, when written with any value, cause the DMA to read the current descriptor pointed to by the `EMAC_DMA1_RXDSC_CUR` register. If that descriptor is not available (owned by application), reception returns to the Suspended state, and the `EMAC_DMA1_STAT.RU` bit is asserted. If the descriptor is available, the Receive DMA returns to the active state.

DMA Status Register

The `EMAC_DMA1_STAT` register indicates EMAC DMA status.

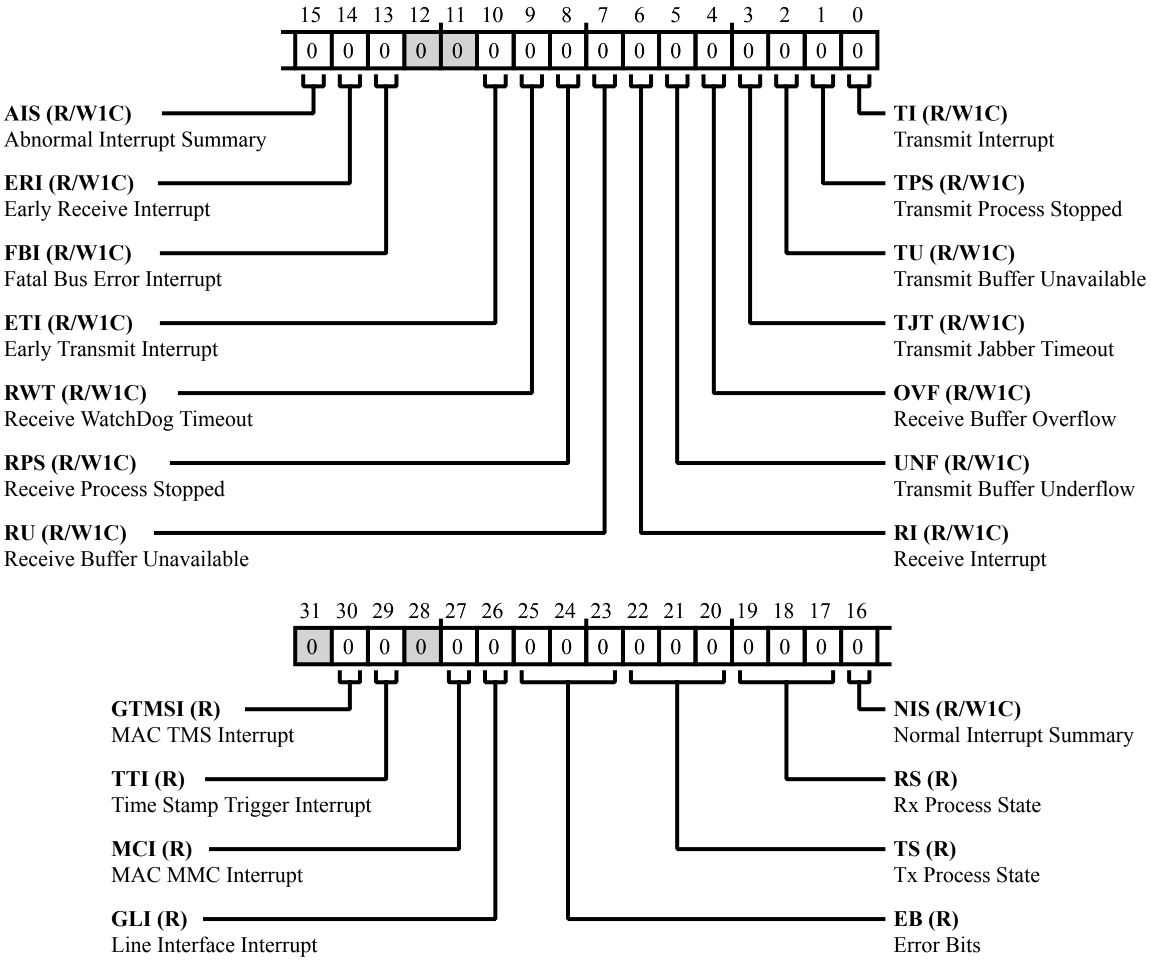

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIS (R/W1C)
Abnormal Interrupt Summary
ERI (R/W1C)
Early Receive Interrupt
FBI (R/W1C)
Fatal Bus Error Interrupt
ETI (R/W1C)
Early Transmit Interrupt
RWT (R/W1C)
Receive WatchDog Timeout
RPS (R/W1C)
Receive Process Stopped
RU (R/W1C)
Receive Buffer Unavailable
TI (R/W1C)
Transmit Interrupt
TPS (R/W1C)
Transmit Process Stopped
TU (R/W1C)
Transmit Buffer Unavailable
TJT (R/W1C)
Transmit Jabber Timeout
OVF (R/W1C)
Receive Buffer Overflow
UNF (R/W1C)
Transmit Buffer Underflow
RI (R/W1C)
Receive Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GTMSI (R)
MAC TMS Interrupt
TTI (R)
Time Stamp Trigger Interrupt
MCI (R)
MAC MMC Interrupt
GLI (R)
Line Interface Interrupt
NIS (R/W1C)
Normal Interrupt Summary
RS (R)
Rx Process State
TS (R)
Tx Process State
EB (R)
Error Bits

Figure 31-60: EMAC_DMA1_STAT Register Diagram

Table 31-92: EMAC_DMA1_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	GTMSI	MAC TMS Interrupt. MAC TMS Interrupt: The `EMAC_DMA1_STAT.GTMSI` bit indicates an interrupt event in the traffic manager and scheduler logic. To reset this bit, the software must read the corresponding registers (Channel Status Register) to get the exact cause of the interrupt and clear its source.

Table 31-92: EMAC_DMA1_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/NW)	TTI	Time Stamp Trigger Interrupt. The `EMAC_DMA1_STAT.TTI` bit indicates an interrupt event in the MAC core's Time Stamp Generator block. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to =0. When this bit is high, the interrupt signal from the MAC is high.	
27 (R/NW)	MCI	MAC MMC Interrupt. The `EMAC_DMA1_STAT.MCI` bit reflects an interrupt event in the MMC module of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as =0. The interrupt signal from the MAC is high when this bit is high.	
26 (R/NW)	GLI	Line Interface Interrupt. The `EMAC_DMA1_STAT.GLI` bit When set, this bit reflects any of the following interrupt events in the DWC_gmac interfaces	
25:23 (R/NW)	EB	Error Bits. The `EMAC_DMA1_STAT.EB` bits indicate the type of error that caused a Bus Error (for example, error response on the SCB interface). These bits are valid only when the `EMAC_DMA1_STAT.FBI` bit is set. This field does not generate an interrupt.	
		0	Error during data buffer access, write transfer, Rx DMA
		1	Error during data buffer access, write transfer, Tx DMA
		2	Error during data buffer access, read transfer, Rx DMA
		3	Error during data buffer access, read transfer, Tx DMA
		4	Error during descriptor access, write transfer, Rx DMA
		5	Error during descriptor access, write transfer, Tx DMA
		6	Error during descriptor access, read transfer, Rx DMA
		7	Error during descriptor access, read transfer, Tx DMA
22:20 (R/NW)	TS	Tx Process State. The `EMAC_DMA1_STAT.TS` bits indicate the transmit DMA state. This field does not generate an interrupt.	
		0	Stopped; Reset or Stop Tx Command Issued
		1	Running; Fetching Tx Transfer Descriptor
		2	Running; Waiting for Status
		3	Reading Data from Host Memory Buffer and Queuing It to Tx Buffer
		4	TIME_STAMP Write State

Table 31-92: EMAC_DMA1_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Reserved
		6	Suspended; Tx Descriptor Unavailable or Tx Buffer Underflow
		7	Closing Tx Descriptor
19:17 (R/NW)	RS	Rx Process State. The `EMAC_DMA1_STAT.RS` bits indicate the receive DMA state. This field does not generate an interrupt.	
		0	Stopped: Reset or Stop Rx Command Issued.
		1	Running: Fetching Rx Transfer Descriptor.
		2	Reserved
		3	Running: Waiting for Rx Packet
		4	Suspended: Rx Descriptor Unavailable
		5	Running: Closing Rx Descriptor
		6	TIME_STAMP Write State
		7	Running: Transferring Rx Packet Data from Rx Buffer to Host Memory
16 (R/W1C)	NIS	Normal Interrupt Summary. The value of the `EMAC_DMA1_STAT.NIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA1_STAT.TI`, `EMAC_DMA1_STAT.TU`, `EMAC_DMA1_STAT.RI`, and `EMAC_DMA1_STAT.ERI`. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes `EMAC_DMA1_STAT.NIS` to be set is cleared.	
15 (R/W1C)	AIS	Abnormal Interrupt Summary. The value of the `EMAC_DMA1_STAT.AIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA1_IEN.TSE`, `EMAC_DMA1_IEN.TJE`, `EMAC_DMA1_IEN.OVE`, `EMAC_DMA1_IEN.UNE`, `EMAC_DMA1_IEN.RUE`, `EMAC_DMA1_IEN.RSE`, `EMAC_DMA1_IEN.RWE`, `EMAC_DMA1_IEN.ETE`, and `EMAC_DMA1_IEN.FBE`. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit that causes `EMAC_DMA1_STAT.AIS` to be set is cleared.	
14 (R/W1C)	ERI	Early Receive Interrupt. The `EMAC_DMA1_STAT.ERI` bit indicates that the DMA had filled the first data buffer of the packet. The `EMAC_DMA1_STAT.RI` bit automatically clears this bit.	

Table 31-92: EMAC_DMA1_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	FBI	Fatal Bus Error Interrupt. The `EMAC_DMA1_STAT.FBI` bit indicates that a bus error occurred, as detailed in the `EMAC_DMA1_STAT.EB` field. When this bit is set, the corresponding DMA engine disables all its bus accesses.
10 (R/W1C)	ETI	Early Transmit Interrupt. The `EMAC_DMA1_STAT.ETI` bit indicates that the frame to be transmitted was fully transferred to the MFL Transmit FIFO.
9 (R/W1C)	RWT	Receive WatchDog Timeout. The `EMAC_DMA1_STAT.RWT` bit is asserted when a frame with a length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).
8 (R/W1C)	RPS	Receive Process Stopped. The `EMAC_DMA1_STAT.RPS` bit is asserted when the Receive Process enters the Stopped state.
7 (R/W1C)	RU	Receive Buffer Unavailable. The `EMAC_DMA1_STAT.RU` bit indicates that the Next Descriptor in the Receive List is owned by the application and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the application should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor was owned by the DMA.
6 (R/W1C)	RI	Receive Interrupt. The `EMAC_DMA1_STAT.RI` bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.
5 (R/W1C)	UNF	Transmit Buffer Underflow. The `EMAC_DMA1_STAT.UNF` bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
4 (R/W1C)	OVF	Receive Buffer Overflow. The `EMAC_DMA1_STAT.OVF` bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].
3 (R/W1C)	TJT	Transmit Jabber Timeout. The `EMAC_DMA1_STAT.TJT` bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.

Table 31-92: EMAC_DMA1_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	TU	Transmit Buffer Unavailable. The `EMAC_DMA1_STAT.TU` bit indicates that the Next Descriptor in the Transmit List is owned by the application and cannot be acquired by the DMA. Transmission is suspended. The value in the `EMAC_DMA1_STAT.TS` bits explain the Transmit Process state transitions. To resume processing transmit descriptors, the application should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.
1 (R/W1C)	TPS	Transmit Process Stopped. The `EMAC_DMA1_STAT.TPS` bit is set when the transmission is stopped.
0 (R/W1C)	TI	Transmit Interrupt. The `EMAC_DMA1_STAT.TI` bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.

DMA Tx Buffer Current Register

The `EMAC_DMA1_TXBUF_CUR` register holds the pointer to the current transmit DMA buffer.

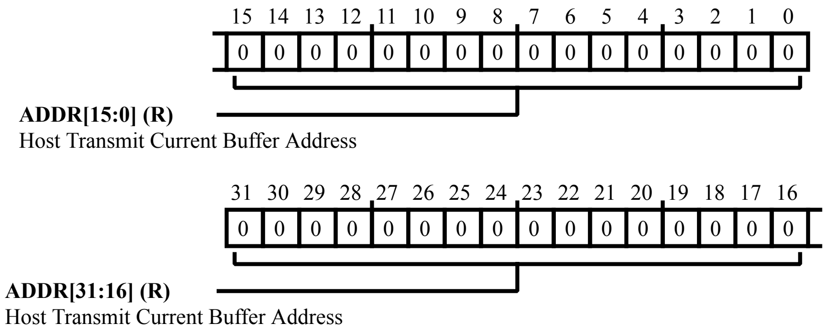

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Current Buffer Address

Figure 31-61: EMAC_DMA1_TXBUF_CUR Register Diagram

Table 31-93: EMAC_DMA1_TXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Current Buffer Address. The `EMAC_DMA1_TXBUF_CUR.ADDR` bit field points to the current Transmit Buffer Address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Descriptor List Address Register

The `EMAC_DMA1_TXDSC_ADDR` register holds the address for the DMA transmit descriptor list. The processor can write to this Register only when Tx DMA has stopped (`EMAC_DMA1_OPMODE.ST` bit =0). When stopped, this can be written with a new descriptor list address. When the processor sets the `EMAC_DMA1_OPMODE.ST` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA1_OPMODE.ST` bit is cleared to 0, then the DMA takes the descriptor address where it was stopped earlier.

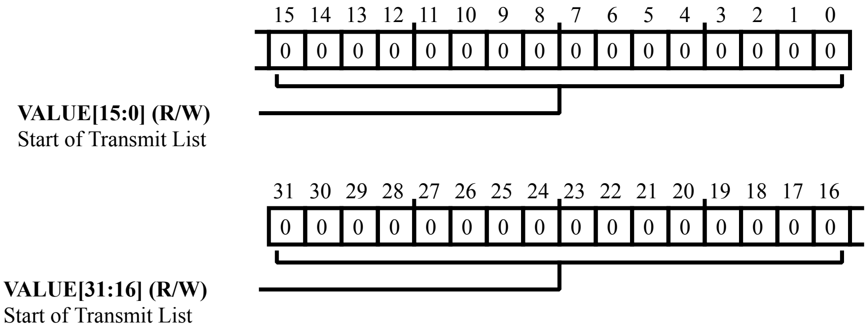

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Transmit List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Transmit List

Figure 31-62: EMAC_DMA1_TXDSC_ADDR Register Diagram

Table 31-94: EMAC_DMA1_TXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Transmit List. The `EMAC_DMA1_TXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1:0] for 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Tx Descriptor Current Register

The `EMAC_DMA1_TXDSC_CUR` register contains the current DMA transmit descriptor.

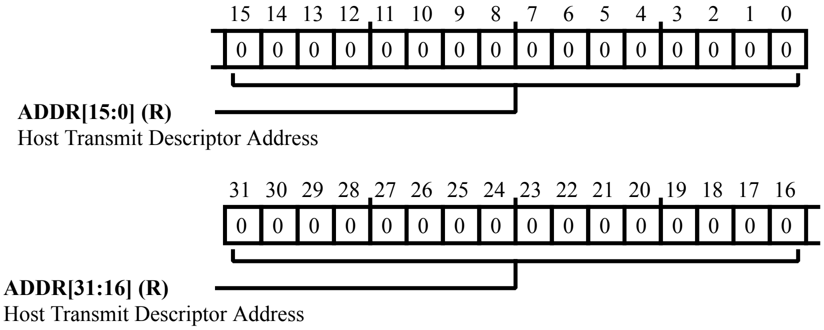

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Descriptor Address

Figure 31-63: EMAC_DMA1_TXDSC_CUR Register Diagram

Table 31-95: EMAC_DMA1_TXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Descriptor Address. The `EMAC_DMA1_TXDSC_CUR.ADDR` bit field points to the start address of the current Transmit Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Poll Demand Register

The `EMAC_DMA1_TXPOLL` register directs the EMAC to poll the transmit descriptor list.

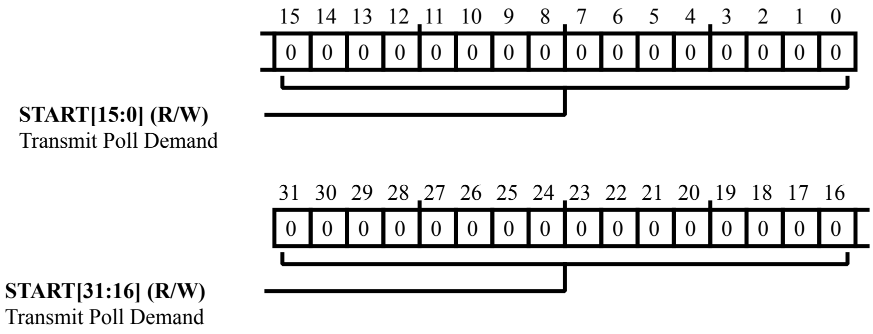

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Transmit Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Transmit Poll Demand

Figure 31-64: EMAC_DMA1_TXPOLL Register Diagram

Table 31-96: EMAC_DMA1_TXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Transmit Poll Demand. The `EMAC_DMA1_TXPOLL.START` bits, when written with any value, cause the DMA to read the current descriptor pointed to by `EMAC_DMA1_TXDSC_CUR` register. If that descriptor is not available (owned by application), transmission returns to the Suspend state, and the `EMAC_DMA1_STAT.TU` bit is asserted. If the descriptor is available, transmission resumes.

DMA Bus Mode Register

The `EMAC_DMA2_BUSMODE` register selects the DMA bus operating modes for EMAC DMA.

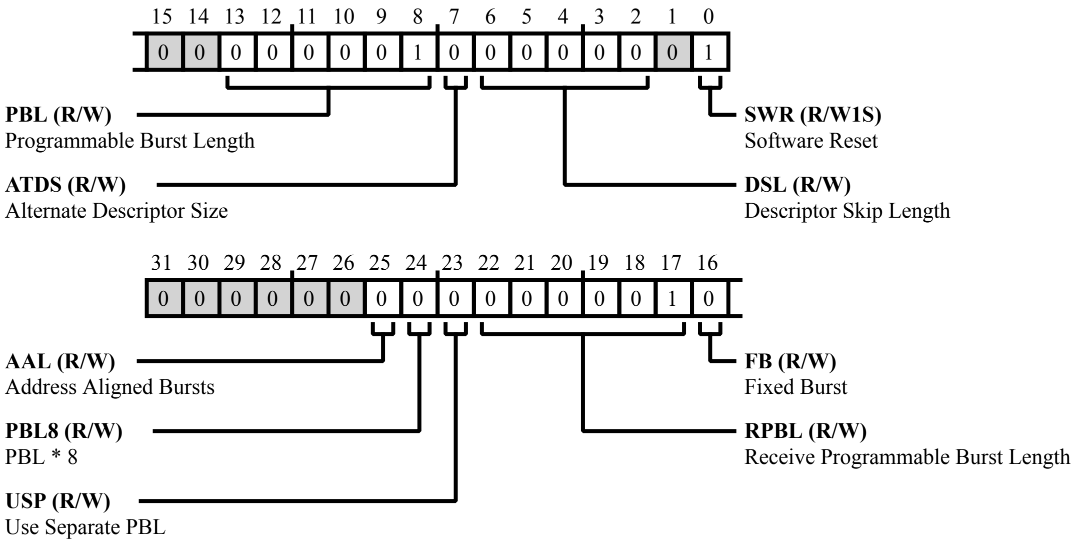

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
PBL (R/W)
Programmable Burst Length
ATDS (R/W)
Alternate Descriptor Size
SWR (R/W1S)
Software Reset
DSL (R/W)
Descriptor Skip Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
AAL (R/W)
Address Aligned Bursts
PBL8 (R/W)
PBL * 8
USP (R/W)
Use Separate PBL
FB (R/W)
Fixed Burst
RPBL (R/W)
Receive Programmable Burst Length

Figure 31-65: EMAC_DMA2_BUSMODE Register Diagram

Table 31-97: EMAC_DMA2_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	AAL	Address Aligned Bursts. The `EMAC_DMA2_BUSMODE.AAL` bit, when set high and the FB bit equals 1, directs the SCB interface to generate all bursts aligned to the start address LS bits. If the FB bit is equal to 0, the first burst (accessing the data buffers start address) is not aligned, but subsequent bursts are aligned to the address.
24 (R/W)	PBL8	PBL * 8. The `EMAC_DMA2_BUSMODE.PBL8` bit, when set high, multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, and 32 beats depending on the PBL value.
23 (R/W)	USP	Use Separate PBL. The `EMAC_DMA2_BUSMODE.USP` bit, when set high, configures the Rx DMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to Tx DMA operations only.

Table 31-97: EMAC_DMA2_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22:17 (R/W)	RPBL	Receive Programmable Burst Length. The `EMAC_DMA2_BUSMODE.RPBL` bits indicate the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read/Write. The Rx DMA always attempts to burst as specified in RPBL every time it starts a Burst transfer on the host bus. RPBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. These bits are valid and applicable only when USP is set high.
16 (R/W)	FB	Fixed Burst. The `EMAC_DMA2_BUSMODE.FB` bit controls whether the SCB Master interface performs fixed burst transfers or not. See the `EMAC_DMA0_BMMODE.UNDEF` bit description for more information.
13:8 (R/W)	PBL	Programmable Burst Length. The `EMAC_DMA2_BUSMODE.PBL` bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read/Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the host bus. Any other value results in undefined behavior. When USP is set high, this PBL value is applicable for Tx DMA transactions only. PBL-max limit = (FIFO size / 2) / 4. PBL-max limit (transmit) = 256 bytes / 2 /4 = 32. PBL-max limit (receive) = 128 bytes / 2 /4 = 16. Note that this PBL is at the DMA end. If PBL= 32 and if BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If `EMAC_DMA2_BUSMODE.PBL` =8, and if `EMAC_DMA0_BMMODE.BLEN16` is enabled, the max burst is limited to `EMAC_DMA0_BMMODE.BLEN8`. If `EMAC_DMA2_BUSMODE.PBL8` bit is set, the programmed PBL value is multiplied by 8 times internally. However, the result cannot be more than the above maximum limits specified above.
7 (R/W)	ATDS	Alternate Descriptor Size. The `EMAC_DMA2_BUSMODE.ATDS` bit, when set, increases the size of the alternate descriptor to 32 bytes (8 DWORDS). This is required when the Advanced Time Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). The enhanced descriptor is not required if the Advanced Time Stamp and IPC Full Checksum Offload features are not enabled. In such case, you can use the 16 bytes descriptor to save 4 bytes of memory.

Table 31-97: EMAC_DMA2_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:2 (R/W)	DSL	Descriptor Skip Length. The `EMAC_DMA2_BUSMODE.DSL` bit specifies the number of 32-bit words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.
0 (R/W1S)	SWR	Software Reset. The `EMAC_DMA2_BUSMODE.SWR` bit, when set, directs the MAC DMA Controller to reset all MAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in this bit before re-programming any register of the core. Note: The reset operation is completed only when all the resets in all the active clock domains are de-asserted. Therefore, it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for software reset completion. This field cleared to 1b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.

Channel 2 Credit Shaping Control Register

The `EMAC_DMA2_CHCBSCTL` register controls the credit-based shaper algorithm in the Traffic Manager for scheduling the frames for transmission. This register is present only when you select the Transmit Channel 1 in the AV mode.

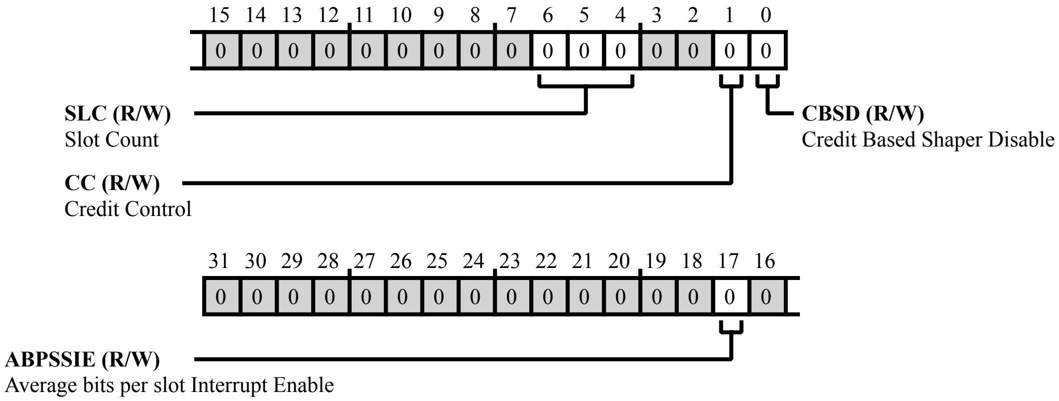

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLC (R/W)
Slot Count
CC (R/W)
Credit Control
CBSD (R/W)
Credit Based Shaper Disable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABPSSIE (R/W)
Average bits per slot Interrupt Enable

Figure 31-66: EMAC_DMA2_CHCBSCTL Register Diagram

Table 31-98: EMAC_DMA2_CHCBSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	ABPSSIE	Average bits per slot Interrupt Enable. The `EMAC_DMA2_CHCBSCTL.ABPSSIE` bit asserts an interrupt (sbd_intr_o or mci_intr_o) when the average bits per slot status is updated for Channel 1. When this bit is cleared, interrupt is not asserted for such an event.
6:4 (R/W)	SLC	Slot Count. The `EMAC_DMA2_CHCBSCTL.SLC` bit field programs the number of slots (of duration 125 micro-sec) over which the average transmitted bits per slot (provided in the CBS Status register) need to be computed for Channel 1 when the credit-based shaper algorithm is enabled. The
1 (R/W)	CC	Credit Control. The `EMAC_DMA2_CHCBSCTL.CC` bit, when reset, sets the accumulated credit parameter in the credit-based shaper algorithm logic to zero when there is positive credit and no frame to transmit in Channel 1.
0 (R/W)	CBSD	Credit Based Shaper Disable. The `EMAC_DMA2_CHCBSCTL.CBSD` bit disables the credit-based shaper algorithm for Channel 1 traffic and makes the traffic management algorithm to strict priority for Channel 1 over Channel 0. When reset, the credit-based shaper algorithm schedules the traffic in Channel 1 for transmission.

Channel 2 Avg Traffic Transmitted Status Register

The `EMAC_DMA2_CHCBSSTAT` register provides the average traffic transmitted in Channel 1. This register is present only when you select the Transmit Channel 1 in the AV mode.

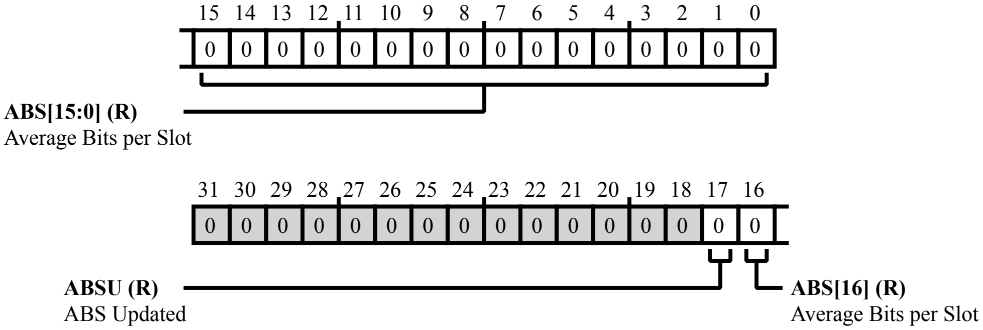

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABS[15:0] (R)
Average Bits per Slot
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ABSU (R)
ABS Updated
ABS[16] (R)
Average Bits per Slot

Figure 31-67: EMAC_DMA2_CHCBSSTAT Register Diagram

Table 31-99: EMAC_DMA2_CHCBSSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	ABSU	ABS Updated. The `EMAC_DMA2_CHCBSSTAT.ABSU` bits When set, this bit indicates that the MAC has updated the ABS value. This bit is cleared when the application reads the ABS value.
16:0 (R/NW)	ABS	Average Bits per Slot. The `EMAC_DMA2_CHCBSSTAT.ABS` bit field contains the average transmitted bits per slot. This field is computed over programmed number of slots (`EMAC_DMA2_CHCBSCTL.SLC` bit field) for Channel 1 traffic. The maximum value is 0x30D4 for 100 Mbps and 0x1E848 for 1000 Mbps.

Channel 2 High Credit Value Register

The `EMAC_DMA2_CHHIC` register provides the maximum value that can be accumulated for Channel 1 in the credit parameter of the credit-based shaper algorithm. This register is present only when you select the Transmit Channel 1 in the AV mode.

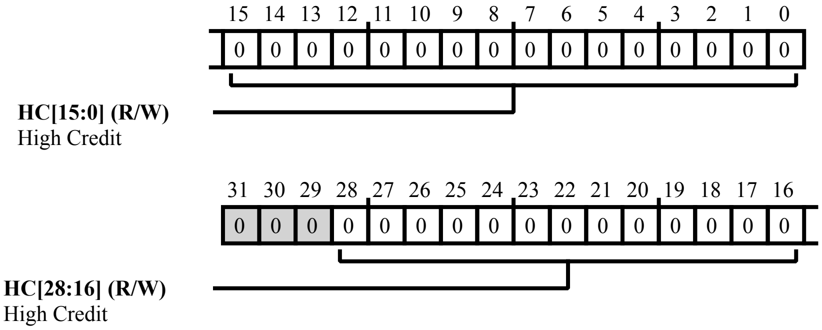

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HC[15:0] (R/W)
High Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HC[28:16] (R/W)
High Credit

Figure 31-68: EMAC_DMA2_CHHIC Register Diagram

Table 31-100: EMAC_DMA2_CHHIC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28:0 (R/W)	HC	High Credit. The `EMAC_DMA2_CHHIC.HC` bit field contains the hiCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 2 Idle Slope Credit Value Register

The `EMAC_DMA2_CHISC` register provides the bandwidth allocated for the AV traffic on Channel 1. This register is present only when you select the Transmit Channel 1 in the AV mode.

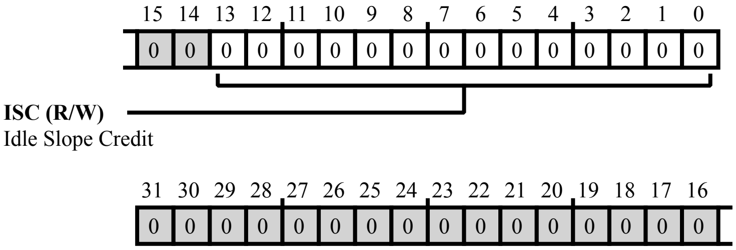

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ISC (R/W)
Idle Slope Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-69: EMAC_DMA2_CHISC Register Diagram

Table 31-101: EMAC_DMA2_CHISC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/W)	ISC	Idle Slope Credit. The `EMAC_DMA2_CHISC.ISC` bit field contains the idleSlopeCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 2 Low Credit Value Register

The `EMAC_DMA2_CHLOC` register provides the minimum value that can be accumulated for Channel 1 in the credit parameter of the credit-based shaper algorithm. This register is present only when you select Transmit Channel 1 in the AV mode.

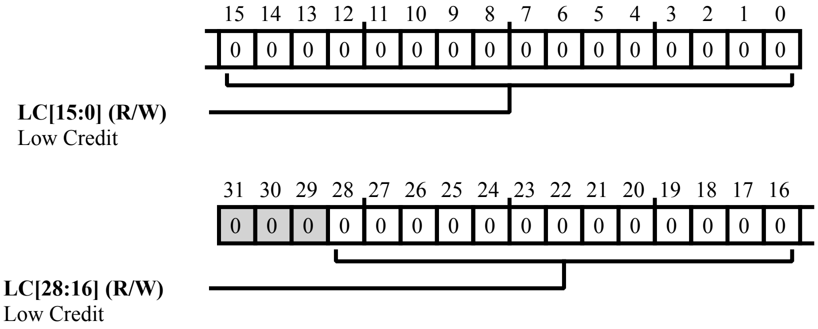

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LC[15:0] (R/W)
Low Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LC[28:16] (R/W)
Low Credit

Figure 31-70: EMAC_DMA2_CHLOC Register Diagram

Table 31-102: EMAC_DMA2_CHLOC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28:0 (R/W)	LC	Low Credit. The `EMAC_DMA2_CHLOC.LC` bit field contains the loCredit value required for the credit-based shaper algorithm for Channel 1.

Channel 2 Control Bits for Slot Function Register

The `EMAC_DMA2_CHSFCS` register controls the slot comparison feature that the Channel 1 transmit DMA uses to fetch the buffer data from system memory.

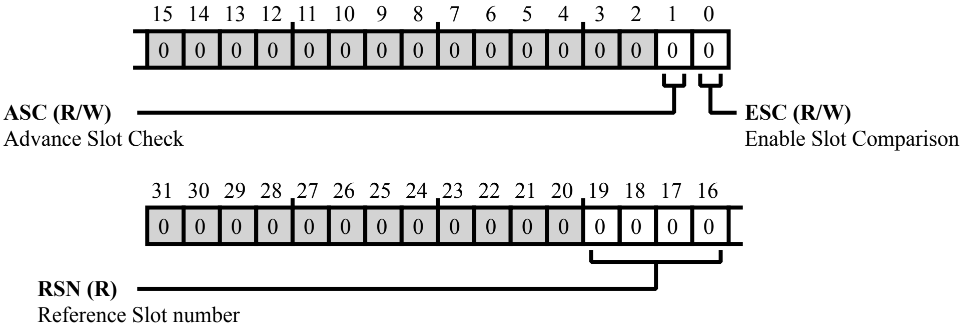

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ASC (R/W)
Advance Slot Check
ESC (R/W)
Enable Slot Comparison
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RSN (R)
Reference Slot number

Figure 31-71: EMAC_DMA2_CHSFCS Register Diagram

Table 31-103: EMAC_DMA2_CHSFCS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/NW)	RSN	Reference Slot number. The `EMAC_DMA2_CHSFCS.RSN` bits, gives the current value of the reference slot number in DMA used for comparison checking.
1 (R/W)	ASC	Advance Slot Check. The `EMAC_DMA2_CHSFCS.ASC` bits, When set, this bit enables the DMA to fetch the data from the buffer when the slot number (SLOTNUM) programmed in the transmit descriptor
0 (R/W)	ESC	Enable Slot Comparison. The `EMAC_DMA2_CHSFCS.ESC` bit enables the checking of the slot numbers, programmed in the transmit descriptor, with the current reference given in Bits [19:16]. The DMA fetches the data from the corresponding buffer only when the slot number is equal to the reference slot number or is ahead of the reference slot number by one slot. When reset, this bit disables the checking of the slot numbers. The DMA fetches the data immediately after the descriptor is processed.

Channel 2 Send Slope Credit Value Register

The `EMAC_DMA2_CHSSC` register provides the bandwidth that is available for the AV traffic on other channels. This register is present only when you select the Transmit Channel 1 in the AV mode.

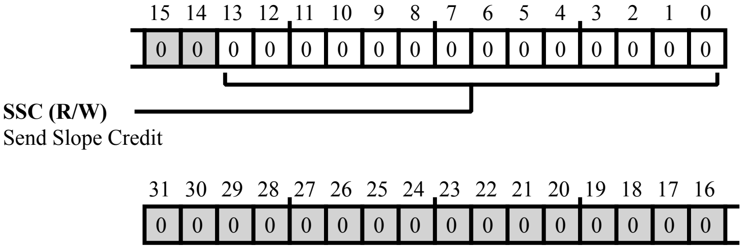

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SSC (R/W)
Send Slope Credit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-72: EMAC_DMA2_CHSSC Register Diagram

Table 31-104: EMAC_DMA2_CHSSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/W)	SSC	Send Slope Credit. The `EMAC_DMA2_CHSSC.SSC` bit field contains the sendSlopeCredit value required for credit-based shaper algorithm for Channel 1.

DMA Interrupt Enable Register

The `EMAC_DMA2_IEN` register enables (unmasks) EMAC DMA interrupts.

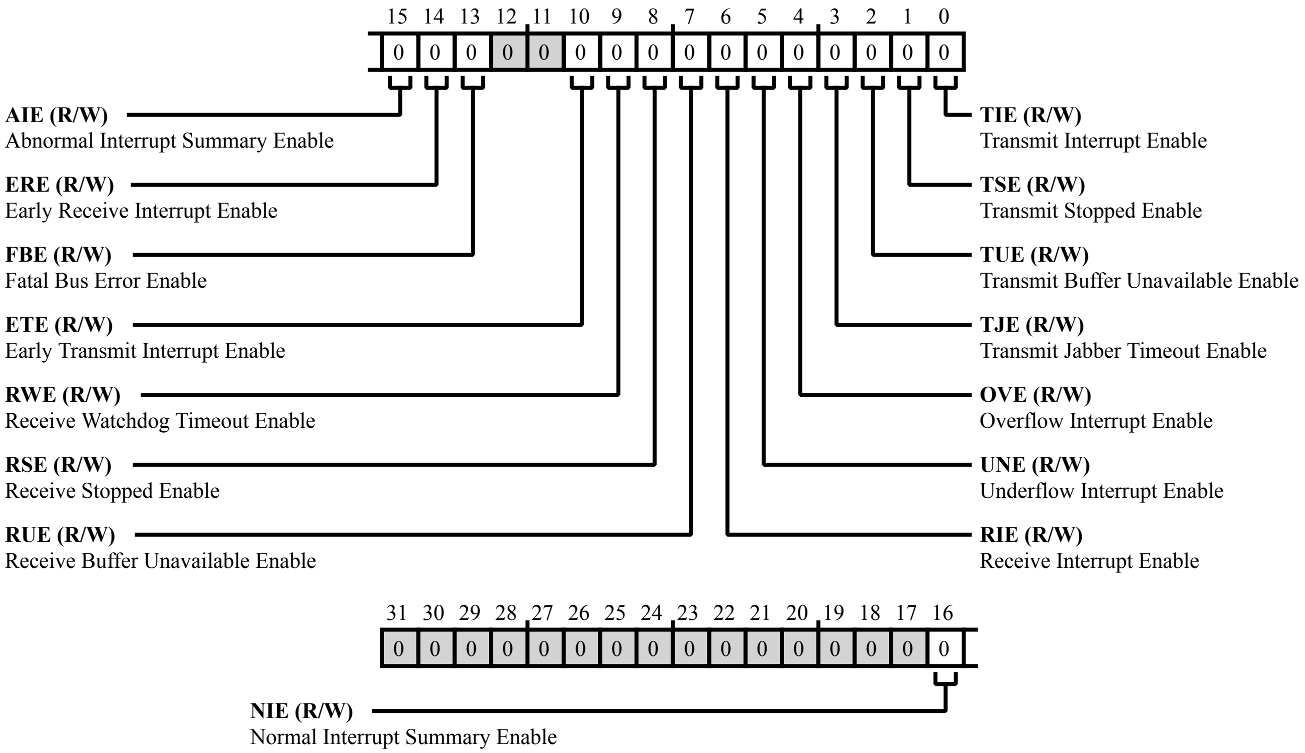

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIE (R/W)
Abnormal Interrupt Summary Enable
ERE (R/W)
Early Receive Interrupt Enable
FBE (R/W)
Fatal Bus Error Enable
ETE (R/W)
Early Transmit Interrupt Enable
RWE (R/W)
Receive Watchdog Timeout Enable
RSE (R/W)
Receive Stopped Enable
RUE (R/W)
Receive Buffer Unavailable Enable
TIE (R/W)
Transmit Interrupt Enable
TSE (R/W)
Transmit Stopped Enable
TUE (R/W)
Transmit Buffer Unavailable Enable
TJE (R/W)
Transmit Jabber Timeout Enable
OVE (R/W)
Overflow Interrupt Enable
UNE (R/W)
Underflow Interrupt Enable
RIE (R/W)
Receive Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NIE (R/W)
Normal Interrupt Summary Enable

Figure 31-73: EMAC_DMA2_IEN Register Diagram

Table 31-105: EMAC_DMA2_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	NIE	Normal Interrupt Summary Enable. The `EMAC_DMA2_IEN.NIE` bit, when set, enables a normal interrupt. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: `EMAC_DMA2_STAT.TI`, `EMAC_DMA2_STAT.TU`, `EMAC_DMA2_STAT.RI`, and `EMAC_DMA2_STAT.ERI`.
15 (R/W)	AIE	Abnormal Interrupt Summary Enable. The `EMAC_DMA2_IEN.AIE` bit, when set, enables an abnormal interrupt. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits: `EMAC_DMA2_STAT.TPS`, `EMAC_DMA2_STAT.TJT`, `EMAC_DMA2_STAT.OVF`, `EMAC_DMA2_STAT.RU`, `EMAC_DMA2_STAT.RPS`, `EMAC_DMA2_STAT.RWT`, `EMAC_DMA2_STAT.ETI`, and `EMAC_DMA2_STAT.FBI`.

Table 31-105: EMAC_DMA2_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	ERE	Early Receive Interrupt Enable. The `EMAC_DMA2_IEN.ERE` bit, when set (and with `EMAC_DMA2_IEN.NIE` =1), enables the Early Receive Interrupt. When this bit is reset, Early Receive Interrupt is disabled.
13 (R/W)	FBE	Fatal Bus Error Enable. The `EMAC_DMA2_IEN.FBE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Fatal Bus Error Interrupt. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.
10 (R/W)	ETE	Early Transmit Interrupt Enable. The `EMAC_DMA2_IEN.ETE` bit, when this bit is set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Early Transmit Interrupt. When this bit is reset, Early Transmit Interrupt is disabled.
9 (R/W)	RWE	Receive Watchdog Timeout Enable. The `EMAC_DMA2_IEN.RWE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Receive Watchdog Timeout Interrupt. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.
8 (R/W)	RSE	Receive Stopped Enable. The `EMAC_DMA2_IEN.RSE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.
7 (R/W)	RUE	Receive Buffer Unavailable Enable. The `EMAC_DMA2_IEN.RUE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Receive Buffer Unavailable Interrupt. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
6 (R/W)	RIE	Receive Interrupt Enable. The `EMAC_DMA2_IEN.RIE` bit, when set (and with `EMAC_DMA2_IEN.NIE` =1), enables the Receive Interrupt. When this bit is reset, Receive Interrupt is disabled.
5 (R/W)	UNE	Underflow Interrupt Enable. The `EMAC_DMA2_IEN.UNE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Transmit Underflow Interrupt. When this bit is reset, Underflow Interrupt is disabled.
4 (R/W)	OVE	Overflow Interrupt Enable. The `EMAC_DMA2_IEN.OVE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Receive Overflow Interrupt. When this bit is reset, Overflow Interrupt is disabled.

Table 31-105: EMAC_DMA2_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TJE	Transmit Jabber Timeout Enable. The `EMAC_DMA2_IEN.TJE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Transmit Jabber Timeout Interrupt. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.
2 (R/W)	TUE	Transmit Buffer Unavailable Enable. The `EMAC_DMA2_IEN.TUE` bit, when set (and with `EMAC_DMA2_IEN.NIE` =1), enables the Transmit Buffer Unavailable Interrupt. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.
1 (R/W)	TSE	Transmit Stopped Enable. The `EMAC_DMA2_IEN.TSE` bit, when set (and with `EMAC_DMA2_IEN.AIE` =1), enables the Transmission Stopped Interrupt. When this bit is reset, Transmission Stopped Interrupt is disabled.
0 (R/W)	TIE	Transmit Interrupt Enable. The `EMAC_DMA2_IEN.TIE` bit, when set (and with `EMAC_DMA2_IEN.NIE` =1), enables the Transmit Interrupt. When this bit is reset, Transmit Interrupt is disabled.

DMA Missed Frame Register

The EMAC_DMA2_MISS_FRM register contains counters for EMAC DMA missed frames and buffer overflows.

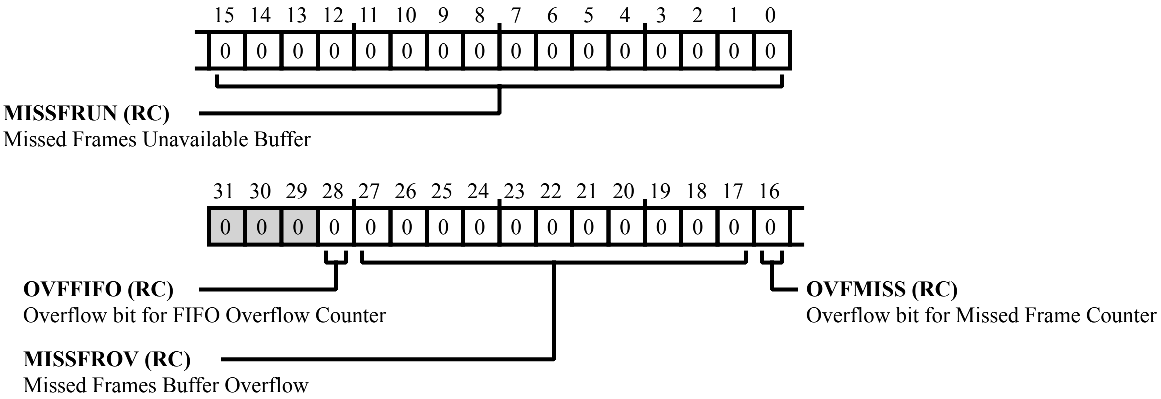

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISSFRUN (RC)
Missed Frames Unavailable Buffer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OVFFIFO (RC)
Overflow bit for FIFO Overflow Counter
OVFMISS (RC)
Overflow bit for Missed Frame Counter
MISSFROV (RC)
Missed Frames Buffer Overflow

Figure 31-74: EMAC_DMA2_MISS_FRM Register Diagram

Table 31-106: EMAC_DMA2_MISS_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (RC/NW)	OVFFIFO	Overflow bit for FIFO Overflow Counter. The EMAC_DMA2_MISS_FRM.OVFFIFO bit holds the overflow bit for FIFO Overflow Counter.
27:17 (RC/NW)	MISSFROV	Missed Frames Buffer Overflow. The EMAC_DMA2_MISS_FRM.MISSFROV bits indicate the number of frames missed by the application due to buffer overflow.
16 (RC/NW)	OVFMISS	Overflow bit for Missed Frame Counter. The EMAC_DMA2_MISS_FRM.OVFMISS bit holds the overflow bit for the Missed Frame Counter.
15:0 (RC/NW)	MISSFRUN	Missed Frames Unavailable Buffer. The EMAC_DMA2_MISS_FRM.MISSFRUN bits indicate the number of frames missed by the controller because of the Application Receive Buffer being unavailable.

DMA Operation Mode Register

The `EMAC_DMA2_OPMODE` register selects receive and transmit DMA operating modes.

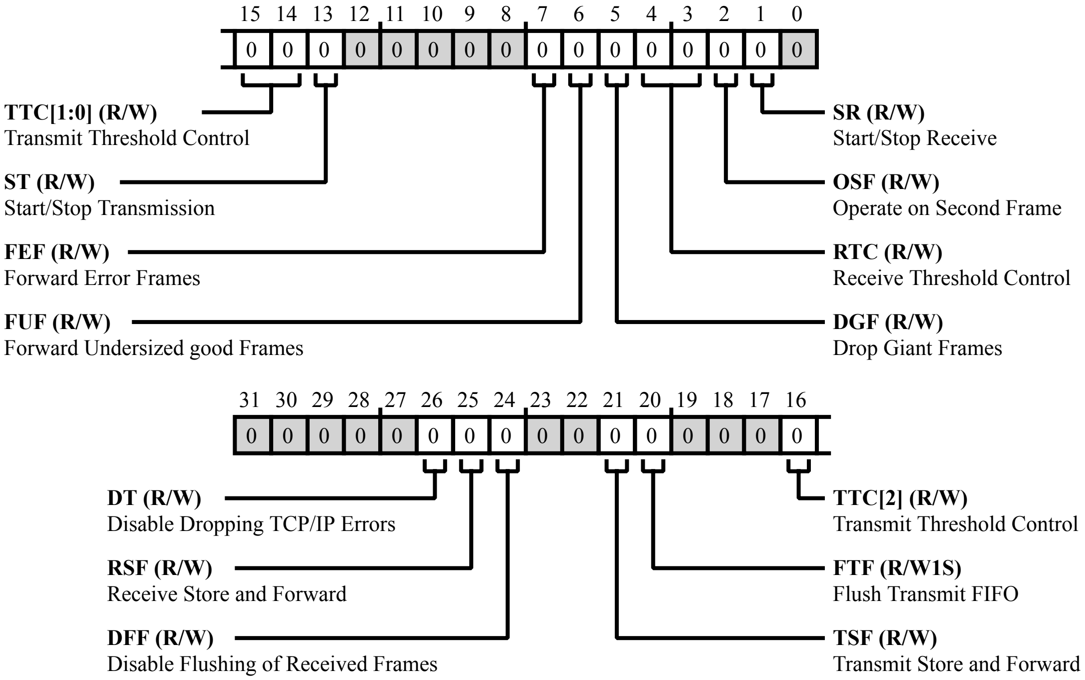

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TTC[1:0] (R/W)
Transmit Threshold Control
ST (R/W)
Start/Stop Transmission
FEF (R/W)
Forward Error Frames
FUF (R/W)
Forward Undersized good Frames
SR (R/W)
Start/Stop Receive
OSF (R/W)
Operate on Second Frame
RTC (R/W)
Receive Threshold Control
DGF (R/W)
Drop Giant Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DT (R/W)
Disable Dropping TCP/IP Errors
RSF (R/W)
Receive Store and Forward
DFF (R/W)
Disable Flushing of Received Frames
TTC[2] (R/W)
Transmit Threshold Control
FTF (R/W1S)
Flush Transmit FIFO
TSF (R/W)
Transmit Store and Forward

Figure 31-75: EMAC_DMA2_OPMODE Register Diagram

Table 31-107: EMAC_DMA2_OPMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	DT	Disable Dropping TCP/IP Errors. The `EMAC_DMA2_OPMODE.DT` bit, when set, directs the core not to drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the `EMAC_DMA2_OPMODE.FEF` bit is reset.
25 (R/W)	RSF	Receive Store and Forward. The `EMAC_DMA2_OPMODE.RSF` bit, when set, directs the MFL only to read a frame from the Rx FIFO after the complete frame has been written to it, ignoring the `EMAC_DMA2_OPMODE.RTC` bits. When this bit is reset, the Rx FIFO operates in threshold mode, subject to the threshold specified by the `EMAC_DMA2_OPMODE.RTC` bits.
24 (R/W)	DFF	Disable Flushing of Received Frames. The `EMAC_DMA2_OPMODE.DFF` bit, when set, directs the Rx DMA not to flush any frames because of the unavailability of receive descriptors/buffers as it does normally when this bit is reset.

Table 31-107: EMAC_DMA2_OPMODE Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>21
(R/W)</td><td>TSF</td><td colspan="2">Transmit Store and Forward.
The <code>EMAC_DMA2_OPMODE.TSF</code> bit, when set, starts transmission when a full frame resides in the MFL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.</td></tr>
<tr><td>20
(R/W1S)</td><td>FTF</td><td colspan="2">Flush Transmit FIFO.
The <code>EMAC_DMA2_OPMODE.FTF</code> bit, when set, directs the transmit FIFO controller logic to reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation completes only after emptying the Tx FIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock is required to be active. This field cleared to 1b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.</td></tr>
<tr><td rowspan="9">16:14
(R/W)</td><td rowspan="9">TTC</td><td colspan="2">Transmit Threshold Control.
The <code>EMAC_DMA2_OPMODE.TTC</code> bits control the threshold level of the MFL Transmit FIFO. Transmission starts when the frame size within the MFL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the <code>EMAC_DMA2_OPMODE.TSF</code> bit is reset. The value =011 is not used.</td></tr>
<tr><td>0</td><td>64</td></tr>
<tr><td>1</td><td>128</td></tr>
<tr><td>2</td><td>192</td></tr>
<tr><td>3</td><td>256</td></tr>
<tr><td>4</td><td>40</td></tr>
<tr><td>5</td><td>32</td></tr>
<tr><td>6</td><td>24</td></tr>
<tr><td>7</td><td>16</td></tr>
</table>

Table 31-107: EMAC_DMA2_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	ST	Start/Stop Transmission. The `EMAC_DMA2_OPMODE.ST` bit, when set, places transmission in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Transmit Descriptor List Address, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state, and the `EMAC_DMA2_STAT.TU` bit is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the `EMAC_DMA2_TXDSC_CUR` address register, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.
7 (R/W)	FEF	Forward Error Frames. The `EMAC_DMA2_OPMODE.FEF` bit, when reset, directs the Rx FIFO to drop frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frames start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. When `EMAC_DMA2_OPMODE.FEF` bit is set, all frames except runt error frames are forwarded to the DMA. But when Rx FIFO overflows when a partial frame is written, then such frames are dropped even when `EMAC_DMA2_OPMODE.FEF` is set.
6 (R/W)	FUF	Forward Undersized good Frames. The `EMAC_DMA2_OPMODE.FUF` bit, when set, directs the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO drops all frames of less than 64 bytes, unless it is already transferred because of lower value of Receive Threshold (for example, `EMAC_DMA2_OPMODE.RTC` =01).
5 (R/W)	DGF	Drop Giant Frames. The `EMAC_DMA2_OPMODE.DGF` bit, when set, the MAC drops the received giant frames in the Rx FIFO, that is, frames that are larger than the computed giant frame limit. When reset, the MAC does not drop the giant frames in the Rx FIFO.

Table 31-107: EMAC_DMA2_OPMODE Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="5">4:3
(R/W)</td><td rowspan="5">RTC</td><td colspan="2">Receive Threshold Control.
The <code>EMAC_DMA2_OPMODE.RTC</code> bits control the threshold level of the MFL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MFL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. These bits are valid only when the <code>EMAC_DMA2_OPMODE.RSF</code> bit is zero, and are ignored when the <code>EMAC_DMA2_OPMODE.RSF</code> bit is set to 1. The value =11 is not used.</td></tr>
<tr><td>0</td><td>64</td></tr>
<tr><td>1</td><td>32</td></tr>
<tr><td>2</td><td>96</td></tr>
<tr><td>3</td><td>128</td></tr>
<tr><td>2
(R/W)</td><td>OSF</td><td colspan="2">Operate on Second Frame.
The <code>EMAC_DMA2_OPMODE.OSF</code> bit, when set, instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.</td></tr>
<tr><td>1
(R/W)</td><td>SR</td><td colspan="2">Start/Stop Receive.
The <code>EMAC_DMA2_OPMODE.SR</code> bit, when set, places the Receive process in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Receive Descriptor List Address or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended, and the <code>EMAC_DMA2_STAT.RU</code> bit is set.
The Start Receive command is effective only when reception has stopped. If the command was issued before setting <code>EMAC_DMA2_RXDSC_CUR</code> address register, DMA behavior is unpredictable. When this bit is cleared, Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.</td></tr>
</table>

DMA Rx Buffer Current Register

The `EMAC_DMA2_RXBUF_CUR` register holds the pointer to the current receive DMA buffer.

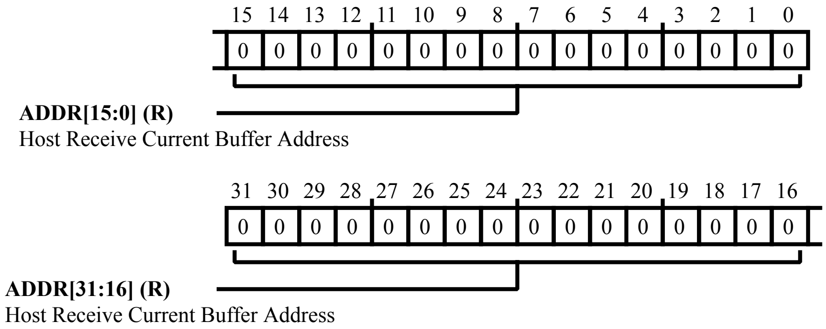

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Buffer Address

Figure 31-76: EMAC_DMA2_RXBUF_CUR Register Diagram

Table 31-108: EMAC_DMA2_RXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Buffer Address. The `EMAC_DMA2_RXBUF_CUR.ADDR` bit field points to the current Receive Buffer address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Descriptor List Address Register

The `EMAC_DMA2_RXDSC_ADDR` register holds the address for the DMA receive descriptor list. Writing to this Register is permitted only when reception is stopped. When stopped, this must be written to before the receive Start command is given. The processor can write to `EMAC_DMA2_RXDSC_ADDR` only when Rx DMA has stopped (`EMAC_DMA2_OPMODE.SR` bit =0). When stopped, it can be written with a new descriptor list address. When the processor sets the `EMAC_DMA2_OPMODE.SR` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA2_OPMODE.SR` bit is cleared to 0, the DMA takes the descriptor address where it was stopped earlier.

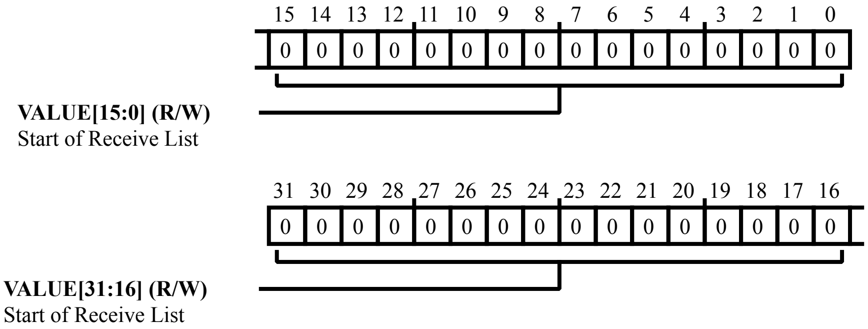

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Receive List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Receive List

Figure 31-77: EMAC_DMA2_RXDSC_ADDR Register Diagram

Table 31-109: EMAC_DMA2_RXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Receive List. The `EMAC_DMA2_RXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1:0] for the 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Rx Descriptor Current Register

The `EMAC_DMA2_RXDSC_CUR` register contains the current DMA receive descriptor.

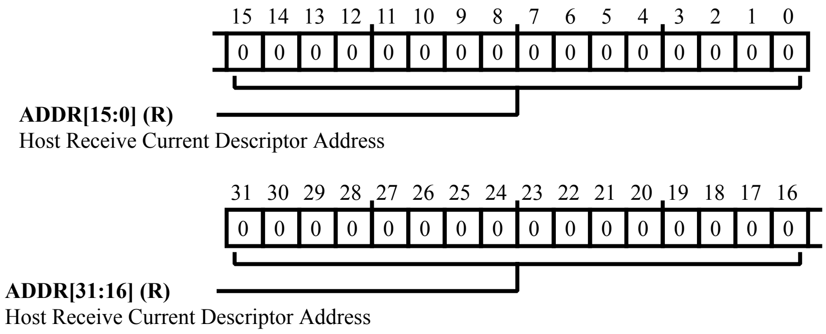

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Receive Current Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Receive Current Descriptor Address

Figure 31-78: EMAC_DMA2_RXDSC_CUR Register Diagram

Table 31-110: EMAC_DMA2_RXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Descriptor Address. The `EMAC_DMA2_RXDSC_CUR.ADDR` bit field points to the start address of the current Receive Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Interrupt Watch Dog Register

The `EMAC_DMA2_RXIWDOG` register contains the timeout value for the EMAC DMA receive interrupt watch dog timer.

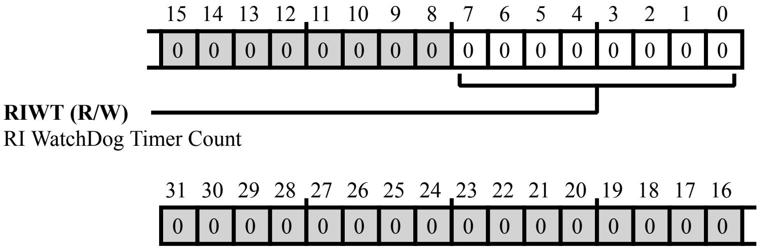

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RIWT (R/W)
RI WatchDog Timer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-79: EMAC_DMA2_RXIWDOG Register Diagram

Table 31-111: EMAC_DMA2_RXIWDOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	RIWT	RI WatchDog Timer Count. The `EMAC_DMA2_RXIWDOG.RIWT` bit field indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watch-dog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when `EMAC_DMA2_STAT.RI` bit is set high because of automatic setting of `EMAC_DMA2_STAT.RI` as per RDES1[31] of any received frame.

DMA Rx Poll Demand register

The EMAC_DMA2_RXPOLL register directs the EMAC to poll the receive descriptor list.

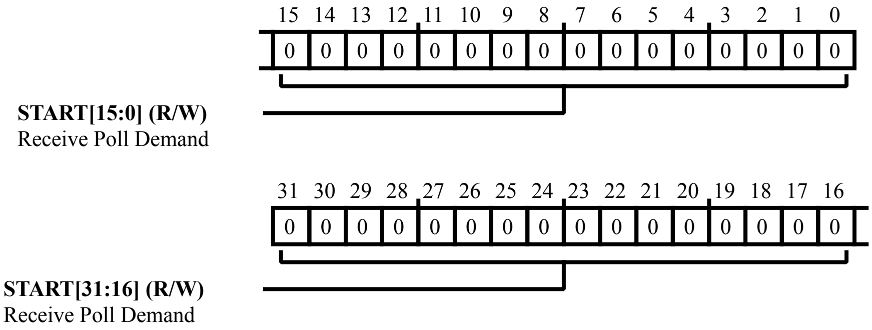

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Receive Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Receive Poll Demand

Figure 31-80: EMAC_DMA2_RXPOLL Register Diagram

Table 31-112: EMAC_DMA2_RXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Receive Poll Demand. The `EMAC_DMA2_RXPOLL.START` bits, when written with any value, cause the DMA to read the current descriptor pointed to by the `EMAC_DMA2_RXDSC_CUR` register. If that descriptor is not available (owned by application), reception returns to the Suspended state, and the `EMAC_DMA2_STAT.RU` bit is asserted. If the descriptor is available, the Receive DMA returns to the active state.

DMA Status Register

The `EMAC_DMA2_STAT` register indicates EMAC DMA status.

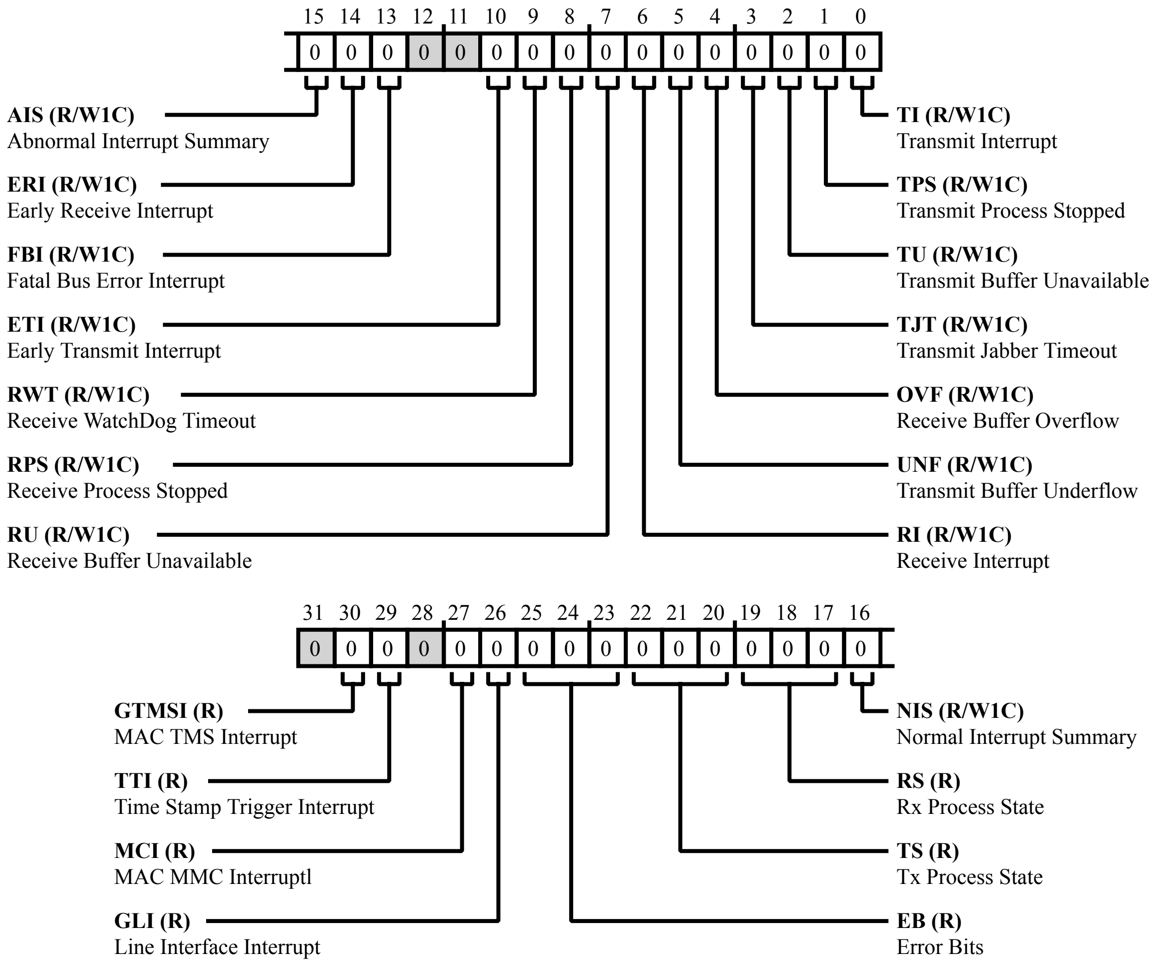

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AIS (R/W1C)
Abnormal Interrupt Summary
ERI (R/W1C)
Early Receive Interrupt
FBI (R/W1C)
Fatal Bus Error Interrupt
ETI (R/W1C)
Early Transmit Interrupt
RWT (R/W1C)
Receive WatchDog Timeout
RPS (R/W1C)
Receive Process Stopped
RU (R/W1C)
Receive Buffer Unavailable
TI (R/W1C)
Transmit Interrupt
TPS (R/W1C)
Transmit Process Stopped
TU (R/W1C)
Transmit Buffer Unavailable
TJT (R/W1C)
Transmit Jabber Timeout
OVF (R/W1C)
Receive Buffer Overflow
UNF (R/W1C)
Transmit Buffer Underflow
RI (R/W1C)
Receive Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GTMSI (R)
MAC TMS Interrupt
TTI (R)
Time Stamp Trigger Interrupt
MCI (R)
MAC MMC Interruptl
GLI (R)
Line Interface Interrupt
NIS (R/W1C)
Normal Interrupt Summary
RS (R)
Rx Process State
TS (R)
Tx Process State
EB (R)
Error Bits

Figure 31-81: EMAC_DMA2_STAT Register Diagram

Table 31-113: EMAC_DMA2_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	GTMSI	MAC TMS Interrupt. The `EMAC_DMA2_STAT.GTMSI` bit indicates an interrupt event in the traffic manager and scheduler logic. To reset this bit, the software must read the corresponding registers (Channel Status Register) to get the exact cause of the interrupt and clear its source.

Table 31-113: EMAC_DMA2_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/NW)	TTI	Time Stamp Trigger Interrupt. The `EMAC_DMA2_STAT.TTI` bit indicates an interrupt event in the MAC core's Time Stamp Generator block. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to =0. When this bit is high, the interrupt signal from the MAC is high.	
27 (R/NW)	MCI	MAC MMC Interruptl. The `EMAC_DMA2_STAT.MCI` bit reflects an interrupt event in the MMC module of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as =0. The interrupt signal from the MAC is high when this bit is high.	
26 (R/NW)	GLI	Line Interface Interrupt. The `EMAC_DMA2_STAT.GLI` bit When set, this bit reflects any of the following interrupt events in the DWC_gmac interfaces	
25:23 (R/NW)	EB	Error Bits. The `EMAC_DMA2_STAT.EB` bits indicate the type of error that caused a Bus Error (for example, error response on the SCB interface). These bits are valid only when the `EMAC_DMA2_STAT.FBI` bit is set. This field does not generate an interrupt.	
		0	Error during data buffer access, write transfer, Rx DMA
		1	Error during data buffer access, write transfer, Tx DMA
		2	Error during data buffer access, read transfer, Rx DMA
		3	Error during data buffer access, read transfer, Tx DMA
		4	Error during descriptor access, write transfer, Rx DMA
		5	Error during descriptor access, write transfer, Tx DMA
		6	Error during descriptor access, read transfer, Rx DMA
		7	Error during descriptor access, read transfer, Tx DMA
22:20 (R/NW)	TS	Tx Process State. The `EMAC_DMA2_STAT.TS` bits indicate the transmit DMA state. This field does not generate an interrupt.	
		0	Stopped; Reset or Stop Tx Command Issued
		1	Running; Fetching Tx Transfer Descriptor
		2	Running; Waiting for Status
		3	Reading Data from Host Memory Buffer and Queuing It to Tx Buffer
		4	TIME_STAMP Write State

Table 31-113: EMAC_DMA2_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Reserved
		6	Suspended; Tx Descriptor Unavailable or Tx Buffer Underflow
		7	Closing Tx Descriptor
19:17 (R/NW)	RS	Rx Process State. The `EMAC_DMA2_STAT.RS` bits indicate the receive DMA state. This field does not generate an interrupt.	
		0	Stopped: Reset or Stop Rx Command Issued.
		1	Running: Fetching Rx Transfer Descriptor.
		2	Reserved
		3	Running: Waiting for Rx Packet
		4	Suspended: Rx Descriptor Unavailable
		5	Running: Closing Rx Descriptor
		6	TIME_STAMP Write State
		7	Running: Transferring Rx Packet Data from Rx Buffer to Host Memory
16 (R/W1C)	NIS	Normal Interrupt Summary. The value of the `EMAC_DMA2_STAT.NIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA2_STAT.TI`, `EMAC_DMA2_STAT.TU`, `EMAC_DMA2_STAT.RI`, and `EMAC_DMA2_STAT.ERI`. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes `EMAC_DMA2_STAT.NIS` to be set is cleared.	
15 (R/W1C)	AIS	Abnormal Interrupt Summary. The value of the `EMAC_DMA2_STAT.AIS` bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: `EMAC_DMA2_IEN.TSE`, `EMAC_DMA2_IEN.TJE`, `EMAC_DMA2_IEN.OVE`, `EMAC_DMA2_IEN.UNE`, `EMAC_DMA2_IEN.RUE`, `EMAC_DMA2_IEN.RSE`, `EMAC_DMA2_IEN.RWE`, `EMAC_DMA2_IEN.ETE`, and `EMAC_DMA2_IEN.FBE`. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit that causes `EMAC_DMA2_STAT.AIS` to be set is cleared.	
14 (R/W1C)	ERI	Early Receive Interrupt. The `EMAC_DMA2_STAT.ERI` bit indicates that the DMA had filled the first data buffer of the packet. The `EMAC_DMA2_STAT.RI` bit automatically clears this bit.	

Table 31-113: EMAC_DMA2_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W1C)	FBI	Fatal Bus Error Interrupt. The `EMAC_DMA2_STAT.FBI` bit indicates that a bus error occurred, as detailed in the `EMAC_DMA2_STAT.EB` field. When this bit is set, the corresponding DMA engine disables all its bus accesses.
10 (R/W1C)	ETI	Early Transmit Interrupt. The `EMAC_DMA2_STAT.ETI` bit indicates that the frame to be transmitted was fully transferred to the MFL Transmit FIFO.
9 (R/W1C)	RWT	Receive WatchDog Timeout. The `EMAC_DMA2_STAT.RWT` bit is asserted when a frame with a length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).
8 (R/W1C)	RPS	Receive Process Stopped. The `EMAC_DMA2_STAT.RPS` bit is asserted when the Receive Process enters the Stopped state.
7 (R/W1C)	RU	Receive Buffer Unavailable. The `EMAC_DMA2_STAT.RU` bit indicates that the Next Descriptor in the Receive List is owned by the application and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the application should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor was owned by the DMA.
6 (R/W1C)	RI	Receive Interrupt. The `EMAC_DMA2_STAT.RI` bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.
5 (R/W1C)	UNF	Transmit Buffer Underflow. The `EMAC_DMA2_STAT.UNF` bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
4 (R/W1C)	OVF	Receive Buffer Overflow. The `EMAC_DMA2_STAT.OVF` bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].
3 (R/W1C)	TJT	Transmit Jabber Timeout. The `EMAC_DMA2_STAT.TJT` bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.

Table 31-113: EMAC_DMA2_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	TU	Transmit Buffer Unavailable. The `EMAC_DMA2_STAT.TU` bit indicates that the Next Descriptor in the Transmit List is owned by the application and cannot be acquired by the DMA. Transmission is suspended. The value in the `EMAC_DMA2_STAT.TS` bits explain the Transmit Process state transitions. To resume processing transmit descriptors, the application should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.
1 (R/W1C)	TPS	Transmit Process Stopped. The `EMAC_DMA2_STAT.TPS` bit is set when the transmission is stopped.
0 (R/W1C)	TI	Transmit Interrupt. The `EMAC_DMA2_STAT.TI` bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.

DMA Tx Buffer Current Register

The `EMAC_DMA2_TXBUF_CUR` register holds the pointer to the current transmit DMA buffer.

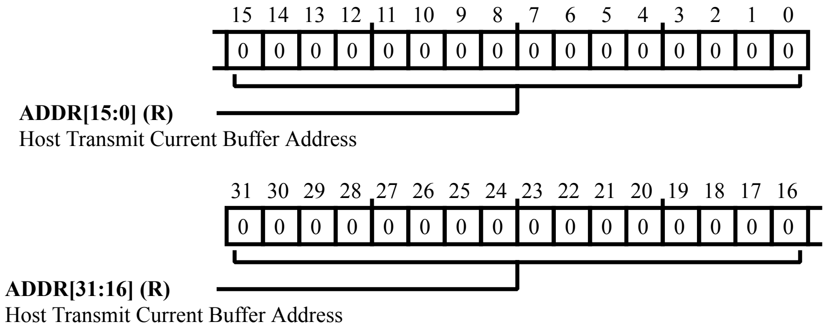

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Current Buffer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Current Buffer Address

Figure 31-82: EMAC_DMA2_TXBUF_CUR Register Diagram

Table 31-114: EMAC_DMA2_TXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Current Buffer Address. The `EMAC_DMA2_TXBUF_CUR.ADDR` bit field points to the current Transmit Buffer Address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Descriptor List Address Register

The `EMAC_DMA2_TXDSC_ADDR` register holds the address for the DMA transmit descriptor list. The processor can write to this Register only when Tx DMA has stopped (`EMAC_DMA2_OPMODE.ST` bit =0). When stopped, this can be written with a new descriptor list address. When the processor sets the `EMAC_DMA2_OPMODE.ST` bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the `EMAC_DMA2_OPMODE.ST` bit is cleared to 0, then the DMA takes the descriptor address where it was stopped earlier.

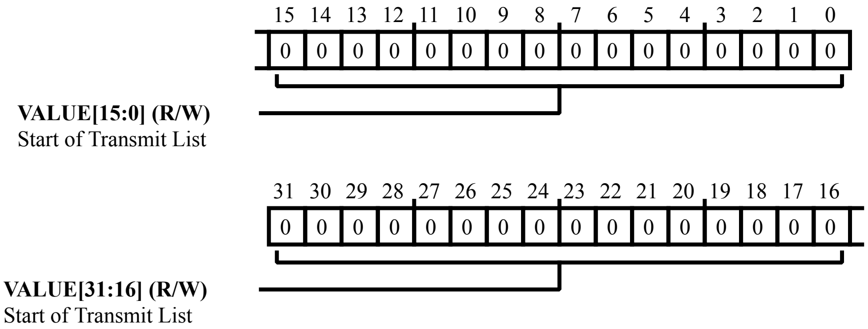

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Start of Transmit List
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Start of Transmit List

Figure 31-83: EMAC_DMA2_TXDSC_ADDR Register Diagram

Table 31-115: EMAC_DMA2_TXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Transmit List. The `EMAC_DMA2_TXDSC_ADDR.VALUE` bit field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1:0] for 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Tx Descriptor Current Register

The `EMAC_DMA2_TXDSC_CUR` register contains the current DMA transmit descriptor.

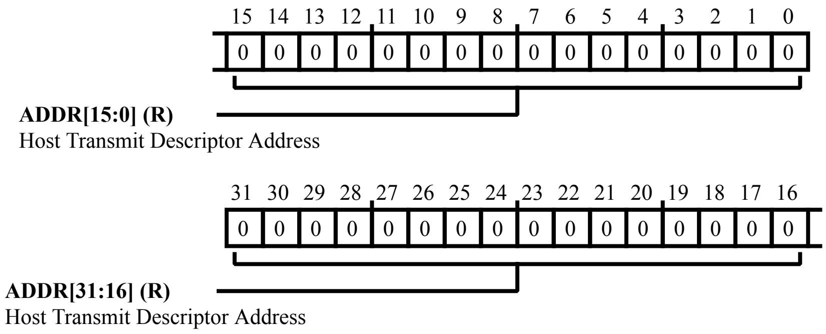

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R)
Host Transmit Descriptor Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R)
Host Transmit Descriptor Address

Figure 31-84: EMAC_DMA2_TXDSC_CUR Register Diagram

Table 31-116: EMAC_DMA2_TXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Descriptor Address. The `EMAC_DMA2_TXDSC_CUR.ADDR` bit field points to the start address of the current Transmit Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Poll Demand Register

The EMAC_DMA2_TXPOLL register directs the EMAC to poll the transmit descriptor list.

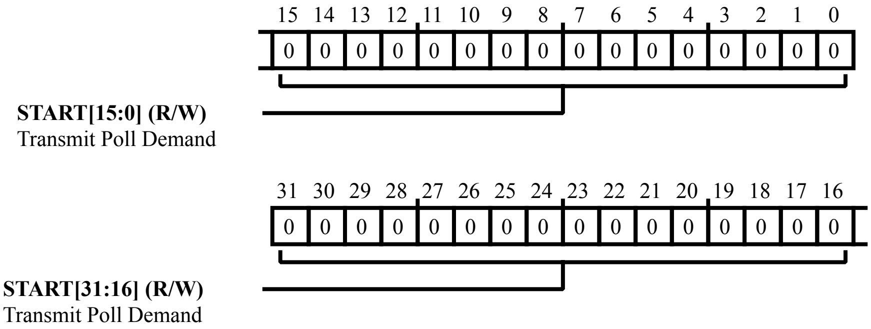

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[15:0] (R/W)
Transmit Poll Demand
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START[31:16] (R/W)
Transmit Poll Demand

Figure 31-85: EMAC_DMA2_TXPOLL Register Diagram

Table 31-117: EMAC_DMA2_TXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Transmit Poll Demand. The `EMAC_DMA2_TXPOLL.START` bits, when written with any value, cause the DMA to read the current descriptor pointed to by `EMAC_DMA2_TXDSC_CUR` register. If that descriptor is not available (owned by application), transmission returns to the Suspend state, and the `EMAC_DMA2_STAT.TU` bit is asserted. If the descriptor is available, transmission resumes.

FLow Control Register

The EMAC_FLOWCTL register controls EMAC flow control features.

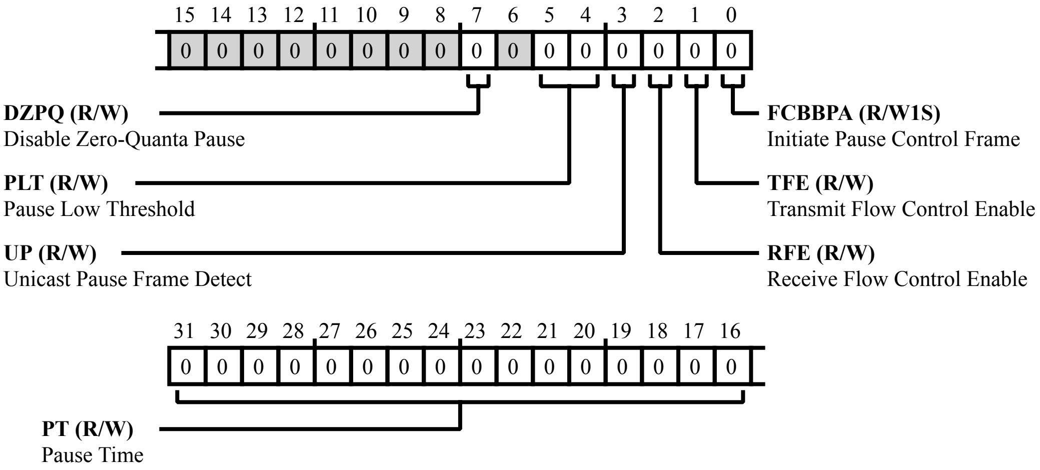

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DZPQ (R/W)
Disable Zero-Quanta Pause
PLT (R/W)
Pause Low Threshold
UP (R/W)
Unicast Pause Frame Detect
FCBBPA (R/W1S)
Initiate Pause Control Frame
TFE (R/W)
Transmit Flow Control Enable
RFE (R/W)
Receive Flow Control Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PT (R/W)
Pause Time

Figure 31-86: EMAC_FLOWCTL Register Diagram

Table 31-118: EMAC_FLOWCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	PT	Pause Time. The EMAC_FLOWCTL.PT bits hold the value to be used in the Pause Time field in the transmit control frame.
7 (R/W)	DZPQ	Disable Zero-Quanta Pause. The EMAC_FLOWCTL.DZPQ bit disables the automatic generation of the Zero-Quanta Pause frames on the de-assertion of the flow-control signal from the FIFO layer
5:4 (R/W)	PLT	Pause Low Threshold. The EMAC_FLOWCTL.PLT bit configures the threshold of the Pause timer at which the input flow control signal mti_flowctrl_i (or sbd_flowctrl_i) is checked for automatic retransmission of the Pause frame.
3 (R/W)	UP	Unicast Pause Frame Detect. The EMAC_FLOWCTL.UP bit, when set, directs the MAC to detect the Pause frames with the station's unicast address specified in EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers. This bit also directs the MAC to the detect Pause frames with the unique multicast address. When this bit is reset, the MAC will detect only a Pause frame with the unique multicast address specified in the 802.3x standard.

Table 31-118: EMAC_FLOWCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	RFE	Receive Flow Control Enable. The `EMAC_FLOWCTL.RFE` bit, when set, directs the MAC to decode the received Pause frame and disable its transmitter for a specified (Pause Time) time. When this bit is reset, the decode function of the Pause frame is disabled.
1 (R/W)	TFE	Transmit Flow Control Enable. In Full-Duplex mode, when the `EMAC_FLOWCTL.TFE` bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In Half-Duplex mode, when this bit is set, the MAC enables the back pressure operation. When this bit is reset, the back pressure feature is disabled.
0 (R/W1S)	FCBBPA	Initiate Pause Control Frame. The `EMAC_FLOWCTL.FCBBPA` bit initiates a Pause Control frame in Full-Duplex mode and activates the back pressure function in Half-Duplex mode if TFE bit is set. In Full-Duplex mode, this bit should be read as =0 before writing to the `EMAC_FLOWCTL` register. To initiate a Pause control frame, the Application must set this bit to =1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to =0. The `EMAC_FLOWCTL` register should not be written to until this bit is cleared. In Half-Duplex mode, when this bit is set (and `EMAC_FLOWCTL.TFE` is set), the back pressure is asserted by the MAC Core. During back pressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. The `EMAC_FLOWCTL.FCBBPA` bit is logically OR'ed with the flow control input signal for the back pressure function. When the MAC is configured to Full-Duplex mode, the back pressure function is automatically disabled.

RGMII Control and Status Register

The `EMAC_GIGE_CTLSTAT` register indicates the status signals received from the PHY through the SGMII, RGMII, or SMII interface.

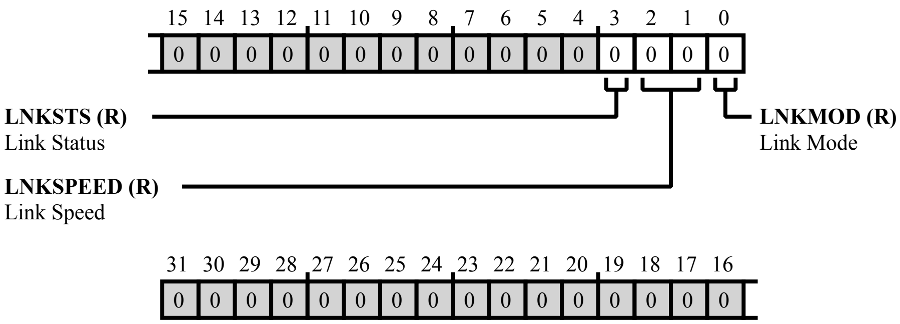

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LNKSTS (R)
Link Status
LNKSPEED (R)
Link Speed
LNKMOD (R)
Link Mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-87: EMAC_GIGE_CTLSTAT Register Diagram

Table 31-119: EMAC_GIGE_CTLSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	LNKSTS	Link Status. The `EMAC_GIGE_CTLSTAT.LNKSTS` bit indicates that the link is up between the local PHY and the remote PHY. When cleared, this bit indicates that the link is down between the local PHY and the remote PHY.
2:1 (R/NW)	LNKSPEED	Link Speed. The `EMAC_GIGE_CTLSTAT.LNKSPEED` bit indicates the current speed of the link.
0 (R/NW)	LNKMOD	Link Mode. The `EMAC_GIGE_CTLSTAT.LNKMOD` bit indicates whether the current mode of link operation is half duplex or full duplex.

Hash Table High Register

The `EMAC_HASHTBL_HI` register contains the upper 32 bits of the hash table.

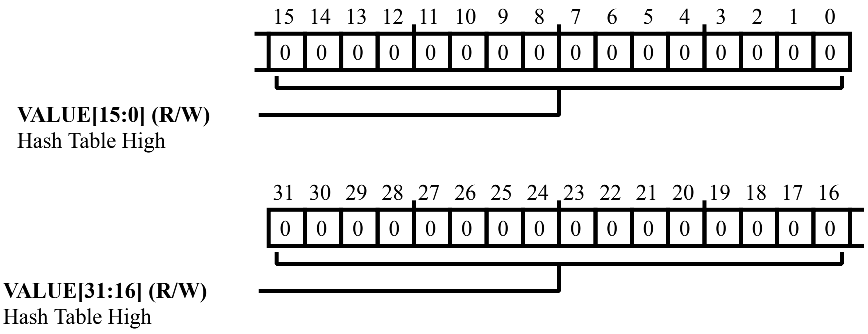

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Hash Table High
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Hash Table High

Figure 31-88: EMAC_HASHTBL_HI Register Diagram

Table 31-120: EMAC_HASHTBL_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table High. The `EMAC_HASHTBL_HI.VALUE` bits contain the upper 32 bits of Hash table.

Hash Table Low Register

The `EMAC_HASHTBL_LO` register contains the lower 32 bits of the hash table.

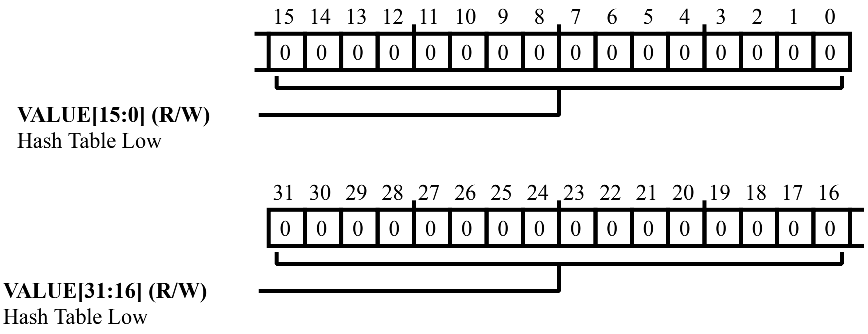

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Hash Table Low
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Hash Table Low

Figure 31-89: EMAC_HASHTBL_LO Register Diagram

Table 31-121: EMAC_HASHTBL_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table Low. The `EMAC_HASHTBL_LO.VALUE` bits contain the lower 32 bits of Hash table.

Interrupt Mask Register

The EMAC_IMSK register enables (unmasks) EMAC interrupts.

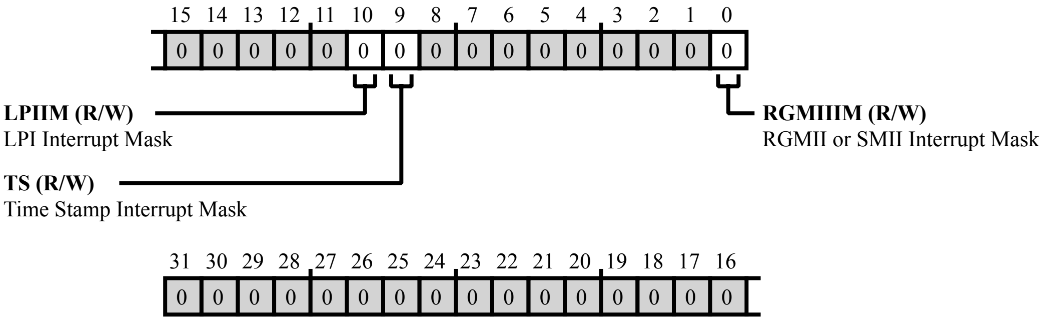

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LPIIM (R/W)
LPI Interrupt Mask
TS (R/W)
Time Stamp Interrupt Mask
RGMIIIM (R/W)
RGMII or SMII Interrupt Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-90: EMAC_IMSK Register Diagram

Table 31-122: EMAC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	LPIIM	LPI Interrupt Mask. The EMAC_IMSK.LPIIM bit, When set, disables the assertion of the interrupt signal because of the setting of the LPI Interrupt Status bit in Interrupt Status Register.
9 (R/W)	TS	Time Stamp Interrupt Mask. The EMAC_IMSK.TS bit, when set, disables the assertion of the interrupt signal, which is generated when the EMAC_ISTAT.TS bit is set.
0 (R/W)	RGMIIIM	RGMII or SMII Interrupt Mask. The EMAC_IMSK.RGMIIIM bit, When set, disables the assertion of the interrupt signal because of the setting of the RGMII Interrupt Status bit in Interrupt Status Register.

MMC IPC Rx Interrupt Mask Register

The `EMAC_IPC_RXIMSK` register enables (unmasks) MMC IPC receive interrupts.

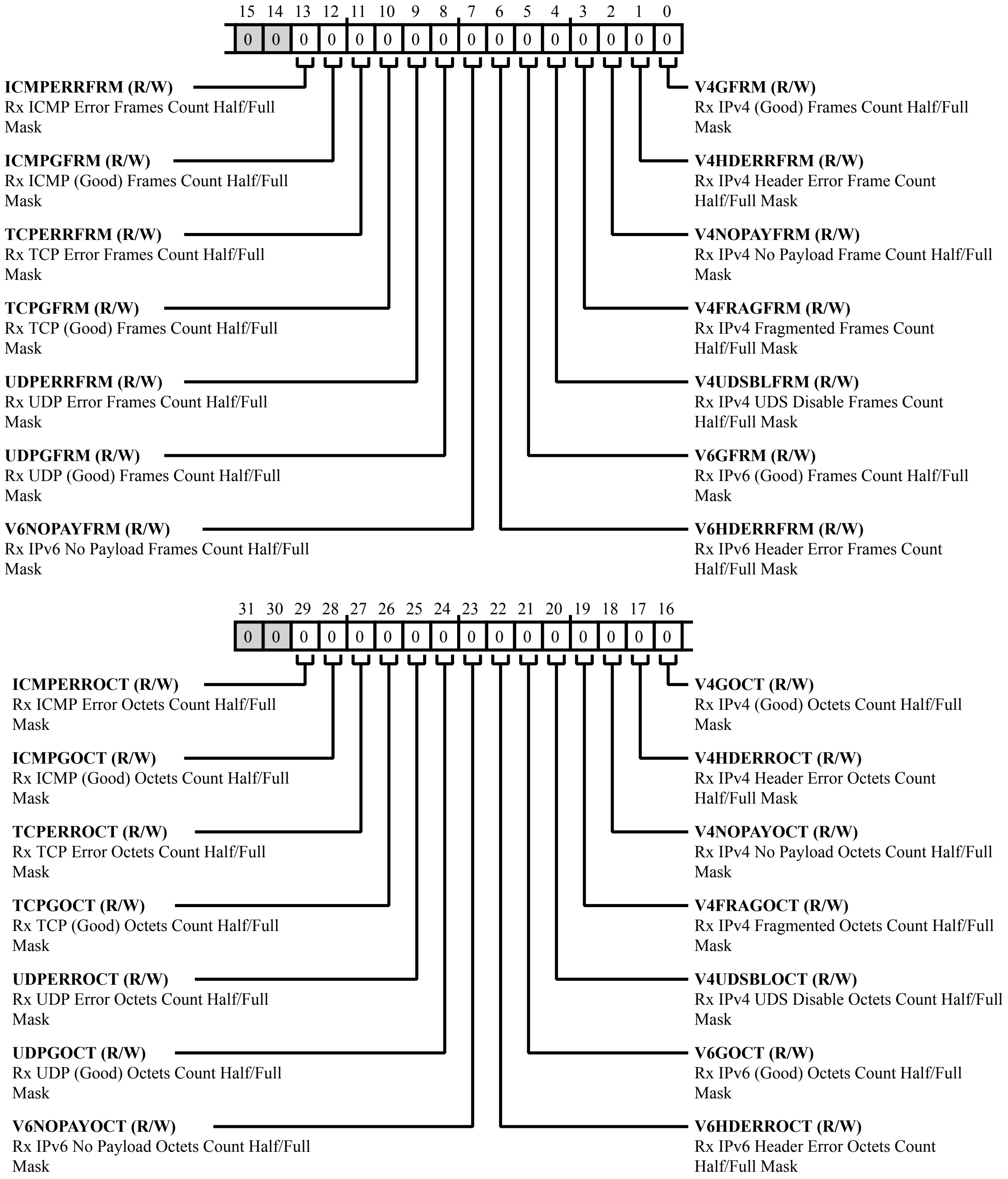

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ICMPERRFRM (R/W)
Rx ICMP Error Frames Count Half/Full Mask
ICMPGFRM (R/W)
Rx ICMP (Good) Frames Count Half/Full Mask
TCPERRFRM (R/W)
Rx TCP Error Frames Count Half/Full Mask
TCPGFRM (R/W)
Rx TCP (Good) Frames Count Half/Full Mask
UDPERRFRM (R/W)
Rx UDP Error Frames Count Half/Full Mask
UDPGFRM (R/W)
Rx UDP (Good) Frames Count Half/Full Mask
V6NOPAYFRM (R/W)
Rx IPv6 No Payload Frames Count Half/Full Mask
V4GFRM (R/W)
Rx IPv4 (Good) Frames Count Half/Full Mask
V4HDERRFRM (R/W)
Rx IPv4 Header Error Frame Count Half/Full Mask
V4NOPAYFRM (R/W)
Rx IPv4 No Payload Frame Count Half/Full Mask
V4FRAGFRM (R/W)
Rx IPv4 Fragmented Frames Count Half/Full Mask
V4UDSBLFRM (R/W)
Rx IPv4 UDS Disable Frames Count Half/Full Mask
V6GFRM (R/W)
Rx IPv6 (Good) Frames Count Half/Full Mask
V6HDERRFRM (R/W)
Rx IPv6 Header Error Frames Count Half/Full Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ICMPERROCT (R/W)
Rx ICMP Error Octets Count Half/Full Mask
ICMPGOCT (R/W)
Rx ICMP (Good) Octets Count Half/Full Mask
TCPERROCT (R/W)
Rx TCP Error Octets Count Half/Full Mask
TCPGOCT (R/W)
Rx TCP (Good) Octets Count Half/Full Mask
UDPERROCT (R/W)
Rx UDP Error Octets Count Half/Full Mask
UDPGOCT (R/W)
Rx UDP (Good) Octets Count Half/Full Mask
V6NOPAYOCT (R/W)
Rx IPv6 No Payload Octets Count Half/Full Mask
V4GOCT (R/W)
Rx IPv4 (Good) Octets Count Half/Full Mask
V4HDERROCT (R/W)
Rx IPv4 Header Error Octets Count Half/Full Mask
V4NOPAYOCT (R/W)
Rx IPv4 No Payload Octets Count Half/Full Mask
V4FRAGOCT (R/W)
Rx IPv4 Fragmented Octets Count Half/Full Mask
V4UDSBLOCT (R/W)
Rx IPv4 UDS Disable Octets Count Half/Full Mask
V6GOCT (R/W)
Rx IPv6 (Good) Octets Count Half/Full Mask
V6HDERROCT (R/W)
Rx IPv6 Header Error Octets Count Half/Full Mask

Figure 31-91: EMAC_IPC_RXIMSK Register Diagram

Table 31-123: EMAC_IPC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.ICMPERROCT` bit, when set, masks the interrupt when the `EMAC_RXICMP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/W)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.ICMPGOCT` bit, when set, masks the interrupt when the `EMAC_RXICMP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/W)	TCPERROCT	Rx TCP Error Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.TCPERROCT` bit, when set, masks the interrupt when the `EMAC_RXTCP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/W)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.TCPGOCT` bit, when set, masks the interrupt when the `EMAC_RXTCP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/W)	UDPERROCT	Rx UDP Error Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.UDPERROCT` bit, when set, masks the interrupt when the `EMAC_RXUDP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/W)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.UDPGOCT` bit, when set, masks the interrupt when the `EMAC_RXUDP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/W)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6NOPAYOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV6_NOPAY_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/W)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6HDERROCT` bit, when set, masks the interrupt when the `EMAC_RXIPV6_HDR_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/W)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6GOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV6_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.

Table 31-123: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4UDSBLOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV4_UDSBL_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
19 (R/W)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4FRAGOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV4_FRAG_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
18 (R/W)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4NOPAYOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV4_NOPAY_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/W)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4HDERROCT` bit, when set, masks the interrupt when the `EMAC_RXIPV4_HDR_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/W)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4GOCT` bit, when set, masks the interrupt when the `EMAC_RXIPV4_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/W)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.ICMPERRFRM` bit, when set, masks the interrupt when the `EMAC_RXICMP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/W)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.ICMPGFRM` bit, when set, masks the interrupt when the `EMAC_RXICMP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/W)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.TCPERRFRM` bit, when set, masks the interrupt when the `EMAC_RXTCP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/W)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.TCPGFRM` bit, when set, masks the interrupt when the `EMAC_RXTCP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.

Table 31-123: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	UDPERRFRM	Rx UDP Error Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.UDPERRFRM` bit, when set, masks the interrupt when the `EMAC_RXUDP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/W)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.UDPGFRM` bit, when set, masks the interrupt when the `EMAC_RXUDP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
7 (R/W)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6NOPAYFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV6_NOPAY_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/W)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6HDERRFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV6_HDR_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
5 (R/W)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V6GFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV6_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/W)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4UDSBLFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV4_UDSBL_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/W)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4FRAGFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV4_FRAG_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/W)	V4NOPAYFRM	Rx IPv4 No Payload Frame Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4NOPAYFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV4_NOPAY_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/W)	V4HDERRFRM	Rx IPv4 Header Error Frame Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4HDERRFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV4_HDR_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.

Table 31-123: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Mask. The `EMAC_IPC_RXIMSK.V4GFRM` bit, when set, masks the interrupt when the `EMAC_RXIPV4_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.

MMC IPC Rx Interrupt Register

The `EMAC_IPC_RXINT` register indicates status of MMC IPC receive interrupts.

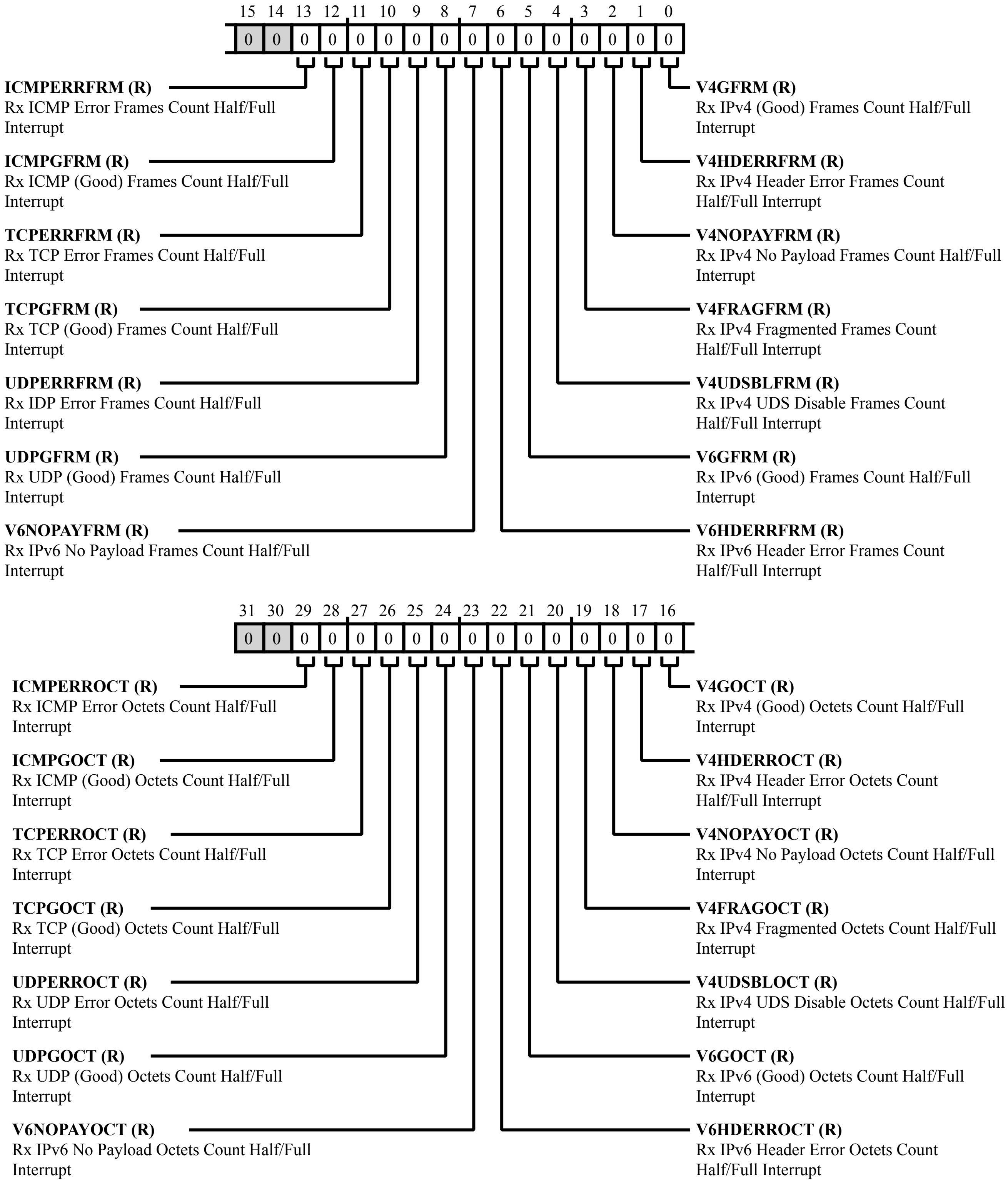

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ICMPERRFRM (R)
Rx ICMP Error Frames Count Half/Full Interrupt
ICMPGFRM (R)
Rx ICMP (Good) Frames Count Half/Full Interrupt
TCPERRFRM (R)
Rx TCP Error Frames Count Half/Full Interrupt
TCPGFRM (R)
Rx TCP (Good) Frames Count Half/Full Interrupt
UDPERRFRM (R)
Rx IDP Error Frames Count Half/Full Interrupt
UDPGFRM (R)
Rx UDP (Good) Frames Count Half/Full Interrupt
V6NOPAYFRM (R)
Rx IPv6 No Payload Frames Count Half/Full Interrupt
V4GFRM (R)
Rx IPv4 (Good) Frames Count Half/Full Interrupt
V4HDERRFRM (R)
Rx IPv4 Header Error Frames Count Half/Full Interrupt
V4NOPAYFRM (R)
Rx IPv4 No Payload Frames Count Half/Full Interrupt
V4FRAGFRM (R)
Rx IPv4 Fragmented Frames Count Half/Full Interrupt
V4UDSBLFRM (R)
Rx IPv4 UDS Disable Frames Count Half/Full Interrupt
V6GFRM (R)
Rx IPv6 (Good) Frames Count Half/Full Interrupt
V6HDERRFRM (R)
Rx IPv6 Header Error Frames Count Half/Full Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ICMPERROCT (R)
Rx ICMP Error Octets Count Half/Full Interrupt
ICMPGOCT (R)
Rx ICMP (Good) Octets Count Half/Full Interrupt
TCPERROCT (R)
Rx TCP Error Octets Count Half/Full Interrupt
TCPGOCT (R)
Rx TCP (Good) Octets Count Half/Full Interrupt
UDPERROCT (R)
Rx UDP Error Octets Count Half/Full Interrupt
UDPGOCT (R)
Rx UDP (Good) Octets Count Half/Full Interrupt
V6NOPAYOCT (R)
Rx IPv6 No Payload Octets Count Half/Full Interrupt
V4GOCT (R)
Rx IPv4 (Good) Octets Count Half/Full Interrupt
V4HDERROCT (R)
Rx IPv4 Header Error Octets Count Half/Full Interrupt
V4NOPAYOCT (R)
Rx IPv4 No Payload Octets Count Half/Full Interrupt
V4FRAGOCT (R)
Rx IPv4 Fragmented Octets Count Half/Full Interrupt
V4UDSBLOCT (R)
Rx IPv4 UDS Disable Octets Count Half/Full Interrupt
V6GOCT (R)
Rx IPv6 (Good) Octets Count Half/Full Interrupt
V6HDERROCT (R)
Rx IPv6 Header Error Octets Count Half/Full Interrupt

Figure 31-92: EMAC_IPC_RXINT Register Diagram

Table 31-124: EMAC_IPC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.ICMPERROCT` bit is set when the `EMAC_RXICMP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/NW)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.ICMPGOCT` bit is set when the `EMAC_RXICMP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/NW)	TCPERROCT	Rx TCP Error Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.TCPERROCT` bit is set when the `EMAC_RXTCP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/NW)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.TCPGOCT` bit is set when the `EMAC_RXTCP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/NW)	UDPERROCT	Rx UDP Error Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.UDPERROCT` bit is set when the `EMAC_RXUDP_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/NW)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.UDPGOCT` bit is set when the `EMAC_RXUDP_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/NW)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6NOPAYOCT` bit is set when the `EMAC_RXIPV6_NOPAY_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/NW)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6HDERROCT` bit is set when the `EMAC_RXIPV6_HDR_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/NW)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6GOCT` bit is set when the `EMAC_RXIPV6_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
20 (R/NW)	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4UDSBLOCT` bit is set when the `EMAC_RXIPV4_UDSBL_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.

Table 31-124: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/NW)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4FRAGOCT` bit is set when the `EMAC_RXIPV4_FRAG_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
18 (R/NW)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4NOPAYOCT` bit set when the `EMAC_RXIPV4_NOPAY_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/NW)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4HDERROCT` bit is set when the `EMAC_RXIPV4_HDR_ERR_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/NW)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4GOCT` bit is set when the `EMAC_RXIPV4_GD_OCT` counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/NW)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.ICMPERRFRM` bit is set when the `EMAC_RXICMP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/NW)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.ICMPGFRM` bit is set when the `EMAC_RXICMP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/NW)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.TCPERRFRM` bit is set when the `EMAC_RXTCP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/NW)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.TCPGFRM` bit is set when the `EMAC_RXTCP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
9 (R/NW)	UDPERRFRM	Rx IDP Error Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.UDPERRFRM` bit is set when the `EMAC_RXUDP_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/NW)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.UDPGFRM` bit is set when the `EMAC_RXUDP_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.

Table 31-124: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6NOPAYFRM` bit is set when the `EMAC_RXIPV6_NOPAY_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/NW)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6HDERRFRM` bit is set when the `EMAC_RXIPV6_HDR_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
5 (R/NW)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V6GFRM` bit is set when the `EMAC_RXIPV6_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/NW)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4UDSBLFRM` bit is set when the `EMAC_RXIPV4_UDSBL_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/NW)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4FRAGFRM` bit is set when the `EMAC_RXIPV4_FRAG_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/NW)	V4NOPAYFRM	Rx IPv4 No Payload Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4NOPAYFRM` bit is set when the `EMAC_RXIPV4_NOPAY_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/NW)	V4HDERRFRM	Rx IPv4 Header Error Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4HDERRFRM` bit is set when the `EMAC_RXIPV4_HDR_ERR_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.
0 (R/NW)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Interrupt. The `EMAC_IPC_RXINT.V4GFRM` bit is set when the `EMAC_RXIPV4_GD_FRM` counter reaches half the maximum value, and also when it reaches the maximum value.

Interrupt Status Register

The EMAC_ISTAT register indicates EMAC interrupt status.

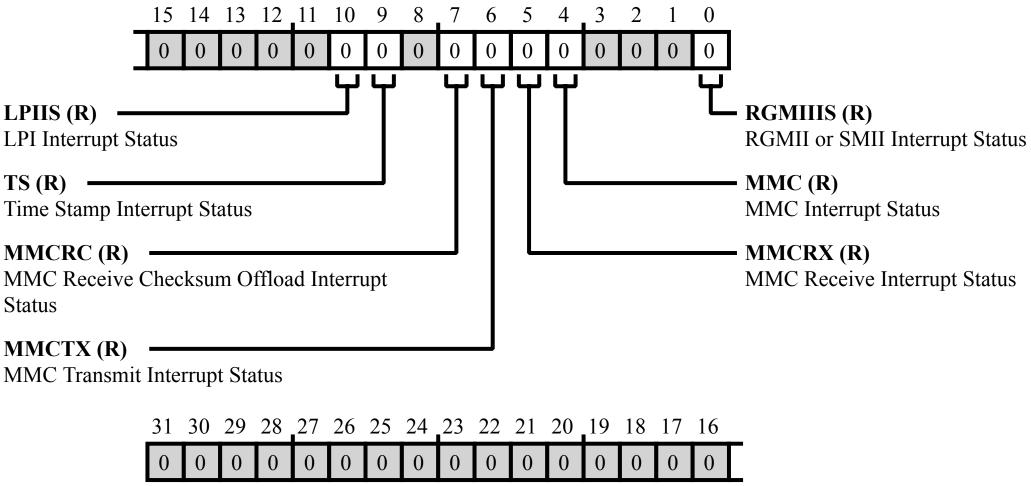

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LPIIS (R)
LPI Interrupt Status
TS (R)
Time Stamp Interrupt Status
MMCRC (R)
MMC Receive Checksum Offload Interrupt Status
MMCTX (R)
MMC Transmit Interrupt Status
RGMIIIS (R)
RGMII or SMII Interrupt Status
MMC (R)
MMC Interrupt Status
MMCRX (R)
MMC Receive Interrupt Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-93: EMAC_ISTAT Register Diagram

Table 31-125: EMAC_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/NW)	LPIIS	LPI Interrupt Status. The `EMAC_ISTAT.LPIIS` bit is set for any LPI state entry or exit in the MAC Transmitter or Receiver.
9 (R/NW)	TS	Time Stamp Interrupt Status. The `EMAC_ISTAT.TS` bit is set when: There is an overflow in the `EMAC_TM_SEC` register, or The `EMAC_TM_STMPSTAT.ATSTS` bit is asserted. The `EMAC_ISTAT.TS` bit is cleared on reading the byte 0 of the `EMAC_TM_STMPSTAT` register.
7 (R/NW)	MMCRC	MMC Receive Checksum Offload Interrupt Status. The `EMAC_ISTAT.MMCRC` bit is set high whenever an interrupt is generated in the `EMAC_IPC_RXINT`. This bit is cleared when all the bits in this interrupt register are cleared.
6 (R/NW)	MMCTX	MMC Transmit Interrupt Status. The `EMAC_ISTAT.MMCTX` bit is set high whenever an interrupt is generated in the `EMAC_MMC_TXINT` register. This bit is cleared when all the bits in this interrupt register are cleared.

Table 31-125: EMAC_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	MMCRX	MMC Receive Interrupt Status. The `EMAC_ISTAT.MMCRX` bit is set high whenever an interrupt is generated in the `EMAC_MMC_RXINT` register. This bit is cleared when all the bits in this interrupt register are cleared.
4 (R/NW)	MMC	MMC Interrupt Status. The `EMAC_ISTAT.MMC` bit is set high whenever any of `EMAC_ISTAT` bits [7:5] is set (=1) and is cleared only when all of these bits are cleared (=0).
0 (R/NW)	RGMIIIS	RGMII or SMII Interrupt Status. The `EMAC_ISTAT.RGMIIIS` bit is set because of any change in value of the Link Status of RGMII interface.

Layer3 and Layer4 Control Register

The `EMAC_L3L4_CTL` register controls the operations of the filter 0 of Layer 3 and Layer 4. This register is reserved if the Layer 3 and Layer 4 Filtering feature is not selected during core configuration.

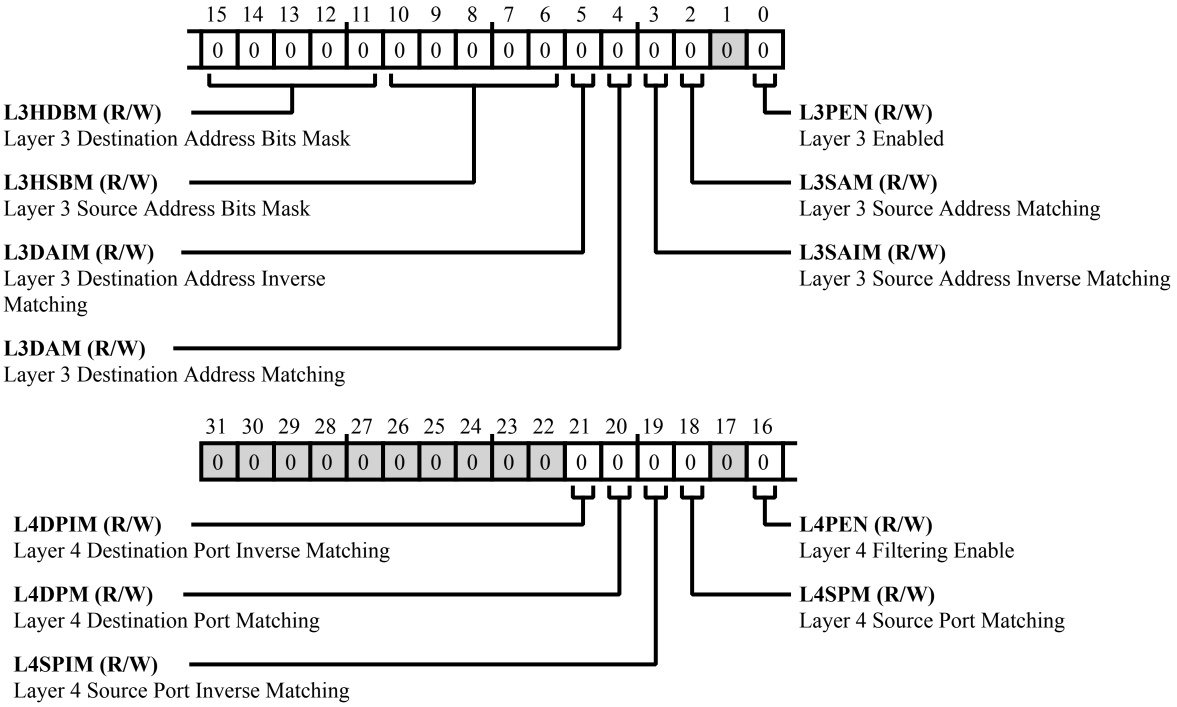

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3HDBM (R/W)
Layer 3 Destination Address Bits Mask
L3HSBM (R/W)
Layer 3 Source Address Bits Mask
L3DAIM (R/W)
Layer 3 Destination Address Inverse Matching
L3DAM (R/W)
Layer 3 Destination Address Matching
L3PEN (R/W)
Layer 3 Enabled
L3SAM (R/W)
Layer 3 Source Address Matching
L3SAIM (R/W)
Layer 3 Source Address Inverse Matching
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L4DPIM (R/W)
Layer 4 Destination Port Inverse Matching
L4DPM (R/W)
Layer 4 Destination Port Matching
L4SPIM (R/W)
Layer 4 Source Port Inverse Matching
L4PEN (R/W)
Layer 4 Filtering Enable
L4SPM (R/W)
Layer 4 Source Port Matching

Figure 31-94: EMAC_L3L4_CTL Register Diagram

Table 31-126: EMAC_L3L4_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	L4DPIM	Layer 4 Destination Port Inverse Matching. The `EMAC_L3L4_CTL.L4DPIM` bit indicates that the Layer 4 Destination Port number field is enabled for inverse matching.
20 (R/W)	L4DPM	Layer 4 Destination Port Matching. The `EMAC_L3L4_CTL.L4DPM` bit indicates that the Layer 4 Destination Port number field is enabled for matching.
19 (R/W)	L4SPIM	Layer 4 Source Port Inverse Matching. The `EMAC_L3L4_CTL.L4SPIM` bit indicates that the Layer 4 Source Port number field is enabled for inverse matching.
18 (R/W)	L4SPM	Layer 4 Source Port Matching. The `EMAC_L3L4_CTL.L4SPM` bit indicates that the Layer 4 Source Port number field is enabled for matching.

Table 31-126: EMAC_L3L4_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	L4PEN	Layer 4 Filtering Enable. The `EMAC_L3L4_CTL.L4PEN` bit indicates that source and destination port number fields for UDP (TCP) frames are used for matching.
15:11 (R/W)	L3HDBM	Layer 3 Destination Address Bits Mask. The `EMAC_L3L4_CTL.L3HDBM` bits are the number of lower bits of IP destination Address that are masked for matching in the IPv4 frames.
10:6 (R/W)	L3HSBM	Layer 3 Source Address Bits Mask. The `EMAC_L3L4_CTL.L3HSBM` bit are the number of lower bits of IP source address that are masked for matching in the IPv4 frames.
5 (R/W)	L3DAIM	Layer 3 Destination Address Inverse Matching. The `EMAC_L3L4_CTL.L3DAIM` bit indicates that the Layer 3 IP destination address field is enabled for inverse matching.
4 (R/W)	L3DAM	Layer 3 Destination Address Matching. The `EMAC_L3L4_CTL.L3DAM` bit indicates that Layer 3 IP destination address field is enabled for matching.
3 (R/W)	L3SAIM	Layer 3 Source Address Inverse Matching. The `EMAC_L3L4_CTL.L3SAIM` bit indicates that the Layer 3 IP Source Address field is enabled for inverse matching.
2 (R/W)	L3SAM	Layer 3 Source Address Matching. The `EMAC_L3L4_CTL.L3SAM` bit indicates that the Layer 3 IP Source Address field is enabled for matching.
0 (R/W)	L3PEN	Layer 3 Enabled. The `EMAC_L3L4_CTL.L3PEN` bit indicates that layer 3 source or destination address matching is enabled for IPV6 (IPV4) frames.

Layer 3 Address0 Register

The `EMAC_L3_ADDR0` register tells For IPv4 frames, the Layer 3 Address 0 Register 0 contains the 32-bit IP Source Address field.

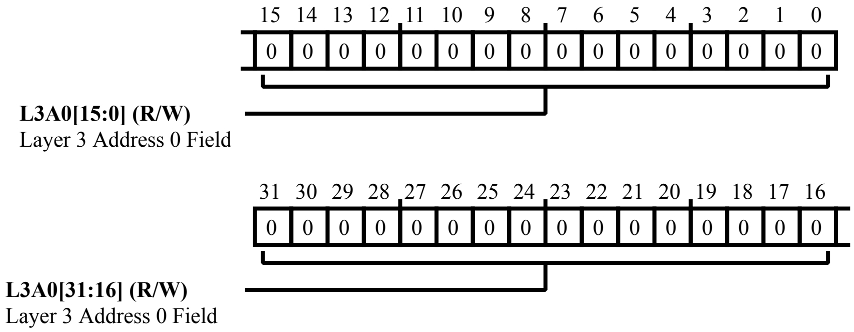

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A0[15:0] (R/W)
Layer 3 Address 0 Field
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A0[31:16] (R/W)
Layer 3 Address 0 Field

Figure 31-95: EMAC_L3_ADDR0 Register Diagram

Table 31-127: EMAC_L3_ADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	L3A0	Layer 3 Address 0 Field. The `EMAC_L3_ADDR0.L3A0` bits, When Bit 0 (L3PEN0) and Bit 2 (L3SAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Source Address field in the IPv6 frames. When Bit 0 (L3PEN0) and Bit 4 (L3DAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Destination Address field in the IPv6 frames. When Bit 0 (L3PEN0) is reset and Bit 2 (L3SAM0) is set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with the IP Source Address field in the IPv4 frames.

Layer 3 Address1 Register

The `EMAC_L3_ADDR1` register tells For IPv4 frames, the Layer 3 Address 1 Register 0 contains the 32-bit IP Source Address field.

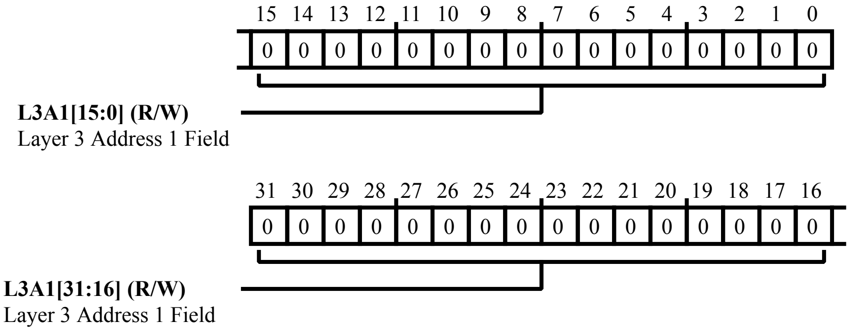

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A1[15:0] (R/W)
Layer 3 Address 1 Field
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A1[31:16] (R/W)
Layer 3 Address 1 Field

Figure 31-96: EMAC_L3_ADDR1 Register Diagram

Table 31-128: EMAC_L3_ADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	L3A1	Layer 3 Address 1 Field. The `EMAC_L3_ADDR1.L3A1` bits, When Bit 0 (L3PEN0) and Bit 2 (L3SAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Source Address field in the IPv6 frames. When Bit 0 (L3PEN0) and Bit 4 (L3DAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Destination Address field in the IPv6 frames. When Bit 0 (L3PEN0) is reset and Bit 2 (L3SAM0) is set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with the IP Source Address field in the IPv4 frames.

Layer 3 Address2 Register

The `EMAC_L3_ADDR2` register tells For IPv4 frames, the Layer 3 Address 2 Register 0 contains the 32-bit IP Source Address field.

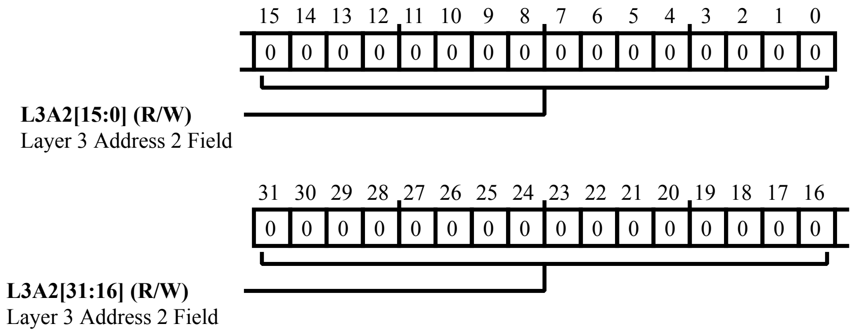

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A2[15:0] (R/W)
Layer 3 Address 2 Field
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A2[31:16] (R/W)
Layer 3 Address 2 Field

Figure 31-97: EMAC_L3_ADDR2 Register Diagram

Table 31-129: EMAC_L3_ADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	L3A2	Layer 3 Address 2 Field. The `EMAC_L3_ADDR2.L3A2` bits, When Bit 0 (L3PEN0) and Bit 2 (L3SAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Source Address field in the IPv6 frames. When Bit 0 (L3PEN0) and Bit 4 (L3DAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Destination Address field in the IPv6 frames. When Bit 0 (L3PEN0) is reset and Bit 2 (L3SAM0) is set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with the IP Source Address field in the IPv4 frames.

Layer 3 Address3 Register

The `EMAC_L3_ADDR3` register tells For IPv4 frames, the Layer 3 Address 3 Register 0 contains the 32-bit IP Source Address field.

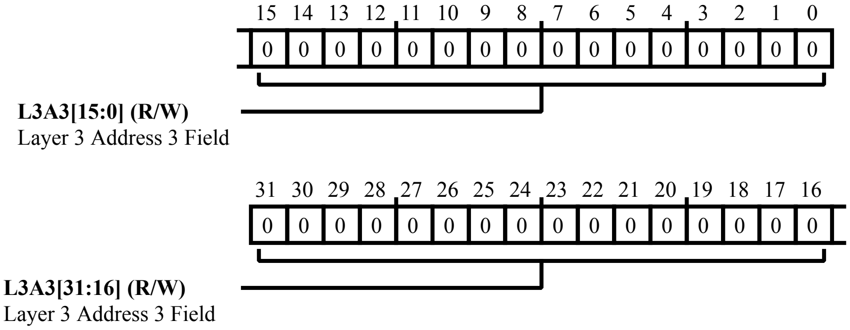

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A3[15:0] (R/W)
Layer 3 Address 3 Field
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L3A3[31:16] (R/W)
Layer 3 Address 3 Field

Figure 31-98: EMAC_L3_ADDR3 Register Diagram

Table 31-130: EMAC_L3_ADDR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	L3A3	Layer 3 Address 3 Field. The `EMAC_L3_ADDR3.L3A3` bits, When Bit 0 (L3PEN0) and Bit 2 (L3SAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Source Address field in the IPv6 frames. When Bit 0 (L3PEN0) and Bit 4 (L3DAM0) are set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with Bits [31:0] of the IP Destination Address field in the IPv6 frames. When Bit 0 (L3PEN0) is reset and Bit 2 (L3SAM0) is set in Register 256 (Layer 3 and Layer 4 Control Register 0), this field contains the value to be matched with the IP Source Address field in the IPv4 frames.

Layer 4 Address Register

The `EMAC_L4_ADDR` register contains Layer 4 Port number field. It contains the 16-bit Source and Destination Port numbers of the TCP or UDP frame.

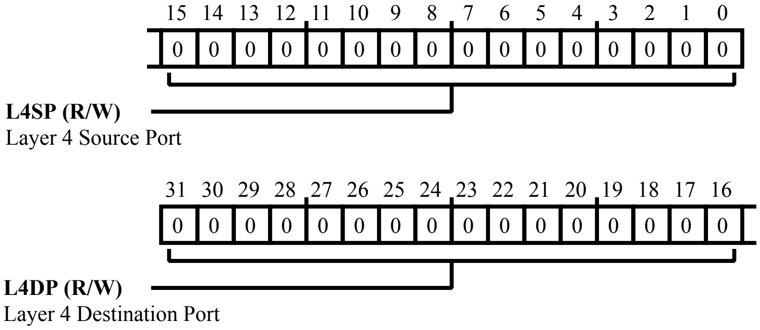

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L4SP (R/W)
Layer 4 Source Port
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
L4DP (R/W)
Layer 4 Destination Port

Figure 31-99: EMAC_L4_ADDR Register Diagram

Table 31-131: EMAC_L4_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	L4DP	Layer 4 Destination Port. When `EMAC_L3L4_CTL.L4PEN` is reset and `EMAC_L3L4_CTL.L4DPM` is set, the `EMAC_L4_ADDR.L4DP` bit field contains the value to be matched with the TCP Destination Port Number field in the IPv4 or IPv6 frames. When `EMAC_L3L4_CTL.L4PEN` and `EMAC_L3L4_CTL.L4DPM` are set, the `EMAC_L4_ADDR.L4DP` bit field contains the value to be matched with the UDP Destination Port Number field in the IPv4 or IPv6 frames.
15:0 (R/W)	L4SP	Layer 4 Source Port. When `EMAC_L3L4_CTL.L4PEN` is reset and `EMAC_L3L4_CTL.L4DPM` is set, the `EMAC_L4_ADDR.L4SP` bit field contains the value to be matched with the TCP Source Port Number field in the IPv4 or IPv6 frames. When `EMAC_L3L4_CTL.L4PEN` and `EMAC_L3L4_CTL.L4DPM` are set, the `EMAC_L4_ADDR.L4SP` bit field contains the value to be matched with the UDP Source Port Number field in the IPv4 or IPv6 frames.

Low Power Idle Control and Status Register

The `EMAC_LPI_CTLSTAT` register controls the behavior of EMAC0 for LPI mode and reports status.

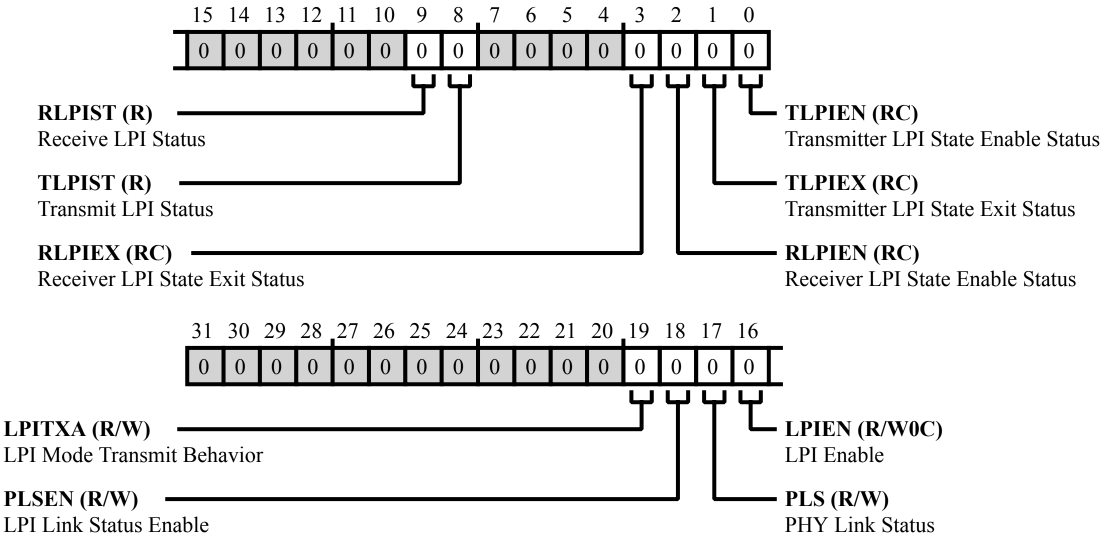

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RLPIST (R)
Receive LPI Status
TLPIST (R)
Transmit LPI Status
RLPIEX (RC)
Receiver LPI State Exit Status
TLPIEN (RC)
Transmitter LPI State Enable Status
TLPIEX (RC)
Transmitter LPI State Exit Status
RLPIEN (RC)
Receiver LPI State Enable Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LPITXA (R/W)
LPI Mode Transmit Behavior
PLSEN (R/W)
LPI Link Status Enable
LPIEN (R/W0C)
LPI Enable
PLS (R/W)
PHY Link Status

Figure 31-100: EMAC_LPI_CTLSTAT Register Diagram

Table 31-132: EMAC_LPI_CTLSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	LPITXA	LPI Mode Transmit Behavior. The `EMAC_LPI_CTLSTAT.LPITXA` bit controls the behavior of the MAC when it is entering or exiting of the LPI mode on the transmit side.
18 (R/W)	PLSEN	LPI Link Status Enable. The `EMAC_LPI_CTLSTAT.PLSEN` bit enables the link status received on the RGMII receive paths to be used for activating the LPI LS timer.
17 (R/W)	PLS	PHY Link Status. The `EMAC_LPI_CTLSTAT.PLS` bit indicates the link status of the PHY
16 (R/W0C)	LPIEN	LPI Enable. The `EMAC_LPI_CTLSTAT.LPIEN` bit instructs the MAC Transmitter to enter the LPI state. When reset, this bit instructs the MAC to exit the LPI state and resume normal transmission.
9 (R/NW)	RLPIST	Receive LPI Status. The `EMAC_LPI_CTLSTAT.RLPIST` bit indicates that the MAC is receiving the LPI pattern on the GMII or MII interface.

Table 31-132: EMAC_LPI_CTLSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/NW)	TLPIST	Transmit LPI Status. The `EMAC_LPI_CTLSTAT.TLPIST` bit indicates that the MAC is transmitting the LPI pattern on the GMII or MII interface.
3 (RC/NW)	RLPIEX	Receiver LPI State Exit Status. The `EMAC_LPI_CTLSTAT.RLPIEX` bit indicates that the MAC Receiver has stopped receiving the LPI pattern on the GMII or MII interface, exited the LPI state, and resumed the normal reception. This bit is cleared by a read into this register.
2 (RC/NW)	RLPIEN	Receiver LPI State Enable Status. The `EMAC_LPI_CTLSTAT.RLPIEN` bit indicates that the MAC Receiver has received an LPI pattern and entered the LPI state. This bit is cleared by a read into this register.
1 (RC/NW)	TLPIEX	Transmitter LPI State Exit Status. The `EMAC_LPI_CTLSTAT.TLPIEX` bit indicates that the MAC transmitter has exited the LPI state after the program has cleared the `EMAC_LPI_CTLSTAT.LPIEN` bit and the LPI TW Timer has expired. This bit is cleared by a read into this register.
0 (RC/NW)	TLPIEN	Transmitter LPI State Enable Status. The `EMAC_LPI_CTLSTAT.TLPIEN` bit indicates that the MAC Transmitter has entered the LPI state due to the setting of the `EMAC_LPI_CTLSTAT.LPIEN` bit. This bit is cleared by a read into this register.

Low Power Idle Timeout Register

The EMAC_LPI_TMRSCTL controls the timeout values in the LPI states. It specifies the time for which the MAC transmits the LPI pattern and also the time for which the MAC waits before resuming the normal transmission.

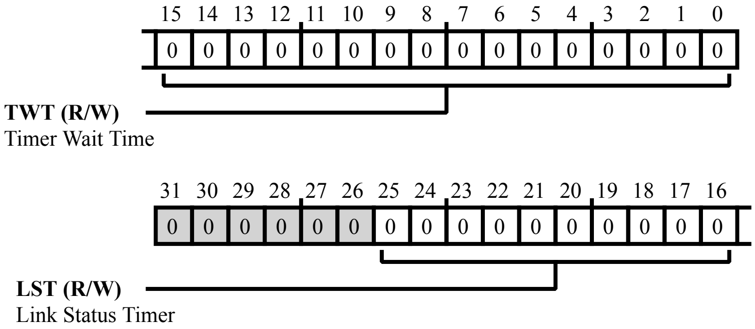

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TWT (R/W)
Timer Wait Time
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LST (R/W)
Link Status Timer

Figure 31-101: EMAC_LPI_TMRSCTL Register Diagram

Table 31-133: EMAC_LPI_TMRSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	LST	Link Status Timer. The `EMAC_LPI_TMRSCTL.LST` bit, specifies the minimum time (in milliseconds) for which the link status from the PHY should be up (OKAY) before the LPI pattern can be transmitted to the PHY.
15:0 (R/W)	TWT	Timer Wait Time. The `EMAC_LPI_TMRSCTL.TWT` bit, specifies the minimum time (in microseconds) for which the MAC waits after it stops transmitting the LPI pattern to the PHY and before it resumes the normal transmission.

MAC Configuration Register

The `EMAC_MACCFG` register configures MAC features.

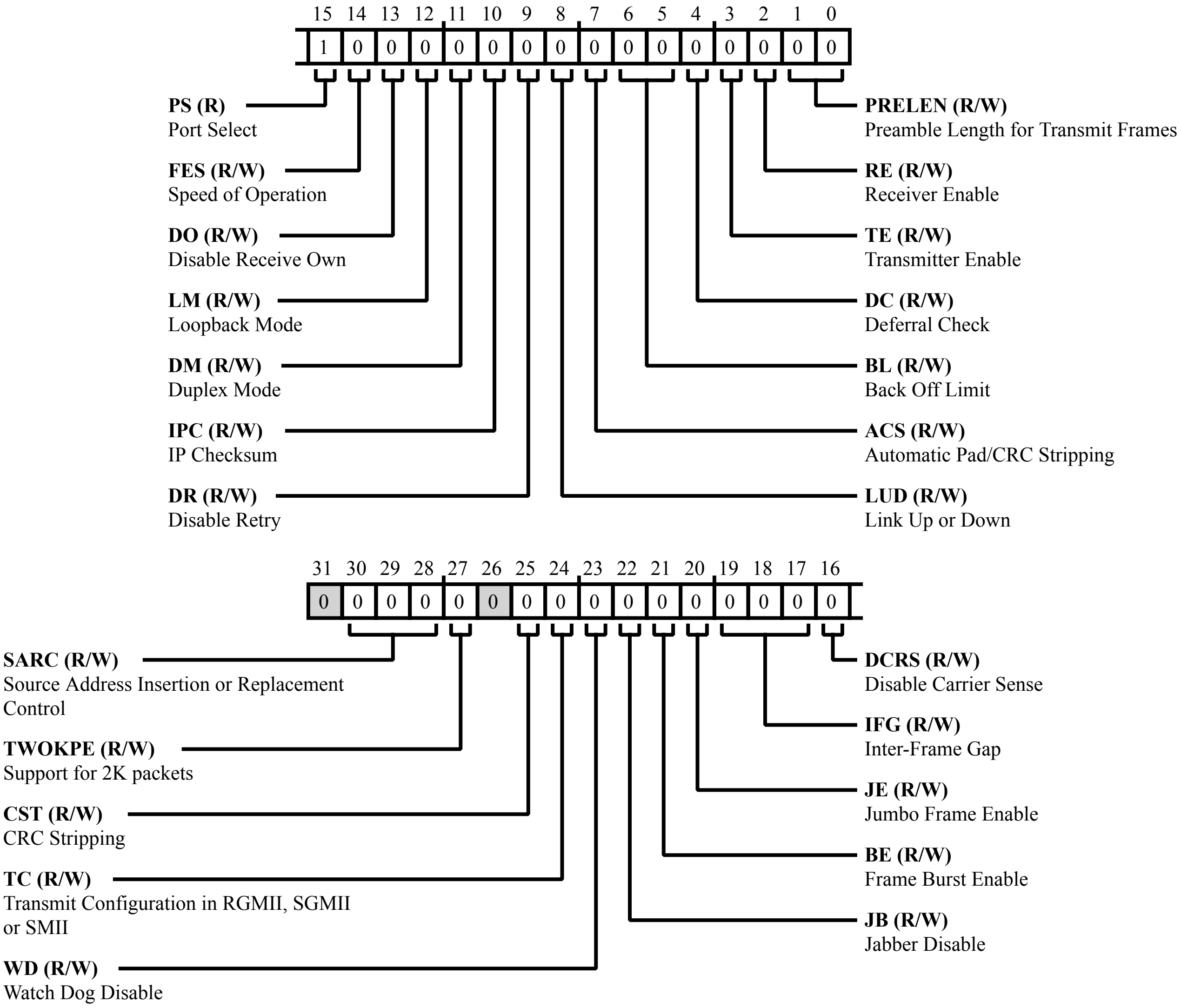

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PS (R)
Port Select
FES (R/W)
Speed of Operation
DO (R/W)
Disable Receive Own
LM (R/W)
Loopback Mode
DM (R/W)
Duplex Mode
IPC (R/W)
IP Checksum
DR (R/W)
Disable Retry
PRELEN (R/W)
Preamble Length for Transmit Frames
RE (R/W)
Receiver Enable
TE (R/W)
Transmitter Enable
DC (R/W)
Deferral Check
BL (R/W)
Back Off Limit
ACS (R/W)
Automatic Pad/CRC Stripping
LUD (R/W)
Link Up or Down
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SARC (R/W)
Source Address Insertion or Replacement Control
TWOKPE (R/W)
Support for 2K packets
CST (R/W)
CRC Stripping
TC (R/W)
Transmit Configuration in RGMII, SGMII or SMII
WD (R/W)
Watch Dog Disable
DCRS (R/W)
Disable Carrier Sense
IFG (R/W)
Inter-Frame Gap
JE (R/W)
Jumbo Frame Enable
BE (R/W)
Frame Burst Enable
JB (R/W)
Jabber Disable

Figure 31-102: EMAC_MACCFG Register Diagram

Table 31-134: EMAC_MACCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:28 (R/W)	SARC	Source Address Insertion or Replacement Control. The `EMAC_MACCFG.SARC` bit, controls the source address insertion or replacement for all transmitted frames.

Table 31-134: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	TWOKPE	Support for 2K packets. The `EMAC_MACCFG.TWOKPE` bit, IEEE 802.3as Support for 2K Packets When set, the MAC considers all frames, with up to 2,000 bytes length, as normal packets.	
25 (R/W)	CST	CRC Stripping. The `EMAC_MACCFG.CST` bit, when set, directs the MAC to strip the last 4 bytes (FCS) of all frames of Ether type (Type field of frame greater than 0x0600) and drop these bytes before forwarding the frame to the application.	
24 (R/W)	TC	Transmit Configuration in RGMII, SGMII or SMII. The `EMAC_MACCFG.TC` bit, enables the transmission of duplex mode, link speed, and link up or down information to the PHY in the RGMII port.	
23 (R/W)	WD	Watch Dog Disable. The `EMAC_MACCFG.WD` bit, when set, disables the watchdog timer on the receiver, and can receive frames of up to 16,384 bytes. When this bit is reset, the MAC allows no more than 2,048 bytes (10,240 if `EMAC_MACCFG.JE` is set high) of the frame being received and cuts off any bytes received after that.	
22 (R/W)	JB	Jabber Disable. The `EMAC_MACCFG.JB` bit, when set, disables the jabber timer on the transmitter, and can transfer frames of up to 16,384 bytes. When this bit is reset, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if `EMAC_MACCFG.JE` is set high) during transmission.	
21 (R/W)	BE	Frame Burst Enable. The `EMAC_MACCFG.BE` bit, allows frame bursting during transmission in the GMII half-duplex mode.	
20 (R/W)	JE	Jumbo Frame Enable. The `EMAC_MACCFG.JE` bit, when set, directs the MAC to allow Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames).	
19:17 (R/W)	IFG	Inter-Frame Gap. The `EMAC_MACCFG.IFG` bits control the minimum inter-frame gap between frames during transmission. Note that in Half-Duplex mode, the minimum gap can be configured for 64 bit times (`EMAC_MACCFG.IFG` =100) only. Lower values are not considered.	
		0	96 bit times
		1	88 bit times
		2	80 bit times
		3	72 bit times
		4	64 bit times

Table 31-134: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	56 bit times
		6	48 bit times
		7	40 bit times
16 (R/W)	DCRS	Disable Carrier Sense. The `EMAC_MACCFG.DCRS` bit, when set, makes the MAC transmitter ignore the CRS signal during frame transmission in Half-Duplex mode. This request results in no errors generated due to Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors due to Carrier Sense and will even abort the transmissions.	
15 (R/NW)	PS	Port Select. The `EMAC_MACCFG.PS` bit selects between GMII and MII as: 0=GMII (1000 Mbps) and 1=MII (10/100 Mbps). This bit is read-only with the appropriate value in the 10/100 Mbps-only (always 1) or 1000 Mbps-only (always 0) configurations, and R_W in the default 10/100/1000 Mbps configuration.	
14 (R/W)	FES	Speed of Operation. The `EMAC_MACCFG.FES` bit indicates the Ethernet speed as 10 Mbps (bit =0) or 100 Mbps (bit =1).	
13 (R/W)	DO	Disable Receive Own. The `EMAC_MACCFG.DO` bit, when set, disables MAC reception of frames when MAC is transmitting in Half-Duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in Full-Duplex mode.	
12 (R/W)	LM	Loopback Mode. The `EMAC_MACCFG.LM` bit, when set, directs the MAC to operate in internal loop back mode. (The media independent interface pins are not driven or sampled.)	
11 (R/W)	DM	Duplex Mode. The `EMAC_MACCFG.DM` bit, when set, directs the MAC to operate in a Full-Duplex mode where it can transmit and receive simultaneously.	
10 (R/W)	IPC	IP Checksum. The `EMAC_MACCFG.IPC` bit, when set, directs the MAC to calculate the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25-26 or 29-30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The `EMAC_MACCFG.IPC` bit, when set, enables IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. When this bit is reset, the Checksum Offload Engine function in the receiver is disabled and the corresponding PCE and IP HCE status bits are always cleared.	

Table 31-134: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	DR	Disable Retry. The `EMAC_MACCFG.DR` bit, when set, directs the MAC to attempt only 1 transmission. When a collision occurs on the media independent interface, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status. When the `EMAC_MACCFG.DR` bit is reset, the MAC attempts retries based on the settings of BL. This bit is applicable only to Half-Duplex mode.	
		0	Retry enabled
		1	Retry disabled
8 (R/W)	LUD	Link Up or Down. The `EMAC_MACCFG.LUD` bit, indicates whether the link is up or down during the transmission of configuration in the RGMII, SGMII, or SMII interface: 0 means Link down and 1 means Link Up	
7 (R/W)	ACS	Automatic Pad/CRC Stripping. The `EMAC_MACCFG.ACS` bit, when set, directs the MAC to strip the Pad/FCS field on incoming frames only if the length fields value is less than or equal to 1,500 bytes. All received frames with length field greater than or equal to 1,501 bytes are passed to the application without stripping the Pad/FCS field. When the `EMAC_MACCFG.ACS` bit is reset, the MAC passes all incoming frames to the Host unmodified.	
6:5 (R/W)	BL	Back Off Limit. The `EMAC_MACCFG.BL` bit selects the back-off limit, determining the random integer number (r) of slot time delays (512 bit times for 10/100 Mbps) the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only to Half-Duplex mode.The random integer r takes the value in the range: 0 less-than-equal-to r less-than 2^k Where k is the minimum of n (number of transmission attempts) or a limit value.	
		0	k = min (n, 10)
		1	k = min (n, 8)
		2	k = min (n, 4)
		3	k = min (n, 1)

Table 31-134: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	DC	Deferral Check. The `EMAC_MACCFG.DC` bit, when set, enables the deferral check function in the MAC. The MAC issues a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status when the transmit state machine is deferred for more than 24,288 bit times in 10/100-Mbps mode. If the Jumbo frame mode is enabled in 10/100-Mbps mode, the threshold for deferral is 155,680 bits times. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense) signal. Defer time is not cumulative. If the transmitter defers for 10,000 bit times, then transmits, collides, backs off, and then has to defer again after completion of back-off, the deferral timer resets to 0 and restarts. When the `EMAC_MACCFG.DC` bit is reset, the deferral check function is disabled and the MAC defers until the CRS signal goes inactive. This bit is applicable only in Half-Duplex mode.
3 (R/W)	TE	Transmitter Enable. The `EMAC_MACCFG.TE` bit, when set, enables the transmit state machine of the MAC for transmission. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and will not transmit any further frames.
2 (R/W)	RE	Receiver Enable. The `EMAC_MACCFG.RE` bit, when set, enables the receiver state machine of the MAC for receiving frames. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames.
1:0 (R/W)	PRELEN	Preamble Length for Transmit Frames. The `EMAC_MACCFG.PRELEN` bit, control the number of preamble bytes that are added to the beginning of every Transmit frame.

MAC Rx Frame Filter Register

The `EMAC_MACFRMFILT` register controls receive frame filter features.

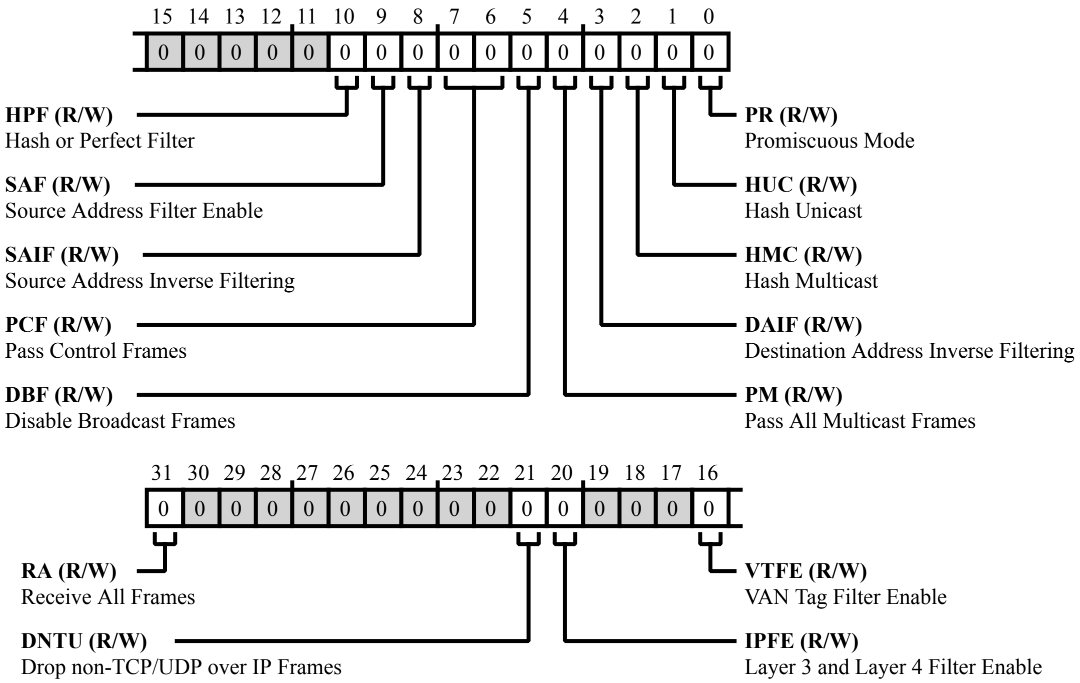

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HPF (R/W)
Hash or Perfect Filter
SAF (R/W)
Source Address Filter Enable
SAIF (R/W)
Source Address Inverse Filtering
PCF (R/W)
Pass Control Frames
DBF (R/W)
Disable Broadcast Frames
PR (R/W)
Promiscuous Mode
HUC (R/W)
Hash Unicast
HMC (R/W)
Hash Multicast
DAIF (R/W)
Destination Address Inverse Filtering
PM (R/W)
Pass All Multicast Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RA (R/W)
Receive All Frames
DNTU (R/W)
Drop non-TCP/UDP over IP Frames
VTFE (R/W)
VAN Tag Filter Enable
IPFE (R/W)
Layer 3 and Layer 4 Filter Enable

Figure 31-103: EMAC_MACFRMFILT Register Diagram

Table 31-135: EMAC_MACFRMFILT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	RA	Receive All Frames. The `EMAC_MACFRMFILT.RA` bit, when set, directs the MAC Receiver module to pass to the Application all frames received irrespective of whether they pass the address filter. The result of the DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes to the Application only those frames that pass the DA address filter.
21 (R/W)	DNTU	Drop non-TCP/UDP over IP Frames. The `EMAC_MACFRMFILT.DNTU` bit, enables the MAC to drop the non-TCP or UDP over IP frames. The MAC forward only those frames that are processed by the Layer 4 filter. When reset, this bit enables the MAC to forward all non-TCP or UDP over IP frames.
20 (R/W)	IPFE	Layer 3 and Layer 4 Filter Enable. The `EMAC_MACFRMFILT.IPFE` bit, enables the MAC to drop frames that do not match the enabled Layer 3 and Layer 4 filters. If Layer 3 or Layer 4 filters are not enabled for matching, this bit does not have any effect.

Table 31-135: EMAC_MACFRMFILT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	VTFE	VAN Tag Filter Enable. The `EMAC_MACFRMFILT.VTFE` bit, enables the MAC to drop VLAN tagged frames that do not match the VLAN Tag comparison.	
10 (R/W)	HPF	Hash or Perfect Filter. The `EMAC_MACFRMFILT.HPF` bit. when set, configures the address filter to pass a frame if it matches either the perfect filtering or the hash filtering as set by `EMAC_MACFRMFILT.HMC` or `EMAC_MACFRMFILT.HUC` bits. When `EMAC_MACFRMFILT.HPF` is low and either the `EMAC_MACFRMFILT.HUC` bit or `EMAC_MACFRMFILT.HMC` bit is set, the frame is passed only if it matches the Hash filter.	
9 (R/W)	SAF	Source Address Filter Enable. When the `EMAC_MACFRMFILT.SAF` bit, is set, the MAC compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison fails, the MAC drops the frame.	
8 (R/W)	SAIF	Source Address Inverse Filtering. WHen the `EMAC_MACFRMFILT.SAIF` bit is set, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA Address filter.	
7:6 (R/W)	PCF	Pass Control Frames. The `EMAC_MACFRMFILT.PCF` bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on the value of the `EMAC_FLOWCTL.RFE` bit.	
		0	Pass no control frames All control frames are filtered from reaching the application.
		1	Pass no PAUSE frames All control frames are passed to the application (even if the fail the address filter), except for PAUSE frames.
		2	Pass all control frames All control frames are passed to the application (even if the fail the address filter).
		3	Pass address filtered control frames All control frames that pass the address filter are passed to the application.
5 (R/W)	DBF	Disable Broadcast Frames. The `EMAC_MACFRMFILT.DBF` bit, when set, directs the AFM module to filter all incoming broadcast frames. When this bit is reset, the AFM module passes all received broadcast frames.	
		0	AFM module passes all received broadcast frames
		1	AFM module filters all incoming broadcast frames

Table 31-135: EMAC_MACFRMFILT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	PM	Pass All Multicast Frames. The `EMAC_MACFRMFILT.PM` bit, when set, indicates that all received frames with a multicast destination address (first bit in the destination address field is =1) are passed. When this bit is reset, filtering of multicast frame depends on `EMAC_MACFRMFILT.HMC` bit.
3 (R/W)	DAIF	Destination Address Inverse Filtering. The `EMAC_MACFRMFILT.DAIF` bit, when set, directs the Address Check block to operate in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When this bit is reset, normal filtering of frames is performed.
2 (R/W)	HMC	Hash Multicast. The `EMAC_MACFRMFILT.HMC` bit, when set, directs the EMAC to perform destination address filtering of received multicast frames according to the hash table. When this bit is reset, the MAC performs a perfect destination address filtering for multicast frames, that is, the MAC compares the DA field with the values programmed in the `EMAC_ADDR0_HI` and `EMAC_ADDR0_LO` address registers.
1 (R/W)	HUC	Hash Unicast. The `EMAC_MACFRMFILT.HUC` bit, when set, directs the EMAC to perform destination address filtering of unicast frames according to the hash table. When this bit is reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in the `EMAC_ADDR0_HI` and `EMAC_ADDR0_LO` address registers.
0 (R/W)	PR	Promiscuous Mode. The `EMAC_MACFRMFILT.PR` bit, when set, directs the Address Filter module to pass all incoming frames regardless of its destination or source address. The DA Filter Fails status bits of the Receive Status Word is always cleared when `EMAC_MACFRMFILT.PR` is set.

AV MAC Control Register

The EMAC_MAC_AVCTL register controls the AV traffic by identifying the AV traffic and queuing it to appropriate channel. This register is present only when you select the AV feature during core configuration.

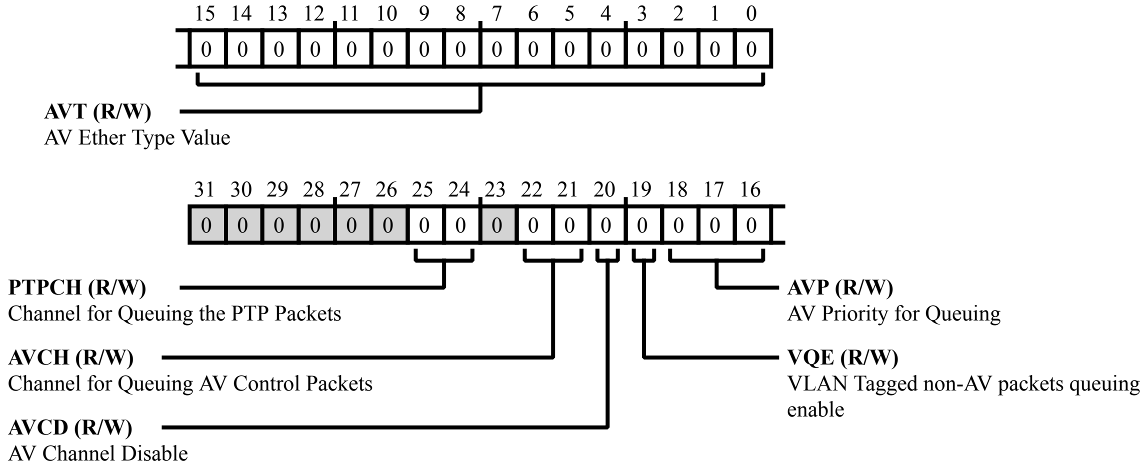

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AVT (R/W)
AV Ether Type Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PTPCH (R/W)
Channel for Queuing the PTP Packets
AVCH (R/W)
Channel for Queuing AV Control Packets
AVCD (R/W)
AV Channel Disable
AVP (R/W)
AV Priority for Queuing
VQE (R/W)
VLAN Tagged non-AV packets queuing enable

Figure 31-104: EMAC_MAC_AVCTL Register Diagram

Table 31-136: EMAC_MAC_AVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/W)	PTPCH	Channel for Queuing the PTP Packets. The `EMAC_MAC_AVCTL.PTPCH` bit, specifies the channel on which the untagged PTP packets, sent over the Ethernet payload and not over IPv4 or IPv6, are queued.
22:21 (R/W)	AVCH	Channel for Queuing AV Control Packets. The `EMAC_MAC_AVCTL.AVCH` bit, specifies the channel on which the received untagged AV control packets are queued.
20 (R/W)	AVCD	AV Channel Disable. The `EMAC_MAC_AVCTL.AVCD` bit, When this bit is set, the MAC forwards all packets to the default Channel 0 and the values programmed in the AVP, AVCH, and PTPCH fields are ignored. This bit is reserved and read-only if Channel 1 or Channel 2 receive paths are not selected during core configuration.
19 (R/W)	VQE	VLAN Tagged non-AV packets queuing enable. The `EMAC_MAC_AVCTL.VQE` bit, When this bit is set, the MAC also queues non-AV VLAN tagged packets into the available channels according to the value of the AVP bits. This bit is reserved and read-only if Channel 1 and Channel 2 Receive paths are not selected during core configuration.

Table 31-136: EMAC_MAC_AVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18:16 (R/W)	AVP	AV Priority for Queuing. The `EMAC_MAC_AVCTL.AVP` bit, The value programmed in these bits control the receive channel (0, 1, or 2) to which an AV packet with a given priority must be queued.
15:0 (R/W)	AVT	AV Ether Type Value. The `EMAC_MAC_AVCTL.AVT` bit, contains the value that is compared with the EtherType field of the incoming (tagged or untagged) Ethernet frame to detect an AV packet.

MMC Control Register

The EMAC_MMC_CTL register selects the MMC operating mode.

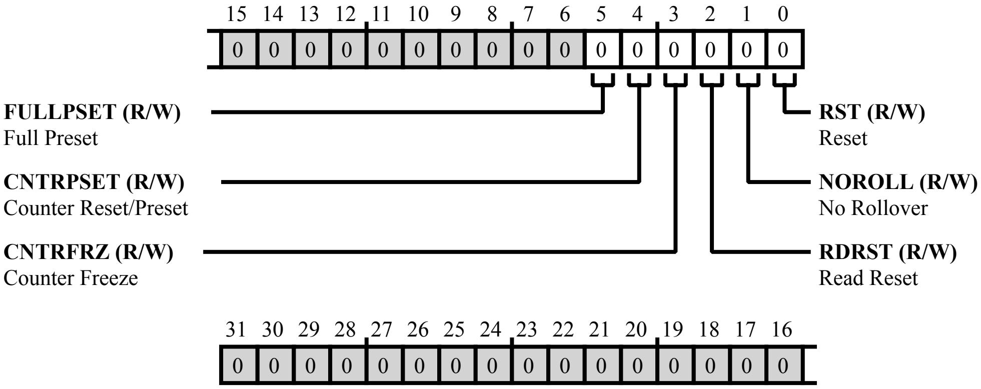

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FULLPSET (R/W)
Full Preset
CNTRPSET (R/W)
Counter Reset/Preset
CNTRFRZ (R/W)
Counter Freeze
RST (R/W)
Reset
NOROLL (R/W)
No Rollover
RDRST (R/W)
Read Reset
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-105: EMAC_MMC_CTL Register Diagram

Table 31-137: EMAC_MMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	FULLPSET	Full Preset. The EMAC_MMC_CTL.FULLPSET bit, when =0 (and EMAC_MMC_CTL.CNTRPSET =1), presets all MMC counters to almost-half value. All octet counters get preset to 0x7FFF_F800 (half - 2KBytes) and all frame-counters gets preset to 0x7FFF_FFF0 (half - 16). When EMAC_MMC_CTL.FULLPSET =1 (and EMAC_MMC_CTL.CNTRPSET =1), all MMC counters get preset to almost-full value. All octet counters get preset to 0xFFFF_F800 (full - 2KBytes) and all frame-counters gets preset to 0xFFFF_FFF0 (full - 16). For 16-bit counters, the almost-half preset values are 0x7800 and 0x7FF0 for the respective octet and frame counters. Similarly, the almost-full preset values for the 16-bit counters are 0xF800 and 0xFFF0.
4 (R/W)	CNTRPSET	Counter Reset/Preset. The EMAC_MMC_CTL.CNTRPSET bit, when set, initializes all counters or presets counters to almost full or almost half as per EMAC_MMC_CTL.FULLPSET. The EMAC_MMC_CTL.CNTRPSET bit is cleared automatically after 1 clock cycle. This bit along with bit5 is useful for debugging and testing the assertion of interrupts because of MMC counter becoming half-full or full.
3 (R/W)	CNTRFRZ	Counter Freeze. The EMAC_MMC_CTL.CNTRFRZ bit, when set, freezes all the MMC counters to their current value. None of the MMC counters are updated due to any transmitted or received frame, until this bit is reset to 0. If any MMC counter is read with the EMAC_MMC_CTL.RDRST bit set, then that counter is also cleared in this mode.

Table 31-137: EMAC_MMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	RDRST	Read Reset. The `EMAC_MMC_CTL.RDRST` bit, when set, resets the MMC counters to zero after Read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits[7:0]) is read.
1 (R/W)	NOROLL	No Rollover. The `EMAC_MMC_CTL.NOROLL` bit, when set, prevents counter rolls over to 0 after reaching max.
0 (R/W)	RST	Reset. The `EMAC_MMC_CTL.RST` bit, when set, resets all counters. This bit is cleared automatically after 1 clock cycle.

MMC Rx Interrupt Mask Register

The `EMAC_MMC_RXIMSK` register enables (unmasks) MMC receive interrupts.

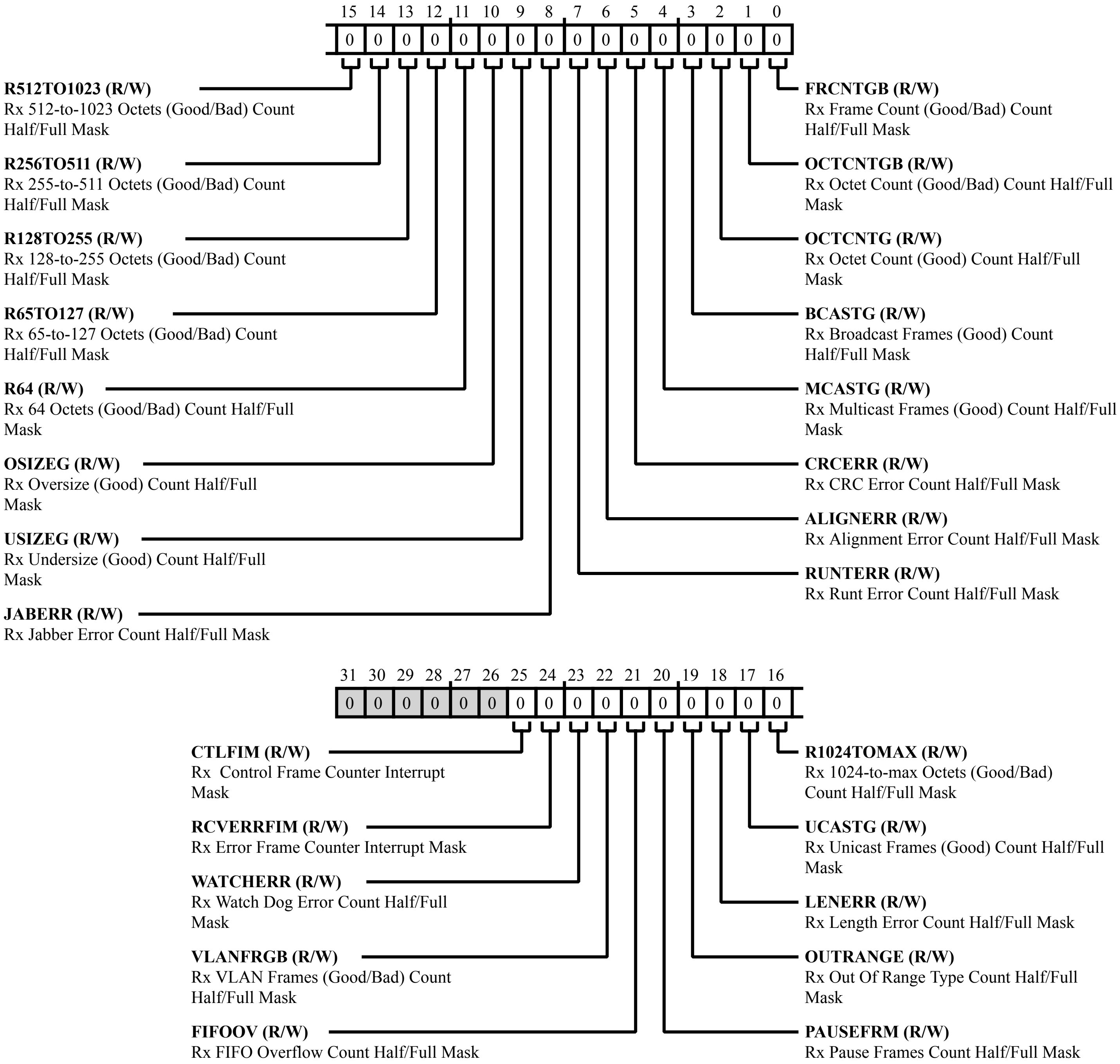

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
R512TO1023 (R/W)
Rx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask
R256TO511 (R/W)
Rx 255-to-511 Octets (Good/Bad) Count Half/Full Mask
R128TO255 (R/W)
Rx 128-to-255 Octets (Good/Bad) Count Half/Full Mask
R65TO127 (R/W)
Rx 65-to-127 Octets (Good/Bad) Count Half/Full Mask
R64 (R/W)
Rx 64 Octets (Good/Bad) Count Half/Full Mask
OSIZEG (R/W)
Rx Oversize (Good) Count Half/Full Mask
USIZEG (R/W)
Rx Undersize (Good) Count Half/Full Mask
JABERR (R/W)
Rx Jabber Error Count Half/Full Mask
FRCNTGB (R/W)
Rx Frame Count (Good/Bad) Count Half/Full Mask
OCTCNTGB (R/W)
Rx Octet Count (Good/Bad) Count Half/Full Mask
OCTCNTG (R/W)
Rx Octet Count (Good) Count Half/Full Mask
BCASTG (R/W)
Rx Broadcast Frames (Good) Count Half/Full Mask
MCASTG (R/W)
Rx Multicast Frames (Good) Count Half/Full Mask
CRCERR (R/W)
Rx CRC Error Count Half/Full Mask
ALIGNERR (R/W)
Rx Alignment Error Count Half/Full Mask
RUNTERR (R/W)
Rx Runt Error Count Half/Full Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CTLFIM (R/W)
Rx Control Frame Counter Interrupt Mask
RCVERRFIM (R/W)
Rx Error Frame Counter Interrupt Mask
WATCHERR (R/W)
Rx Watch Dog Error Count Half/Full Mask
VLANFRGB (R/W)
Rx VLAN Frames (Good/Bad) Count Half/Full Mask
FIFOOV (R/W)
Rx FIFO Overflow Count Half/Full Mask
R1024TOMAX (R/W)
Rx 1024-to-max Octets (Good/Bad) Count Half/Full Mask
UCASTG (R/W)
Rx Unicast Frames (Good) Count Half/Full Mask
LENERR (R/W)
Rx Length Error Count Half/Full Mask
OUTRANGE (R/W)
Rx Out Of Range Type Count Half/Full Mask
PAUSEFRM (R/W)
Rx Pause Frames Count Half/Full Mask

Figure 31-106: EMAC_MMC_RXIMSK Register Diagram

Table 31-138: EMAC_MMC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	CTLFIM	Rx Control Frame Counter Interrupt Mask. The `EMAC_MMC_RXIMSK.CTLFIM` bit, masks the interrupt when the rxctrlframes_g counter reaches half of the maximum value or the maximum value.
24 (R/W)	RCVERRFIM	Rx Error Frame Counter Interrupt Mask. The `EMAC_MMC_RXIMSK.RCVERRFIM` bit,masks the interrupt when the rxrcverror counter reaches half of the maximum value or the maximum value.
23 (R/W)	WATCHERR	Rx Watch Dog Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.WATCHERR` bit, when set, masks the interrupt when `EMAC_RXWDOG_ERR` counter reaches full or half.
22 (R/W)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.VLANFRGB` bit, when set, masks the interrupt when `EMAC_RXVLANFRM_GB` counter reaches full or half.
21 (R/W)	FIFOOV	Rx FIFO Overflow Count Half/Full Mask. The `EMAC_MMC_RXIMSK.FIFOOV` bit, when set, masks the interrupt when `EMAC_RXFIFO_OVF` counter reaches full or half.
20 (R/W)	PAUSEFRM	Rx Pause Frames Count Half/Full Mask. The `EMAC_MMC_RXIMSK.PAUSEFRM` bit, when set, masks the interrupt when `EMAC_RXPAUSEFRM` counter reaches full or half.
19 (R/W)	OUTRANGE	Rx Out Of Range Type Count Half/Full Mask. The `EMAC_MMC_RXIMSK.OUTRANGE` bit, when set, masks the interrupt when `EMAC_RXOORTYPE` counter reaches full or half.
18 (R/W)	LENERR	Rx Length Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.LENERR` bit, when set, masks the interrupt when `EMAC_RXLEN_ERR` counter reaches full or half.
17 (R/W)	UCASTG	Rx Unicast Frames (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.UCASTG` bit, when set, masks the interrupt when `EMAC_RXUCASTFRM_G` counter reaches full or half.
16 (R/W)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R1024TOMAX` bit, when set, masks the interrupt when `EMAC_RX1024TOMAX_GB` counter reaches full or half.
15 (R/W)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R512TO1023` bit, when set, masks the interrupt when `EMAC_RX512TO1023_GB` counter reaches full or half.
14 (R/W)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R256TO511` bit, when set, masks the interrupt when `EMAC_RX256TO511_GB` counter reaches full or half.

Table 31-138: EMAC_MMC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R128TO255` bit, when set, masks the interrupt when `EMAC_RX128TO255_GB` counter reaches full or half.
12 (R/W)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R65TO127` bit, when set, masks the interrupt when `EMAC_RX65TO127_GB` counter reaches full or half.
11 (R/W)	R64	Rx 64 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.R64` bit, when set, masks the interrupt when `EMAC_RX64_GB` counter reaches full or half.
10 (R/W)	OSIZEG	Rx Oversize (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.OSIZEG` bit, when set, masks the interrupt when `EMAC_RXOSIZE_G` counter reaches full or half.
9 (R/W)	USIZEG	Rx Undersize (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.USIZEG` bit, when set, masks the interrupt when `EMAC_RXUSIZE_G` counter reaches full or half.
8 (R/W)	JABERR	Rx Jabber Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.JABERR` bit, when set, masks the interrupt when `EMAC_RXJAB_ERR` counter reaches full or half.
7 (R/W)	RUNTERR	Rx Runt Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.RUNTERR` bit, when set, masks the interrupt when `EMAC_RXRUNT_ERR` counter reaches full or half.
6 (R/W)	ALIGNERR	Rx Alignment Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.ALIGNERR` bit, when set, masks the interrupt when `EMAC_RXALIGN_ERR` counter reaches full or half.
5 (R/W)	CRCERR	Rx CRC Error Count Half/Full Mask. The `EMAC_MMC_RXIMSK.CRCERR` bit, when set, masks the interrupt when `EMAC_RXCRC_ERR` counter reaches full or half.
4 (R/W)	MCASTG	Rx Multicast Frames (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.MCASTG` bit, when set, masks the interrupt when `EMAC_RXMCASTFRM_G` counter reaches full or half.
3 (R/W)	BCASTG	Rx Broadcast Frames (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.BCASTG` bit, when set, masks the interrupt when `EMAC_RXBCASTFRM_G` counter reaches full or half.

Table 31-138: EMAC_MMC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	OCTCNTG	Rx Octet Count (Good) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.OCTCNTG` bit, when set, masks the interrupt when `EMAC_RXOCTCNT_G` counter reaches full or half.
1 (R/W)	OCTCNTGB	Rx Octet Count (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.OCTCNTGB` bit, when set, masks the interrupt when `EMAC_RXOCTCNT_GB` counter reaches half or full.
0 (R/W)	FRCNTGB	Rx Frame Count (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_RXIMSK.FRCNTGB` bit, when set, masks the interrupt when `EMAC_RXFRMCNT_GB` counter reaches half or full.

MMC Rx Interrupt Register

The `EMAC_MMC_RXINT` register indicates status of MMC receive interrupts.

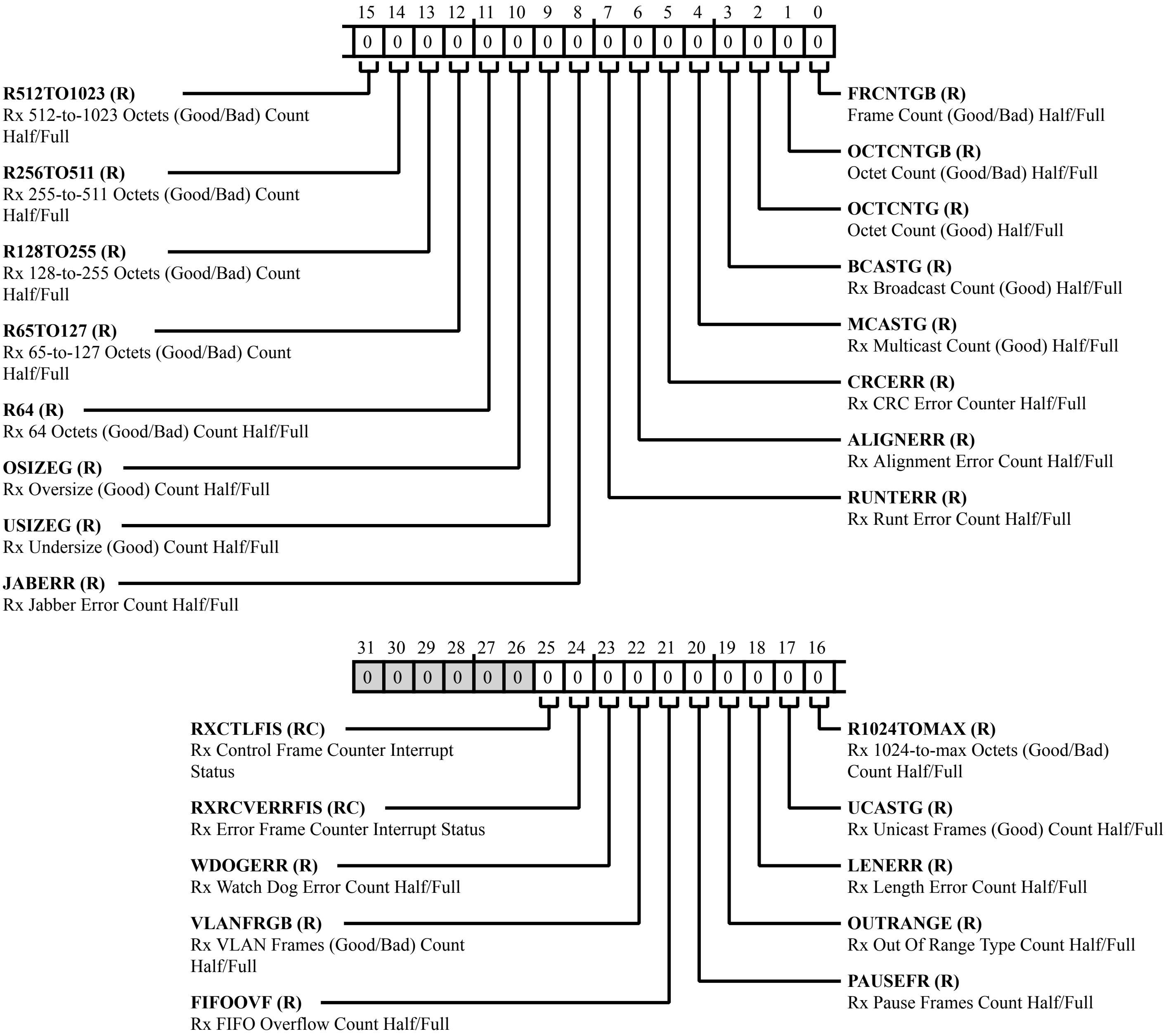

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
R512TO1023 (R)
Rx 512-to-1023 Octets (Good/Bad) Count Half/Full
R256TO511 (R)
Rx 255-to-511 Octets (Good/Bad) Count Half/Full
R128TO255 (R)
Rx 128-to-255 Octets (Good/Bad) Count Half/Full
R65TO127 (R)
Rx 65-to-127 Octets (Good/Bad) Count Half/Full
R64 (R)
Rx 64 Octets (Good/Bad) Count Half/Full
OSIZEG (R)
Rx Oversize (Good) Count Half/Full
USIZEG (R)
Rx Undersize (Good) Count Half/Full
JABERR (R)
Rx Jabber Error Count Half/Full
FRCNTGB (R)
Frame Count (Good/Bad) Half/Full
OCTCNTGB (R)
Octet Count (Good/Bad) Half/Full
OCTCNTG (R)
Octet Count (Good) Half/Full
BCASTG (R)
Rx Broadcast Count (Good) Half/Full
MCASTG (R)
Rx Multicast Count (Good) Half/Full
CRCERR (R)
Rx CRC Error Counter Half/Full
ALIGNERR (R)
Rx Alignment Error Count Half/Full
RUNTERR (R)
Rx Runt Error Count Half/Full
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXCTLFIS (RC)
Rx Control Frame Counter Interrupt Status
RXRCVERRFIS (RC)
Rx Error Frame Counter Interrupt Status
WDOGERR (R)
Rx Watch Dog Error Count Half/Full
VLANFRGB (R)
Rx VLAN Frames (Good/Bad) Count Half/Full
FIFOOVF (R)
Rx FIFO Overflow Count Half/Full
R1024TOMAX (R)
Rx 1024-to-max Octets (Good/Bad) Count Half/Full
UCASTG (R)
Rx Unicast Frames (Good) Count Half/Full
LENERR (R)
Rx Length Error Count Half/Full
OUTRANGE (R)
Rx Out Of Range Type Count Half/Full
PAUSEFR (R)
Rx Pause Frames Count Half/Full

Figure 31-107: EMAC_MMC_RXINT Register Diagram

Table 31-139: EMAC_MMC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (RC/NW)	RXCTLFIS	Rx Control Frame Counter Interrupt Status. The `EMAC_MMC_RXINT.RXCTLFIS` bit is set when the rxctrlframes_g counter reaches half of the maximum value or the maximum value.
24 (RC/NW)	RXRCVERRFIS	Rx Error Frame Counter Interrupt Status. The `EMAC_MMC_RXINT.RXRCVERRFIS` bit is set when the rxrcverror counter reaches half of the maximum value or the maximum value.
23 (R/NW)	WDOGERR	Rx Watch Dog Error Count Half/Full. The `EMAC_MMC_RXINT.WDOGERR` bit is set when the `EMAC_RXWDOG_ERR` counter reaches full or half.
22 (R/NW)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.VLANFRGB` bit is set when `EMAC_RXVLANFRM_GB` counter reaches full or half.
21 (R/NW)	FIFOOVF	Rx FIFO Overflow Count Half/Full. The `EMAC_MMC_RXINT.FIFOOVF` bit is set when `EMAC_RXFIFO_OVF` counter reaches full or half.
20 (R/NW)	PAUSEFR	Rx Pause Frames Count Half/Full. The `EMAC_MMC_RXINT.PAUSEFR` bit is set when `EMAC_RXPAUSEFRM` counter reaches full or half.
19 (R/NW)	OUTRANGE	Rx Out Of Range Type Count Half/Full. The `EMAC_MMC_RXINT.OUTRANGE` bit is set when `EMAC_RXOORTYPE` counter reaches full or half.
18 (R/NW)	LENERR	Rx Length Error Count Half/Full. The `EMAC_MMC_RXINT.LENERR` bit is set when `EMAC_RXLEN_ERR` counter reaches full or half.
17 (R/NW)	UCASTG	Rx Unicast Frames (Good) Count Half/Full. The `EMAC_MMC_RXINT.UCASTG` bit is set when `EMAC_RXUCASTFRM_G` counter reaches full or half.
16 (R/NW)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R1024TOMAX` bit is set when `EMAC_RX1024TOMAX_GB` counter reaches full or half.
15 (R/NW)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R512TO1023` bit is set when `EMAC_RX512TO1023_GB` counter reaches full or half.
14 (R/NW)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R256TO511` bit is set when `EMAC_RX256TO511_GB` counter reaches full or half.

Table 31-139: EMAC_MMC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/NW)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R128TO255` bit is set when `EMAC_RX128TO255_GB` counter reaches full or half.
12 (R/NW)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R65TO127` bit is set when `EMAC_RX65TO127_GB` counter reaches full or half.
11 (R/NW)	R64	Rx 64 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_RXINT.R64` bit is set when `EMAC_RX64_GB` counter reaches full or half.
10 (R/NW)	OSIZEG	Rx Oversize (Good) Count Half/Full. The `EMAC_MMC_RXINT.OSIZEG` bit is set when `EMAC_RXOSIZE_G` counter reaches full or half.
9 (R/NW)	USIZEG	Rx Undersize (Good) Count Half/Full. The `EMAC_MMC_RXINT.USIZEG` bit is set when `EMAC_RXUSIZE_G` counter reaches full or half.
8 (R/NW)	JABERR	Rx Jabber Error Count Half/Full. The `EMAC_MMC_RXINT.JABERR` bit is set when `EMAC_RXJAB_ERR` counter reaches full or half.
7 (R/NW)	RUNTERR	Rx Runt Error Count Half/Full. The `EMAC_MMC_RXINT.RUNTERR` bit is set when `EMAC_RXRUNT_ERR` counter reaches full or half.
6 (R/NW)	ALIGNERR	Rx Alignment Error Count Half/Full. The `EMAC_MMC_RXINT.ALIGNERR` bit is set when `EMAC_RXALIGN_ERR` counter reaches full or half
5 (R/NW)	CRCERR	Rx CRC Error Counter Half/Full. The `EMAC_MMC_RXINT.CRCERR` bit is set when `EMAC_RXCRC_ERR` counter reaches full or half.
4 (R/NW)	MCASTG	Rx Multicast Count (Good) Half/Full. The `EMAC_MMC_RXINT.MCASTG` bit is set when `EMAC_RXMCASTFRM_G` counter reaches full or half.
3 (R/NW)	BCASTG	Rx Broadcast Count (Good) Half/Full. The `EMAC_MMC_RXINT.BCASTG` bit is set when `EMAC_RXBCASTFRM_G` counter reaches full or half.

Table 31-139: EMAC_MMC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	OCTCNTG	Octet Count (Good) Half/Full. The `EMAC_MMC_RXINT.OCTCNTG` bit is set when `EMAC_RXOCTCNT_G` counter reaches full or half.
1 (R/NW)	OCTCNTGB	Octet Count (Good/Bad) Half/Full. The `EMAC_MMC_RXINT.OCTCNTGB` bit is set when `EMAC_RXOCTCNT_GB` counter reaches half or full.
0 (R/NW)	FRCNTGB	Frame Count (Good/Bad) Half/Full. The `EMAC_MMC_RXINT.FRCNTGB` bit is set when `EMAC_RXFRMCNT_GB` counter reaches half or full.

MMC TX Interrupt Mask Register

The EMAC_MMC_TXIMSK register enables (unmasks) MMC transmit interrupts.

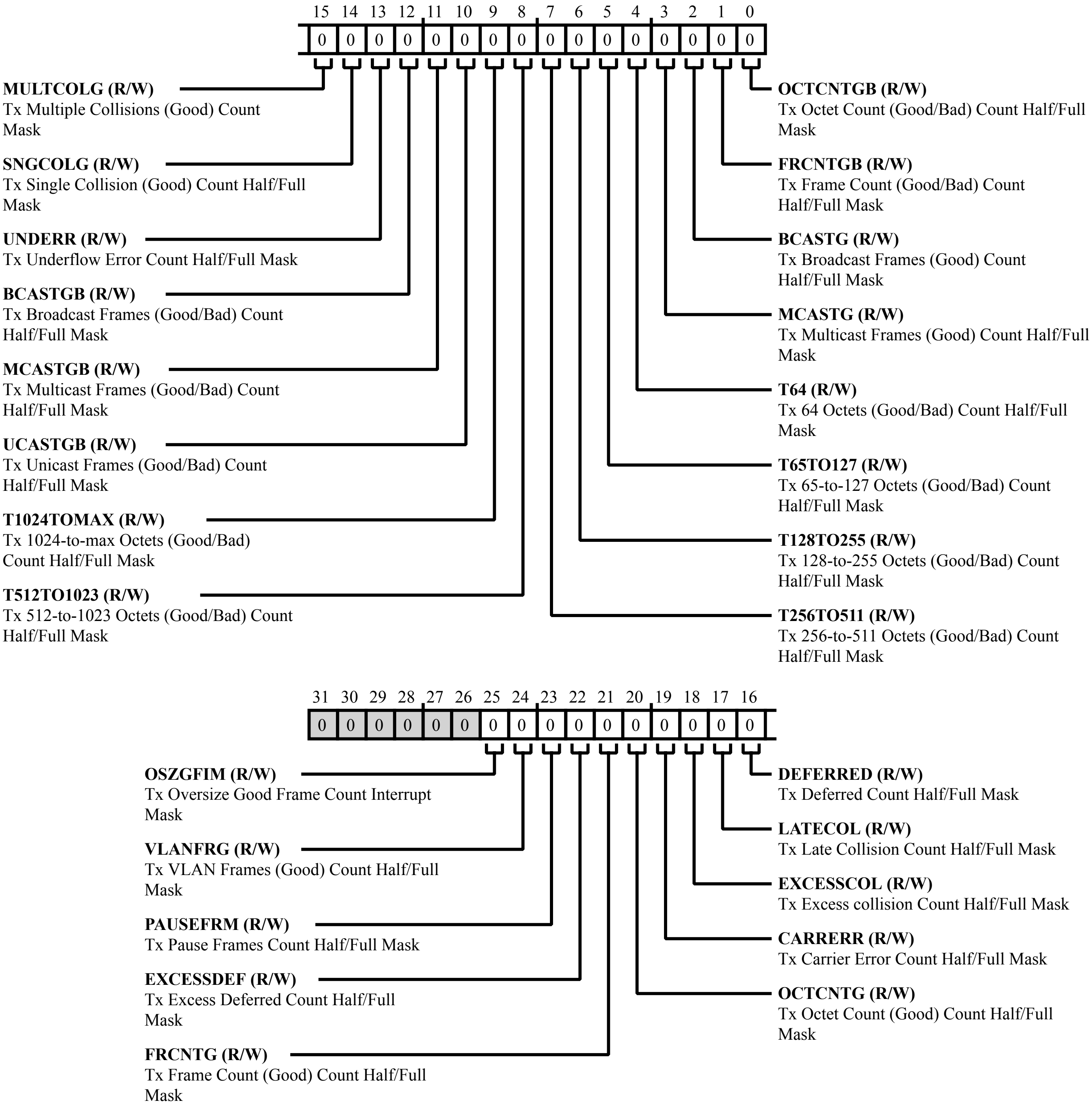

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MULTCOLG (R/W)
Tx Multiple Collisions (Good) Count Mask
SNGCOLG (R/W)
Tx Single Collision (Good) Count Half/Full Mask
UNDERR (R/W)
Tx Underflow Error Count Half/Full Mask
BCASTGB (R/W)
Tx Broadcast Frames (Good/Bad) Count Half/Full Mask
MCASTGB (R/W)
Tx Multicast Frames (Good/Bad) Count Half/Full Mask
UCASTGB (R/W)
Tx Unicast Frames (Good/Bad) Count Half/Full Mask
T1024TOMAX (R/W)
Tx 1024-to-max Octets (Good/Bad) Count Half/Full Mask
T512TO1023 (R/W)
Tx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask
OCTCNTGB (R/W)
Tx Octet Count (Good/Bad) Count Half/Full Mask
FRCNTGB (R/W)
Tx Frame Count (Good/Bad) Count Half/Full Mask
BCASTG (R/W)
Tx Broadcast Frames (Good) Count Half/Full Mask
MCASTG (R/W)
Tx Multicast Frames (Good) Count Half/Full Mask
T64 (R/W)
Tx 64 Octets (Good/Bad) Count Half/Full Mask
T65TO127 (R/W)
Tx 65-to-127 Octets (Good/Bad) Count Half/Full Mask
T128TO255 (R/W)
Tx 128-to-255 Octets (Good/Bad) Count Half/Full Mask
T256TO511 (R/W)
Tx 256-to-511 Octets (Good/Bad) Count Half/Full Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OSZGFIM (R/W)
Tx Oversize Good Frame Count Interrupt Mask
VLANFRG (R/W)
Tx VLAN Frames (Good) Count Half/Full Mask
PAUSEFRM (R/W)
Tx Pause Frames Count Half/Full Mask
EXCESSDEF (R/W)
Tx Excess Deferred Count Half/Full Mask
FRCNTG (R/W)
Tx Frame Count (Good) Count Half/Full Mask
DEFERRED (R/W)
Tx Deferred Count Half/Full Mask
LATECOL (R/W)
Tx Late Collision Count Half/Full Mask
EXCESSCOL (R/W)
Tx Excess collision Count Half/Full Mask
CARRERR (R/W)
Tx Carrier Error Count Half/Full Mask
OCTCNTG (R/W)
Tx Octet Count (Good) Count Half/Full Mask

Figure 31-108: EMAC_MMC_TXIMSK Register Diagram

Table 31-140: EMAC_MMC_TXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	OSZGFIM	Tx Oversize Good Frame Count Interrupt Mask. The `EMAC_MMC_TXIMSK.OSZGFIM` bit masks the interrupt when the txoversize_g counter reaches half of the maximum value or the maximum value.
24 (R/W)	VLANFRG	Tx VLAN Frames (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.VLANFRG` bit, when set, masks the interrupt when `EMAC_TXVLANFRM_G` counter reaches full or half.
23 (R/W)	PAUSEFRM	Tx Pause Frames Count Half/Full Mask. The `EMAC_MMC_TXIMSK.PAUSEFRM` bit, when set, masks the interrupt when `EMAC_TXPAUSEFRM` counter reaches full or half.
22 (R/W)	EXCESSDEF	Tx Excess Deferred Count Half/Full Mask. The `EMAC_MMC_TXIMSK.EXCESSDEF` bit, when set, masks the interrupt when `EMAC_TXEXCESSDEF` counter reaches full or half.
21 (R/W)	FRCNTG	Tx Frame Count (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.FRCNTG` bit, when set, masks the interrupt when `EMAC_TXFRMCNT_G` counter reaches full or half.
20 (R/W)	OCTCNTG	Tx Octet Count (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.OCTCNTG` bit, when set, masks the interrupt when `EMAC_TXOCTCNT_G` counter reaches full or half.
19 (R/W)	CARRERR	Tx Carrier Error Count Half/Full Mask. The `EMAC_MMC_TXIMSK.CARRERR` bit, when set, masks the interrupt when `EMAC_TXCARR_ERR` counter reaches full or half.
18 (R/W)	EXCESSCOL	Tx Excess collision Count Half/Full Mask. The `EMAC_MMC_TXIMSK.EXCESSCOL` bit, when set, masks the interrupt when `EMAC_TXEXCESSCOL` counter reaches full or half.
17 (R/W)	LATECOL	Tx Late Collision Count Half/Full Mask. The `EMAC_MMC_TXIMSK.LATECOL` bit, when set, masks the interrupt when `EMAC_TXLATECOL` counter reaches full or half.
16 (R/W)	DEFERRED	Tx Deferred Count Half/Full Mask. The `EMAC_MMC_TXIMSK.DEFERRED` bit, when set, masks the interrupt when `EMAC_TXDEFERRED` counter reaches full or half.
15 (R/W)	MULTCOLG	Tx Multiple Collisions (Good) Count Mask. The `EMAC_MMC_TXIMSK.MULTCOLG` bit, when set, masks the interrupt when `EMAC_TXMULTCOL_G` counter reaches full or half.
14 (R/W)	SNGCOLG	Tx Single Collision (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.SNGCOLG` bit, when set, masks the interrupt when `EMAC_TXSNGCOL_G` counter reaches full or half.

Table 31-140: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	UNDERR	Tx Underflow Error Count Half/Full Mask. The `EMAC_MMC_TXIMSK.UNDERR` bit, when set, masks the interrupt when `EMAC_TXUNDR_ERR` counter reaches full or half.
12 (R/W)	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.BCASTGB` bit, when set, masks the interrupt when `EMAC_TXBCASTFRM_GB` counter reaches full or half.
11 (R/W)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.MCASTGB` bit, when set, masks the interrupt when `EMAC_TXMCASTFRM_GB` counter reaches full or half.
10 (R/W)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.UCASTGB` bit, when set, masks the interrupt when `EMAC_TXUCASTFRM_GB` counter reaches full or half.
9 (R/W)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T1024TOMAX` bit, when set, masks the interrupt when `EMAC_TX1024TOMAX_GB` counter reaches full or half.
8 (R/W)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T512TO1023` bit, when set, masks the interrupt when `EMAC_TX512TO1023_GB` counter reaches full or half.
7 (R/W)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T256TO511` bit, when set, masks the interrupt when `EMAC_TX256TO511_GB` counter reaches full or half.
6 (R/W)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T128TO255` bit, when set, masks the interrupt when `EMAC_TX128TO255_GB` counter reaches full or half.
5 (R/W)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T65TO127` bit, when set, masks the interrupt when `EMAC_TX65TO127_GB` counter reaches full or half.
4 (R/W)	T64	Tx 64 Octets (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.T64` bit, when set, masks the interrupt when `EMAC_TX64_GB` counter reaches full or half.
3 (R/W)	MCASTG	Tx Multicast Frames (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.MCASTG` bit, when set, masks the interrupt when `EMAC_TXMCASTFRM_G` counter reaches full or half.

Table 31-140: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.BCASTG` bit, when set, masks the interrupt when `EMAC_TXBCASTFRM_G` counter reaches full or half.
1 (R/W)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.FRCNTGB` bit, when set, masks the interrupt when `EMAC_TXFRMCNT_GB` counter reaches full or half.
0 (R/W)	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full Mask. The `EMAC_MMC_TXIMSK.OCTCNTGB` bit, when set, masks the interrupt when `EMAC_TXOCTCNT_GB` counter reaches full or half.

MMC Tx Interrupt Register

The `EMAC_MMC_TXINT` register indicates status of MMC transmit interrupts.

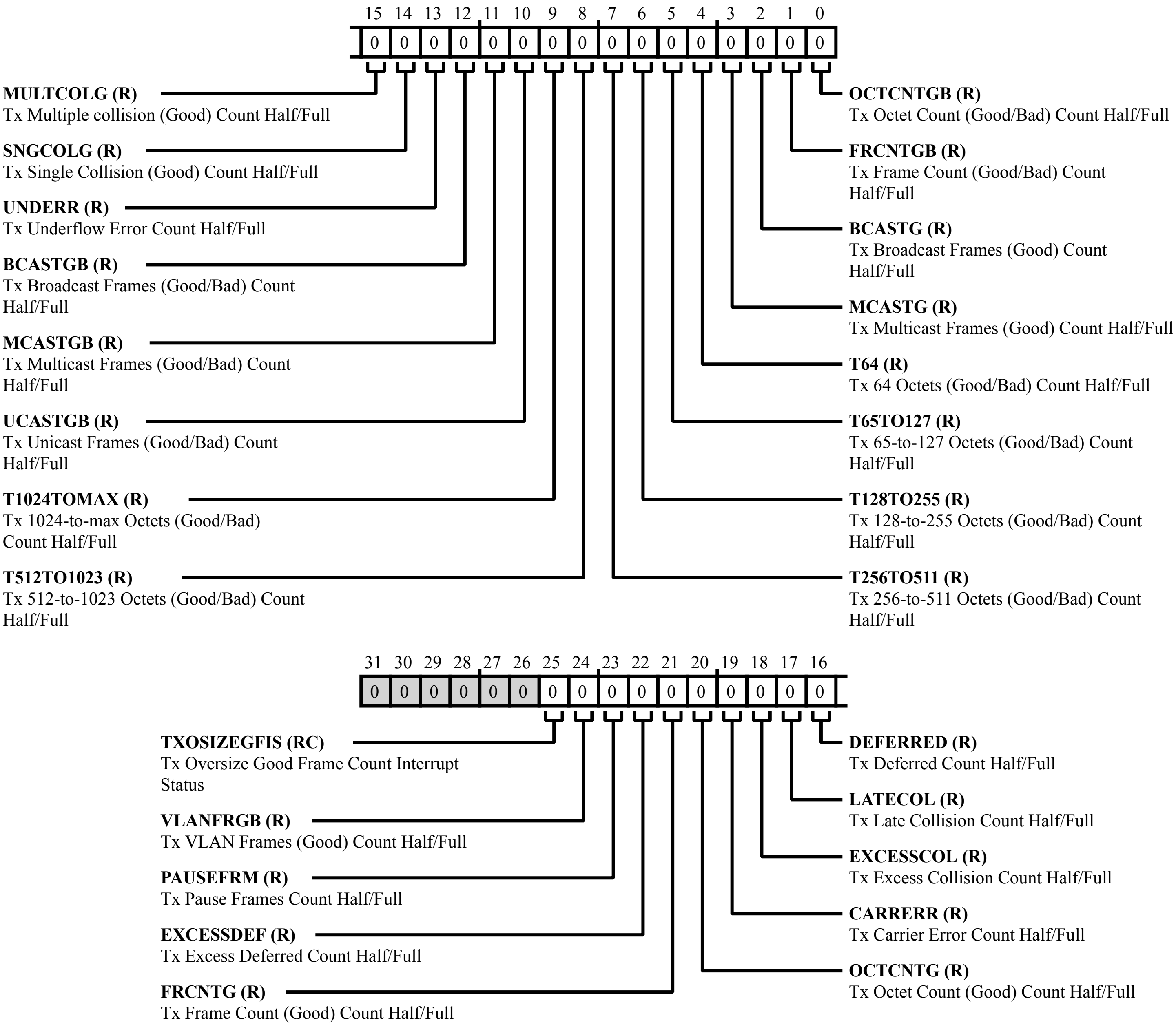

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MULTCOLG (R)
Tx Multiple collision (Good) Count Half/Full
SNGCOLG (R)
Tx Single Collision (Good) Count Half/Full
UNDERR (R)
Tx Underflow Error Count Half/Full
BCASTGB (R)
Tx Broadcast Frames (Good/Bad) Count Half/Full
MCASTGB (R)
Tx Multicast Frames (Good/Bad) Count Half/Full
UCASTGB (R)
Tx Unicast Frames (Good/Bad) Count Half/Full
T1024TOMAX (R)
Tx 1024-to-max Octets (Good/Bad) Count Half/Full
T512TO1023 (R)
Tx 512-to-1023 Octets (Good/Bad) Count Half/Full
OCTCNTGB (R)
Tx Octet Count (Good/Bad) Count Half/Full
FRCNTGB (R)
Tx Frame Count (Good/Bad) Count Half/Full
BCASTG (R)
Tx Broadcast Frames (Good) Count Half/Full
MCASTG (R)
Tx Multicast Frames (Good) Count Half/Full
T64 (R)
Tx 64 Octets (Good/Bad) Count Half/Full
T65TO127 (R)
Tx 65-to-127 Octets (Good/Bad) Count Half/Full
T128TO255 (R)
Tx 128-to-255 Octets (Good/Bad) Count Half/Full
T256TO511 (R)
Tx 256-to-511 Octets (Good/Bad) Count Half/Full
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TXOSIZEGFIS (RC)
Tx Oversize Good Frame Count Interrupt Status
VLANFRGB (R)
Tx VLAN Frames (Good) Count Half/Full
PAUSEFRM (R)
Tx Pause Frames Count Half/Full
EXCESSDEF (R)
Tx Excess Deferred Count Half/Full
FRCNTG (R)
Tx Frame Count (Good) Count Half/Full
DEFERRED (R)
Tx Deferred Count Half/Full
LATECOL (R)
Tx Late Collision Count Half/Full
EXCESSCOL (R)
Tx Excess Collision Count Half/Full
CARRERR (R)
Tx Carrier Error Count Half/Full
OCTCNTG (R)
Tx Octet Count (Good) Count Half/Full

Figure 31-109: EMAC_MMC_TXINT Register Diagram

Table 31-141: EMAC_MMC_TXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (RC/NW)	TXOSIZEGFIS	Tx Oversize Good Frame Count Interrupt Status. The `EMAC_MMC_TXINT.TXOSIZEGFIS` bit is set when the txoversize_g counter reaches half of the maximum value or the maximum value.
24 (R/NW)	VLANFRGB	Tx VLAN Frames (Good) Count Half/Full. The `EMAC_MMC_TXINT.VLANFRGB` bit is set when `EMAC_TXVLANFRM_G` counter reaches full or half.
23 (R/NW)	PAUSEFRM	Tx Pause Frames Count Half/Full. The `EMAC_MMC_TXINT.PAUSEFRM` bit is set when `EMAC_TXPAUSEFRM` counter reaches full or half.
22 (R/NW)	EXCESSDEF	Tx Excess Deferred Count Half/Full. The `EMAC_MMC_TXINT.EXCESSDEF` bit is set when `EMAC_TXEXCESSDEF` counter reaches full or half.
21 (R/NW)	FRCNTG	Tx Frame Count (Good) Count Half/Full. The `EMAC_MMC_TXINT.FRCNTG` bit is set when `EMAC_TXFRMCNT_G` counter reaches full or half.
20 (R/NW)	OCTCNTG	Tx Octet Count (Good) Count Half/Full. The `EMAC_MMC_TXINT.OCTCNTG` bit is set when `EMAC_TXOCTCNT_G` counter reaches full or half.
19 (R/NW)	CARRERR	Tx Carrier Error Count Half/Full. The `EMAC_MMC_TXINT.CARRERR` bit is set when `EMAC_TXCARR_ERR` counter reaches full or half.
18 (R/NW)	EXCESSCOL	Tx Excess Collision Count Half/Full. The `EMAC_MMC_TXINT.EXCESSCOL` bit is set when `EMAC_TXEXCESSCOL` counter reaches full or half.
17 (R/NW)	LATECOL	Tx Late Collision Count Half/Full. The `EMAC_MMC_TXINT.LATECOL` bit is set when `EMAC_TXLATECOL` counter reaches full or half.
16 (R/NW)	DEFERRED	Tx Deferred Count Half/Full. The `EMAC_MMC_TXINT.DEFERRED` bit is set when `EMAC_TXDEFERRED` counter reaches full or half.
15 (R/NW)	MULTCOLG	Tx Multiple collision (Good) Count Half/Full. The `EMAC_MMC_TXINT.MULTCOLG` bit is set when `EMAC_TXMULTCOL_G` counter reaches full or half.
14 (R/NW)	SNGCOLG	Tx Single Collision (Good) Count Half/Full. The `EMAC_MMC_TXINT.SNGCOLG` bit is set when `EMAC_TXSNGCOL_G` counter reaches full or half.

Table 31-141: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/NW)	UNDERR	Tx Underflow Error Count Half/Full. The `EMAC_MMC_TXINT.UNDERR` bit is set when `EMAC_TXUNDR_ERR` counter reaches full or half.
12 (R/NW)	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.BCASTGB` bit is set when `EMAC_TXBCASTFRM_GB` counter reaches full or half.
11 (R/NW)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.MCASTGB` bit is set when `EMAC_TXMCASTFRM_GB` counter reaches full or half.
10 (R/NW)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.UCASTGB` bit is set when `EMAC_TXUCASTFRM_GB` counter reaches full or half.
9 (R/NW)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T1024TOMAX` bit is set when `EMAC_TX1024TOMAX_GB` counter reaches full or half.
8 (R/NW)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T512TO1023` bit is set when `EMAC_TX512TO1023_GB` counter reaches full or half.
7 (R/NW)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T256TO511` bit is set when `EMAC_TX256TO511_GB` counter reaches full or half.
6 (R/NW)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T128TO255` bit is set when `EMAC_TX128TO255_GB` counter reaches full or half.
5 (R/NW)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T65TO127` bit is set when `EMAC_TX65TO127_GB` counter reaches full or half.
4 (R/NW)	T64	Tx 64 Octets (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.T64` bit is set when `EMAC_TX64_GB` counter reaches full or half.
3 (R/NW)	MCASTG	Tx Multicast Frames (Good) Count Half/Full. The `EMAC_MMC_TXINT.MCASTG` bit is set when `EMAC_TXMCASTFRM_G` counter reaches full or half.

Table 31-141: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full. The `EMAC_MMC_TXINT.BCASTG` bit is set when `EMAC_TXBCASTFRM_G` counter reaches full or half.
1 (R/NW)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.FRCNTGB` bit is set when `EMAC_TXFRMCNT_GB` counter reaches full or half.
0 (R/NW)	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full. The `EMAC_MMC_TXINT.OCTCNTGB` bit is set when `EMAC_TXOCTCNT_GB` counter reaches full or half.

Rx 1024- to Max-Byte Frames (Good/Bad) Register

The EMAC_RX1024TOMAX_GB register contains a count of the number of good and bad frames received with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble.

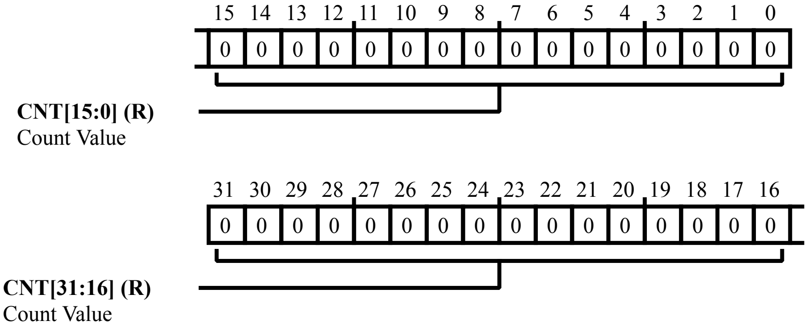

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-110: EMAC_RX1024TOMAX_GB Register Diagram

Table 31-142: EMAC_RX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 128- to 255-Byte Frames (Good/Bad) Register

The `EMAC_RX128TO255_GB` register contains a count of the number of good and bad frames received with length between 128 and 255 (inclusive) bytes, exclusive of preamble.

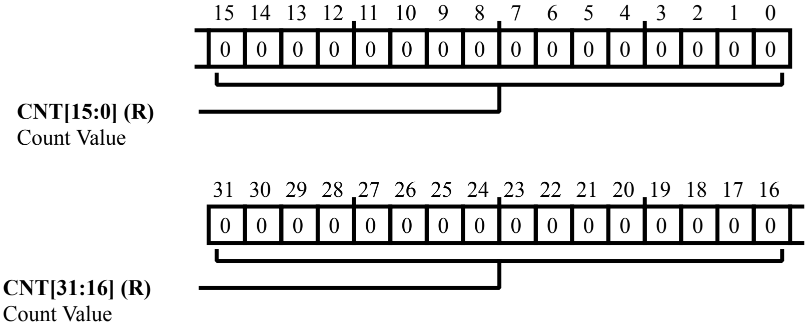

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

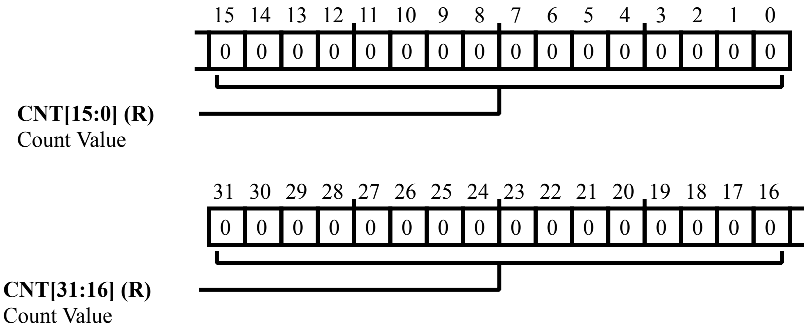
Figure 31-111: EMAC_RX128TO255_GB Register Diagram

Table 31-143: EMAC_RX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 256- to 511-Byte Frames (Good/Bad) Register

The `EMAC_RX256TO511_GB` register contains a count of the number of good and bad frames received with length between 256 and 511 (inclusive) bytes, exclusive of preamble.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-112: EMAC_RX256TO511_GB Register Diagram

Table 31-144: EMAC_RX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 512- to 1023-Byte Frames (Good/Bad) Register

The `EMAC_RX512TO1023_GB` register contains a count of the number of good and bad frames received with length between 512 and 1023 (inclusive) bytes, exclusive of preamble.

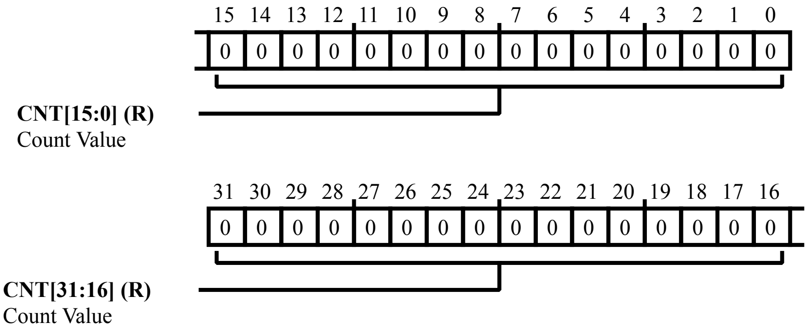

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-113: EMAC_RX512TO1023_GB Register Diagram

Table 31-145: EMAC_RX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 64-Byte Frames (Good/Bad) Register

The `EMAC_RX64_GB` register contains a count of the number of good and bad frames received with length 64 bytes, exclusive of preamble.

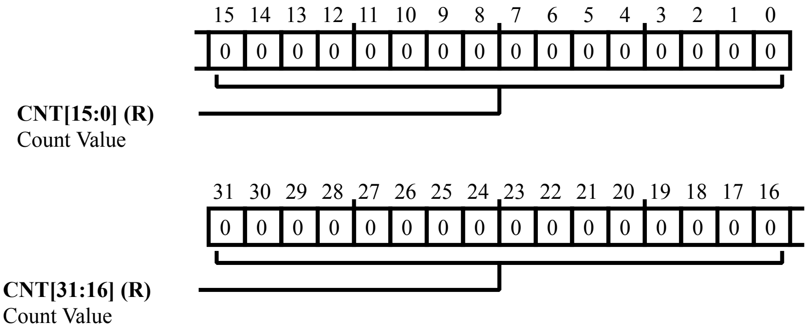

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-114: EMAC_RX64_GB Register Diagram

Table 31-146: EMAC_RX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 65- to 127-Byte Frames (Good/Bad) Register

The `EMAC_RX65TO127_GB` register contains a count of the number of good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.

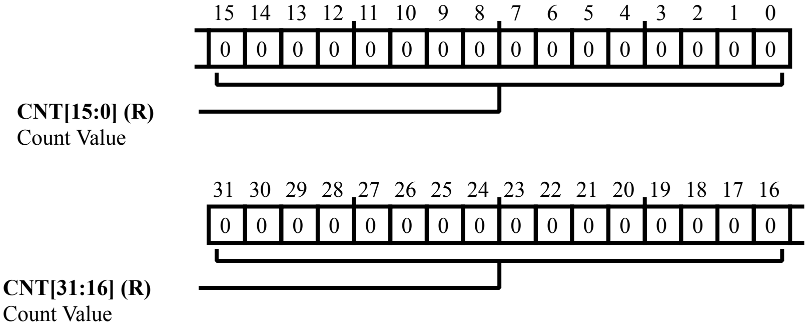

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-115: EMAC_RX65TO127_GB Register Diagram

Table 31-147: EMAC_RX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx alignment Error Register

The `EMAC_RXALIGN_ERR` register contains a count of the number of frames received with alignment (dribble) error. Valid only in 10/100 mode.

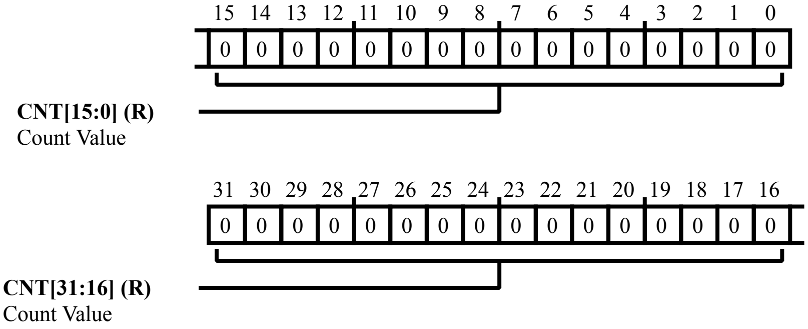

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-116: EMAC_RXALIGN_ERR Register Diagram

Table 31-148: EMAC_RXALIGN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Broadcast Frames (Good) Register

The `EMAC_RXBCASTFRM_G` register contains a count of the number of good broadcast frames received.

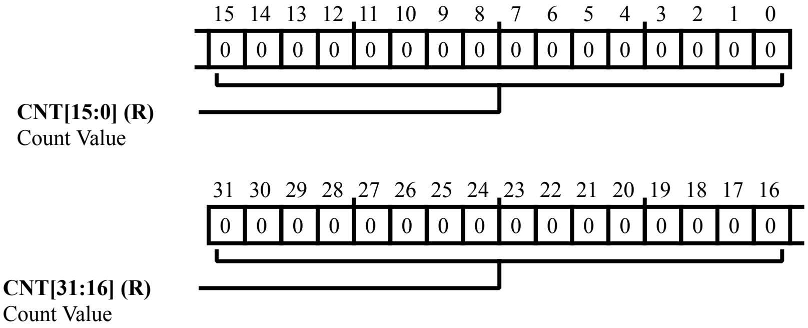

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-117: EMAC_RXBCASTFRM_G Register Diagram

Table 31-149: EMAC_RXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx CRC Error Register

The EMAC_RXCRC_ERR register contains a count of the number of frames received with CRC error.

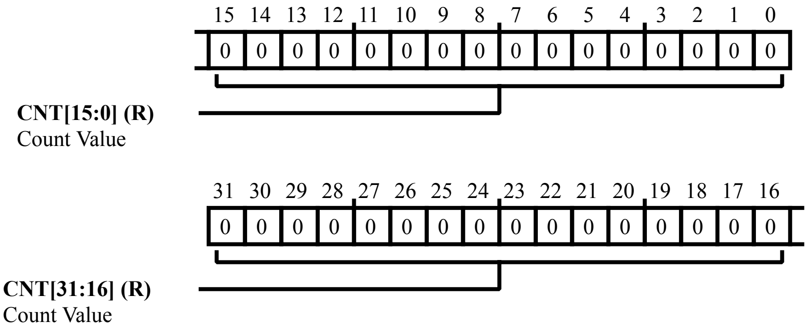

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-118: EMAC_RXCRC_ERR Register Diagram

Table 31-150: EMAC_RXCRC_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Good Control Frames Register

The `EMAC_RXCTLFRM_G` register contains a count of the number of good control frames received

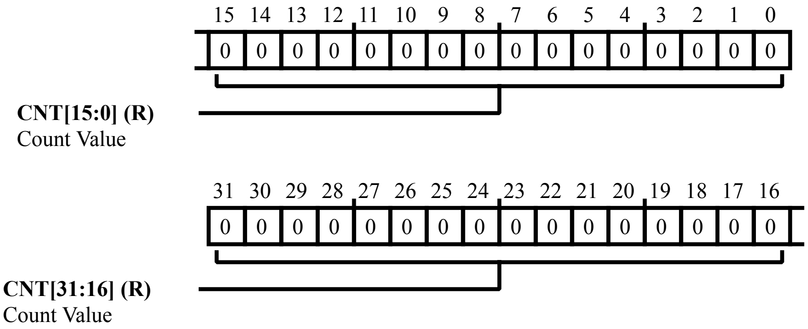

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-119: EMAC_RXCTLFRM_G Register Diagram

Table 31-151: EMAC_RXCTLFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx FIFO Overflow Register

The EMAC_RXFIFO_OVF register contains a count of the number of missed received frames due to FIFO overflow.

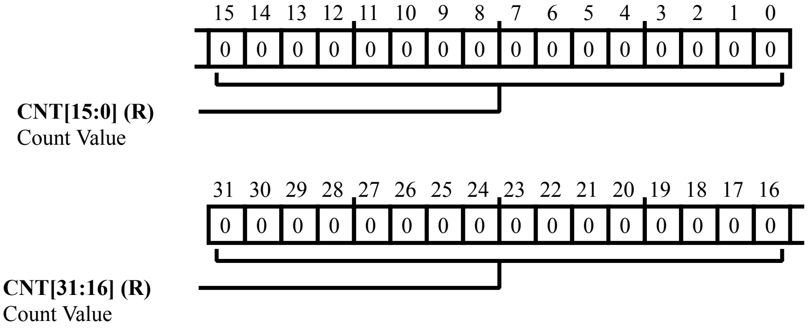

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-120: EMAC_RXFIFO_OVF Register Diagram

Table 31-152: EMAC_RXFIFO_OVF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Frame Count (Good/Bad) Register

The `EMAC_RXFRMCNT_GB` register contains a count of the number of good and bad frames received.

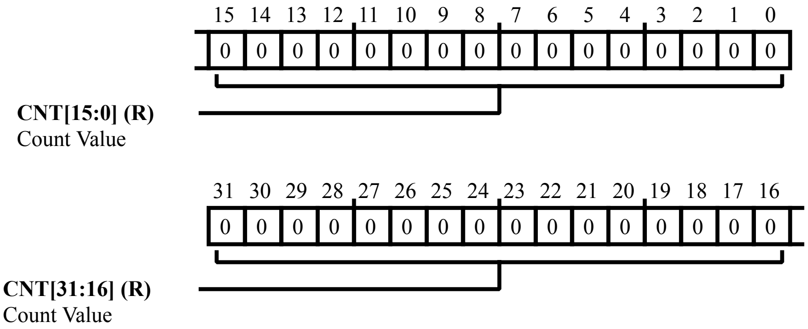

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-121: EMAC_RXFRMCNT_GB Register Diagram

Table 31-153: EMAC_RXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Frames Register

The `EMAC_RXICMP_ERR_FRM` register contains a count of the number of good IP datagrams whose ICMP payload has a checksum error.

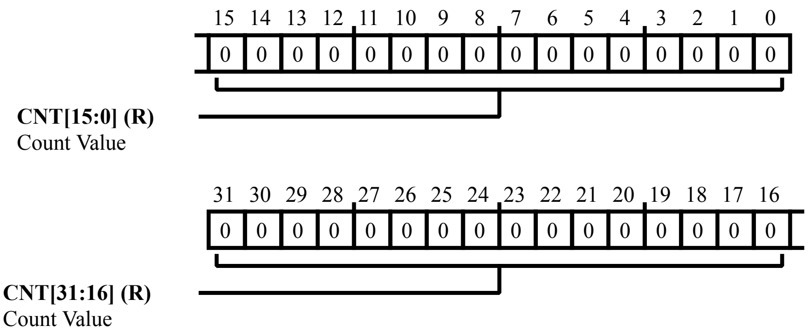

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-122: EMAC_RXICMP_ERR_FRM Register Diagram

Table 31-154: EMAC_RXICMP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Octets Register

The `EMAC_RXICMP_ERR_OCT` register contains a count of the number of bytes received in an ICMP segment with checksum errors.

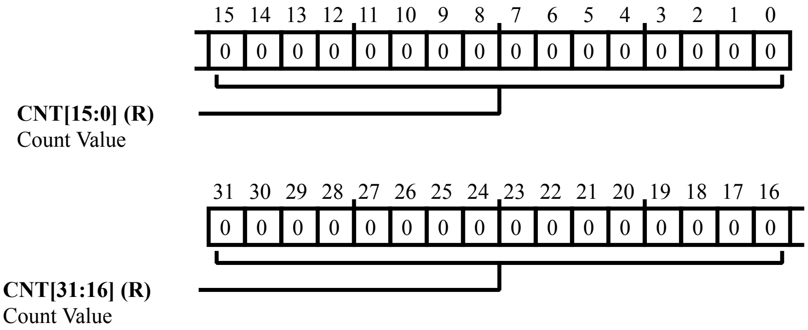

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-123: EMAC_RXICMP_ERR_OCT Register Diagram

Table 31-155: EMAC_RXICMP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Frames Register

The `EMAC_RXICMP_GD_FRM` register contains a count of the number of good IP datagrams with a good ICMP payload.

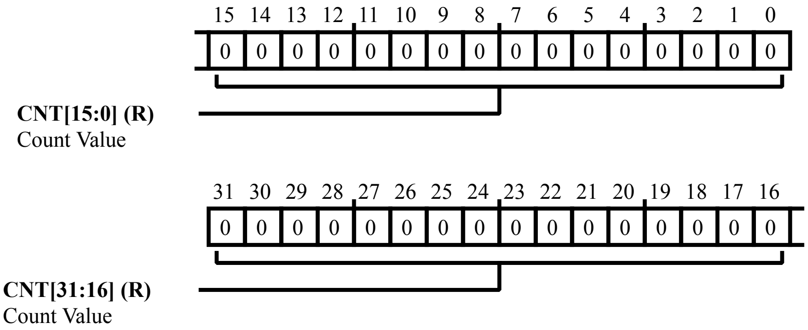

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-124: EMAC_RXICMP_GD_FRM Register Diagram

Table 31-156: EMAC_RXICMP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Octets Register

The `EMAC_RXICMP_GD_OCT` register contains a count of the Number of bytes received in a good ICMP segment.

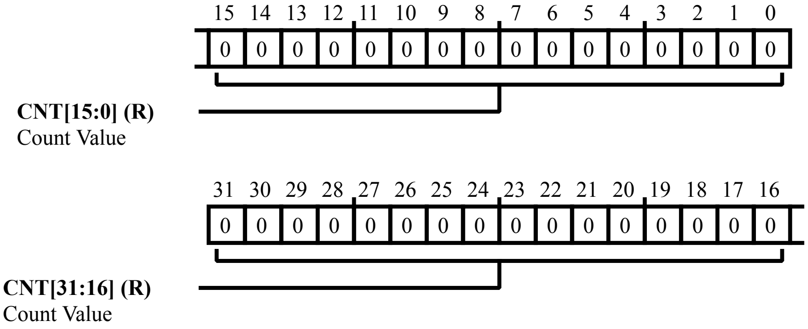

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-125: EMAC_RXICMP_GD_OCT Register Diagram

Table 31-157: EMAC_RXICMP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Frames Register

The `EMAC_RXIPV4_FRAG_FRM` register contains a count of the number of good IPv4 datagrams with fragmentation.

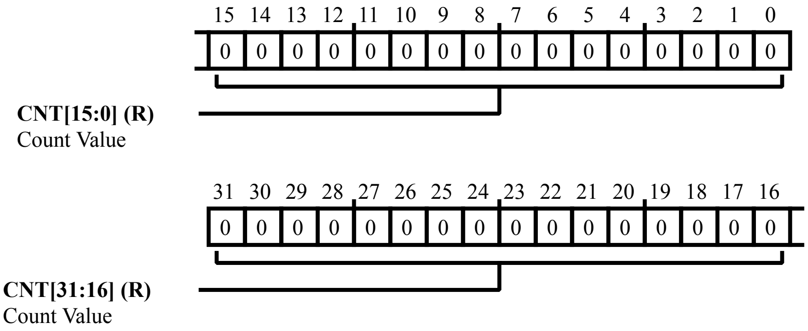

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-126: EMAC_RXIPV4_FRAG_FRM Register Diagram

Table 31-158: EMAC_RXIPV4_FRAG_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Octets Register

The `EMAC_RXIPV4_FRAG_OCT` register contains a count of the number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 headers Length field is used to update this counter.

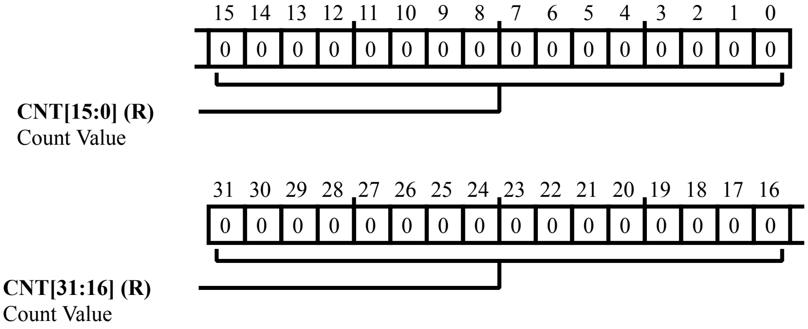

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-127: EMAC_RXIPV4_FRAG_OCT Register Diagram

Table 31-159: EMAC_RXIPV4_FRAG_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams (Good) Register

The `EMAC_RXIPV4_GD_FRM` register contains a count of the number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload.

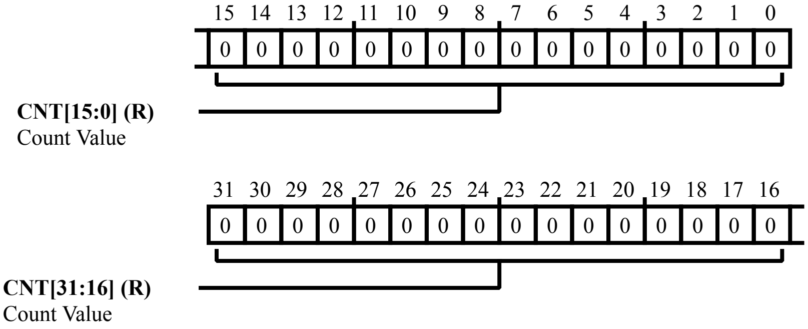

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-128: EMAC_RXIPV4_GD_FRM Register Diagram

Table 31-160: EMAC_RXIPV4_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Good Octets Register

The `EMAC_RXIPV4_GD_OCT` register contains a count of the number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data.

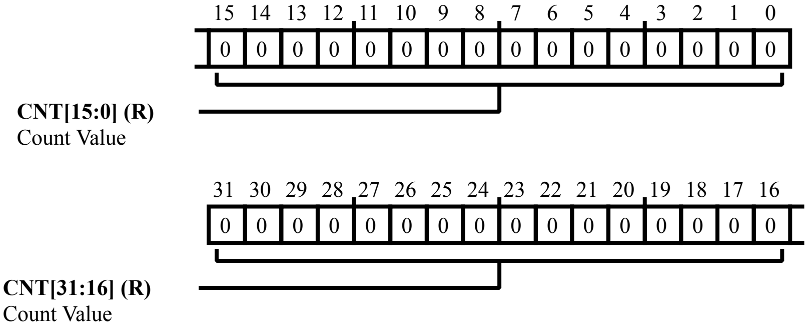

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-129: EMAC_RXIPV4_GD_OCT Register Diagram

Table 31-161: EMAC_RXIPV4_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The `EMAC_RXIPV4_HDR_ERR_FRM` register contains a count of the number of IPv4 datagrams received with header (checksum, length, or version mismatch) errors.

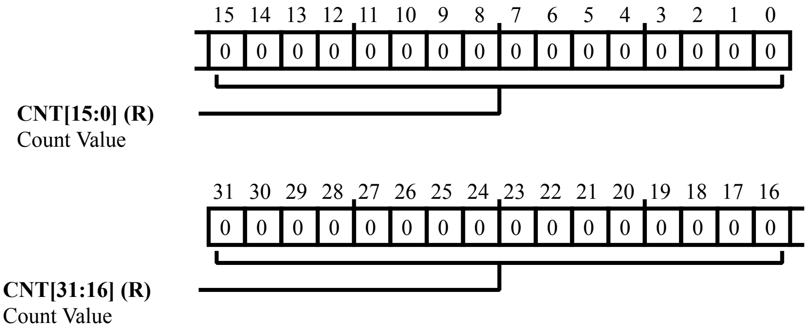

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-130: EMAC_RXIPV4_HDR_ERR_FRM Register Diagram

Table 31-162: EMAC_RXIPV4_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The `EMAC_RXIPV4_HDR_ERR_OCT` register contains a count of the number of bytes received in IPv4 datagrams with header errors (checksum, length, version mismatch). The value in the Length field of IPv4 header is used to update this counter.

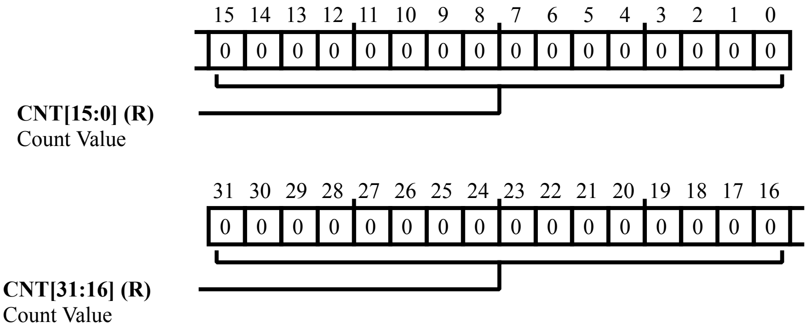

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-131: EMAC_RXIPV4_HDR_ERR_OCT Register Diagram

Table 31-163: EMAC_RXIPV4_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Frame Register

The `EMAC_RXIPV4_NOPAY_FRM` register contains a count of the number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine.

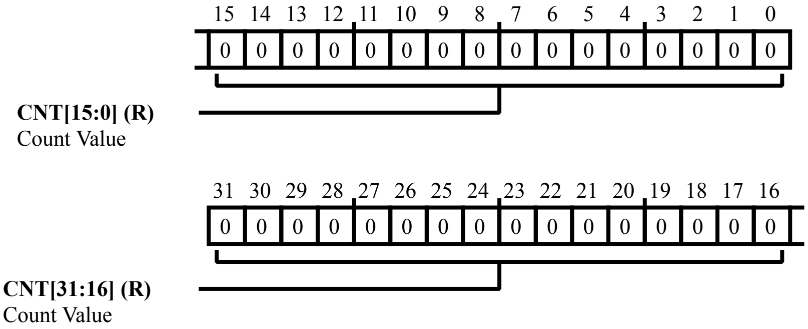

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-132: EMAC_RXIPV4_NOPAY_FRM Register Diagram

Table 31-164: EMAC_RXIPV4_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Octets Register

The `EMAC_RXIPV4_NOPAY_OCT` register contains a count of the number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 headers Length field is used to update this counter.

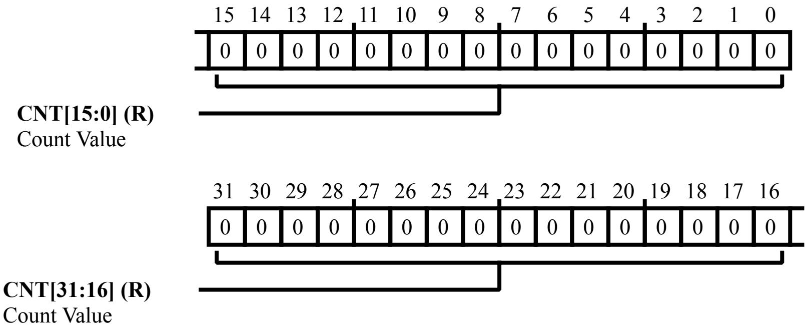

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-133: EMAC_RXIPV4_NOPAY_OCT Register Diagram

Table 31-165: EMAC_RXIPV4_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Frames Register

The `EMAC_RXIPV4_UDSBL_FRM` register contains a count of the number of good IPv4 datagrams received that had a UDP payload with checksum disabled.

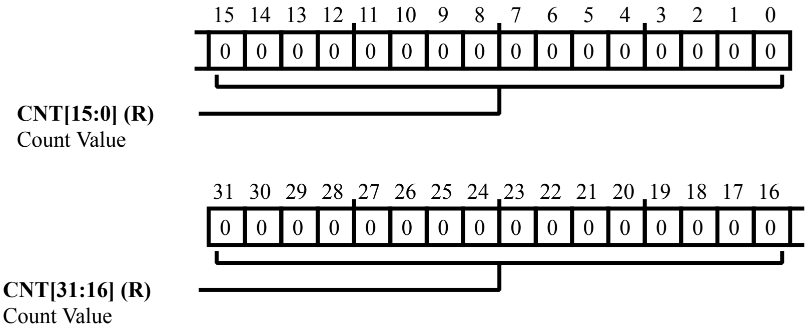

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-134: EMAC_RXIPV4_UDSBL_FRM Register Diagram

Table 31-166: EMAC_RXIPV4_UDSBL_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Octets Register

The `EMAC_RXIPV4_UDSBL_OCT` register contains a count of the number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count IP Header bytes.

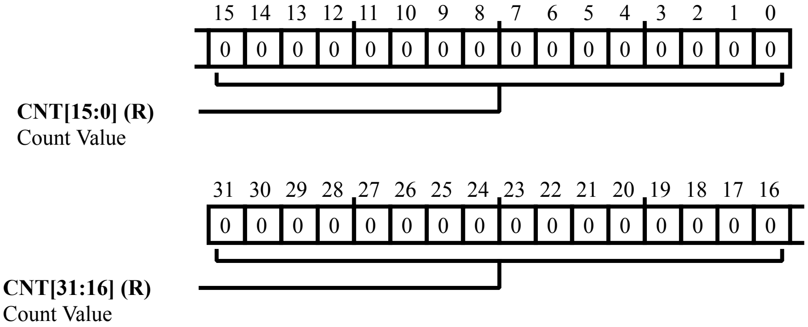

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-135: EMAC_RXIPV4_UDSBL_OCT Register Diagram

Table 31-167: EMAC_RXIPV4_UDSBL_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Good Frames Register

The EMAC_RXIPV6_GD_FRM register contains a count of the number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads.

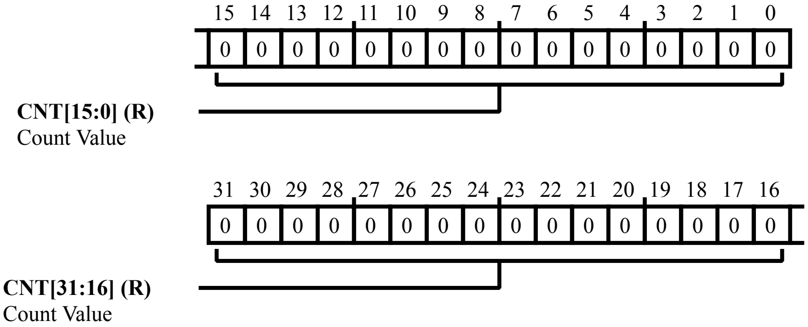

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-136: EMAC_RXIPV6_GD_FRM Register Diagram

Table 31-168: EMAC_RXIPV6_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Good Octets Register

The `EMAC_RXIPV6_GD_OCT` register contains a count of the number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data

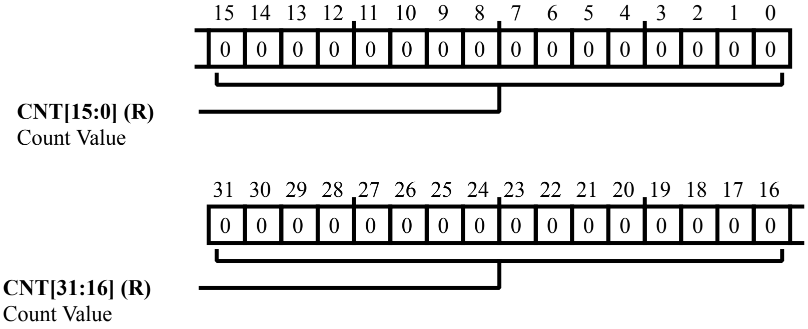

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-137: EMAC_RXIPV6_GD_OCT Register Diagram

Table 31-169: EMAC_RXIPV6_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Header Error Frames Register

The `EMAC_RXIPV6_HDR_ERR_FRM` register contains a count of the number of IPv6 datagrams received with header errors (length or version mismatch).

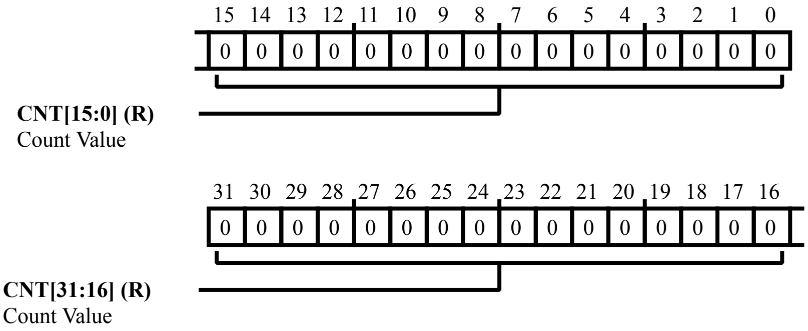

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-138: EMAC_RXIPV6_HDR_ERR_FRM Register Diagram

Table 31-170: EMAC_RXIPV6_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Header Errors Register

The `EMAC_RXIPV6_HDR_ERR_OCT` register contains a count of the number of bytes received in IPv6 datagrams with header errors (length, version mismatch). The value in the IPv6 headers Length field is used to update this counter.

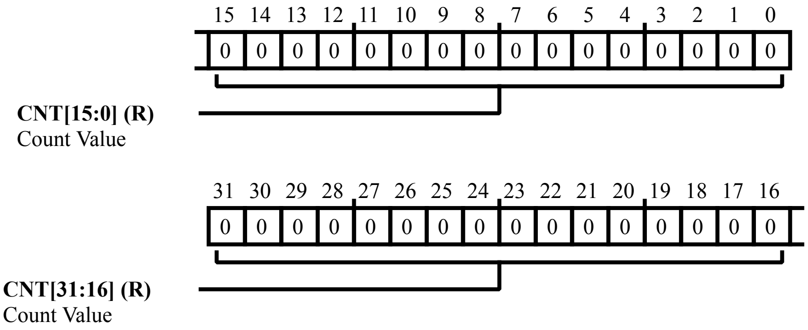

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-139: EMAC_RXIPV6_HDR_ERR_OCT Register Diagram

Table 31-171: EMAC_RXIPV6_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams No Payload Frames Register

The `EMAC_RXIPV6_NOPAY_FRM` register contains a count of the number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers.

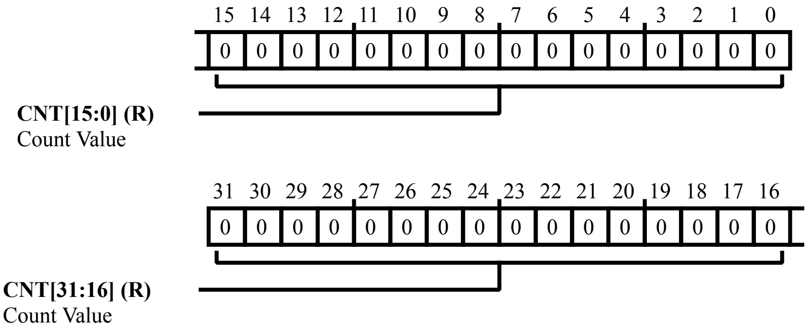

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-140: EMAC_RXIPV6_NOPAY_FRM Register Diagram

Table 31-172: EMAC_RXIPV6_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 No Payload Octets Register

The `EMAC_RXIPV6_NOPAY_OCT` register contains a count of the number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 headers Length field is used to update this counter.

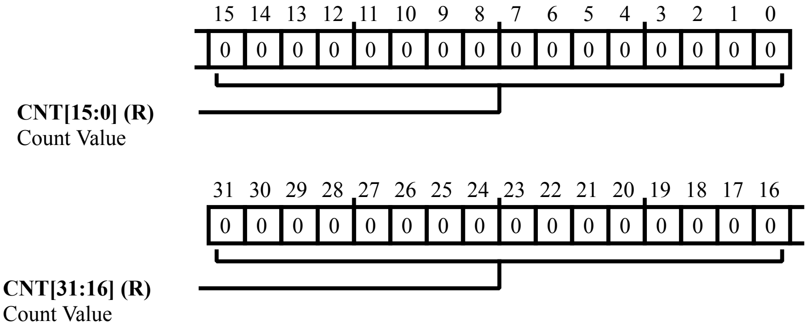

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-141: EMAC_RXIPV6_NOPAY_OCT Register Diagram

Table 31-173: EMAC_RXIPV6_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Jab Error Register

The EMAC_RXJAB_ERR register contains a count of the number of giant frames received with length greater than 1,518 bytes and with CRC error.

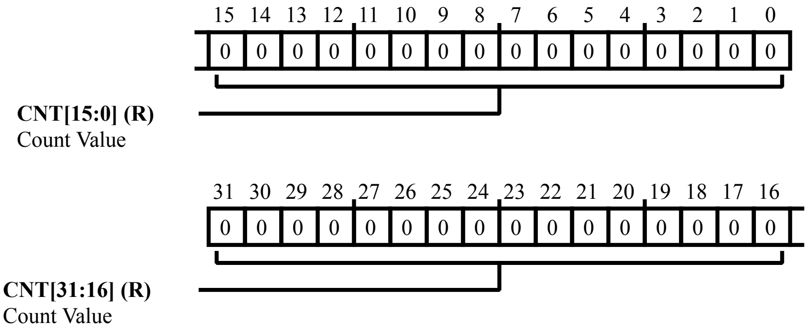

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-142: EMAC_RXJAB_ERR Register Diagram

Table 31-174: EMAC_RXJAB_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Length Error Register

The EMAC_RXLEN_ERR register contains a count of the number of frames received with length error (Length type field frame size), for all frames with valid length field.

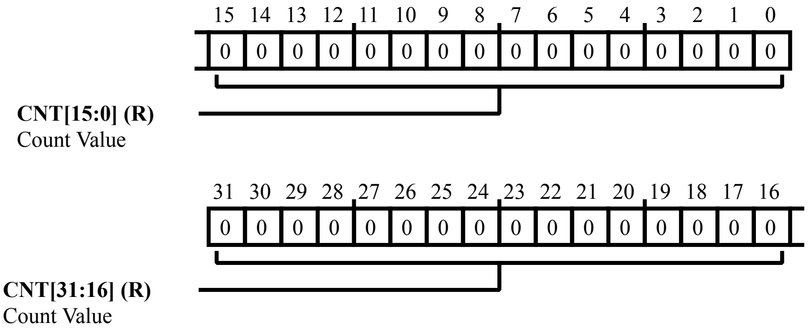

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-143: EMAC_RXLEN_ERR Register Diagram

Table 31-175: EMAC_RXLEN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Multicast Frames (Good) Register

The `EMAC_RXMCASTFRM_G` register contains a count of the number of good multicast frames received.

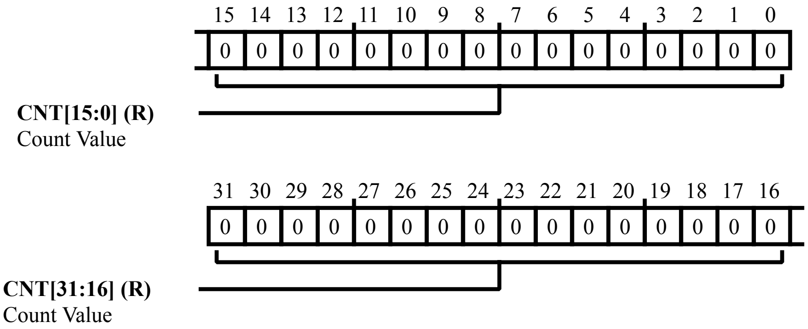

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-144: EMAC_RXMCASTFRM_G Register Diagram

Table 31-176: EMAC_RXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good) Register

The `EMAC_RXOCTCNT_G` register contains a count of the number of bytes received, exclusive of preamble, only in good frames.

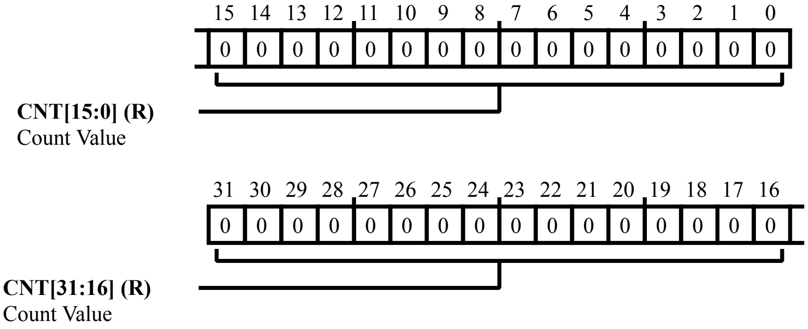

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-145: EMAC_RXOCTCNT_G Register Diagram

Table 31-177: EMAC_RXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good/Bad) Register

The EMAC_RXOCTCNT_GB register contains a count of the number of bytes received, exclusive of preamble, in good and bad frames.

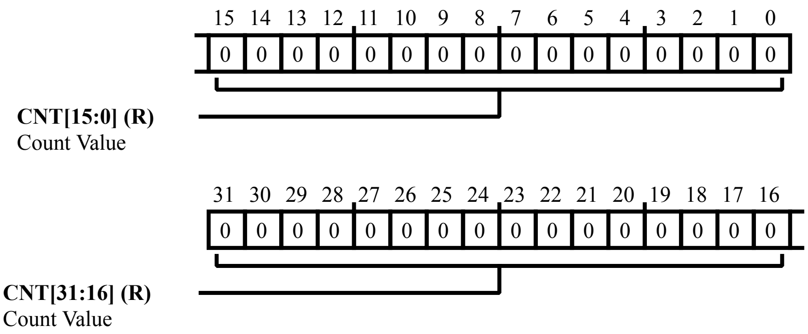

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-146: EMAC_RXOCTCNT_GB Register Diagram

Table 31-178: EMAC_RXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Out Of Range Type Register

The EMAC_RXOORTYPE register contains a count of the number of frames received with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).

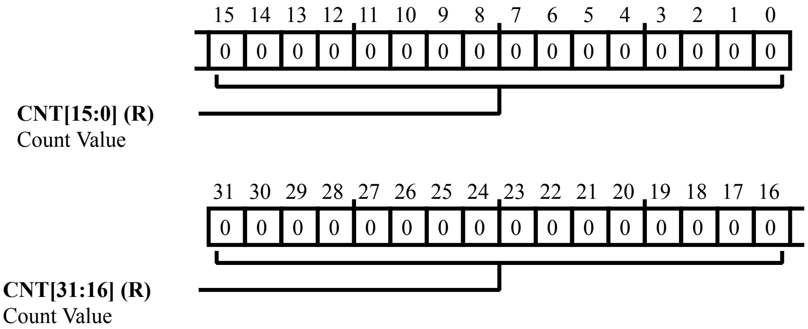

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-147: EMAC_RXOORTYPE Register Diagram

Table 31-179: EMAC_RXOORTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Oversize (Good) Register

The `EMAC_RXOSIZE_G` register contains a count of the number of frames received with length greater than the maxsize, without errors.

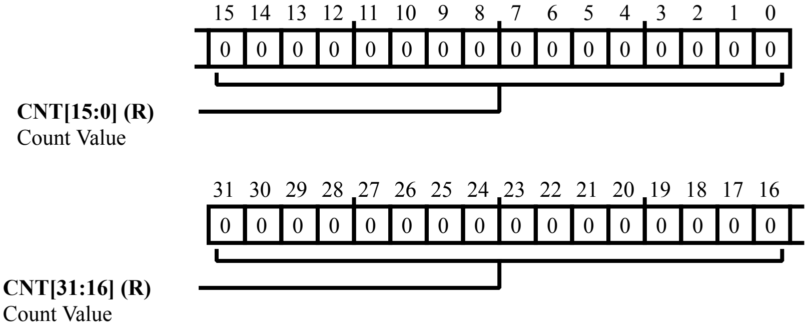

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-148: EMAC_RXOSIZE_G Register Diagram

Table 31-180: EMAC_RXOSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Pause Frames Register

The `EMAC_RXPAUSEFRM` register contains a count of the number of good and valid PAUSE frames received.

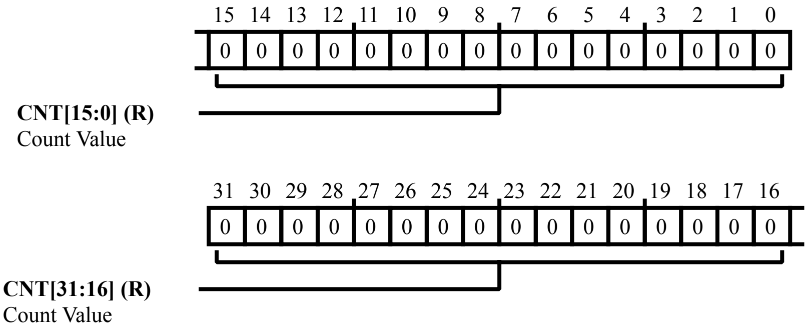

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-149: EMAC_RXPAUSEFRM Register Diagram

Table 31-181: EMAC_RXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Error Frames Received Register

The EMAC_RXRCV_ERR register contains a count of the number of frames received with Receive error or Frame Extension error on GMII or MII interface

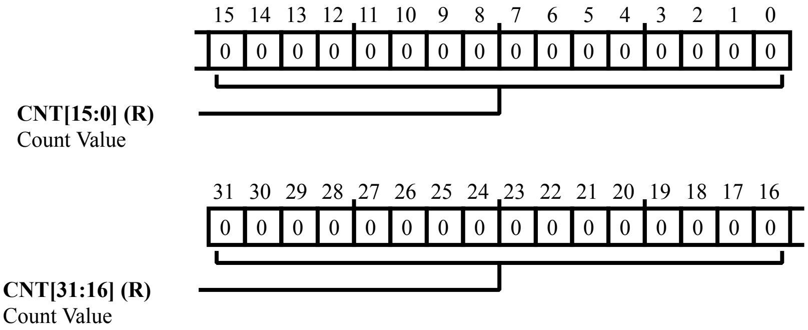

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-150: EMAC_RXRCV_ERR Register Diagram

Table 31-182: EMAC_RXRCV_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Runt Error Register

The `EMAC_RXRUNT_ERR` register contains a count of the number of frames received with runt error.

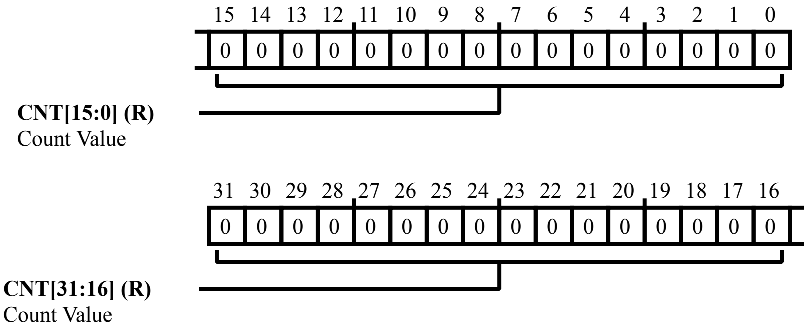

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-151: EMAC_RXRUNT_ERR Register Diagram

Table 31-183: EMAC_RXRUNT_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Frames Register

The `EMAC_RXTCP_ERR_FRM` register contains a count of the number of good IP datagrams whose TCP payload has a checksum error.

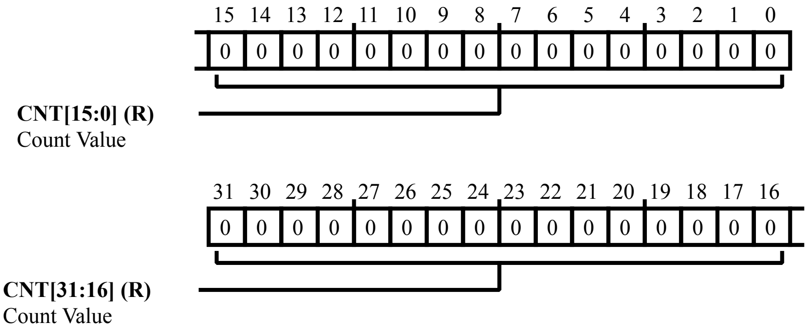

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-152: EMAC_RXTCP_ERR_FRM Register Diagram

Table 31-184: EMAC_RXTCP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Octets Register

The `EMAC_RXTCP_ERR_OCT` register contains a count of the number of bytes received in a TCP segment with checksum errors.

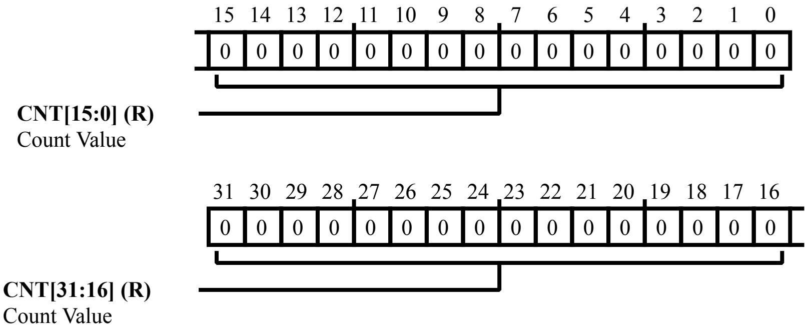

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-153: EMAC_RXTCP_ERR_OCT Register Diagram

Table 31-185: EMAC_RXTCP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Frames Register

The `EMAC_RXTCP_GD_FRM` register contains a count of the number of good IP datagrams with a good TCP payload.

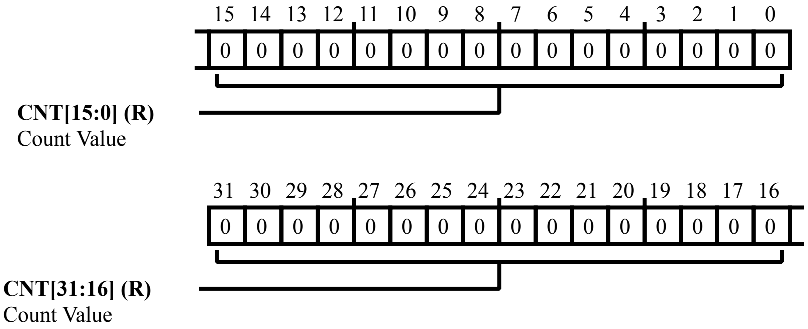

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-154: EMAC_RXTCP_GD_FRM Register Diagram

Table 31-186: EMAC_RXTCP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Octets Register

The EMAC_RXTCP_GD_OCT register contains a count of the number of bytes received in a good TCP segment.

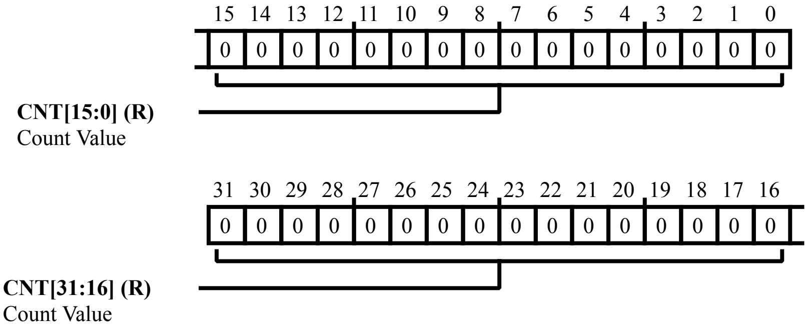

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-155: EMAC_RXTCP_GD_OCT Register Diagram

Table 31-187: EMAC_RXTCP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Unicast Frames (Good) Register

The EMAC_RXUCASTFRM_G register contains a count of the number of good unicast frames received.

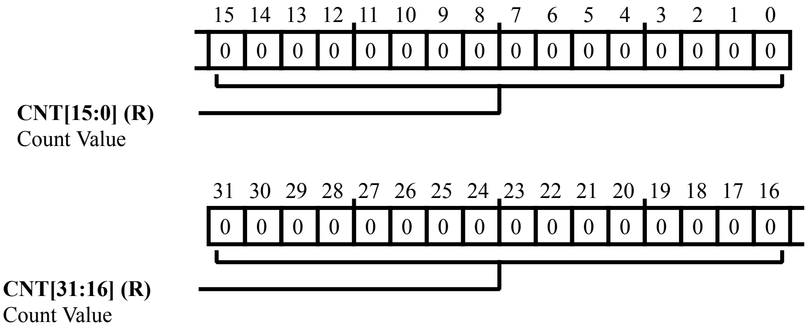

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-156: EMAC_RXUCASTFRM_G Register Diagram

Table 31-188: EMAC_RXUCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Frames Register

The `EMAC_RXUDP_ERR_FRM` register contains a count of the number of good IP datagrams whose UDP payload has a checksum error.

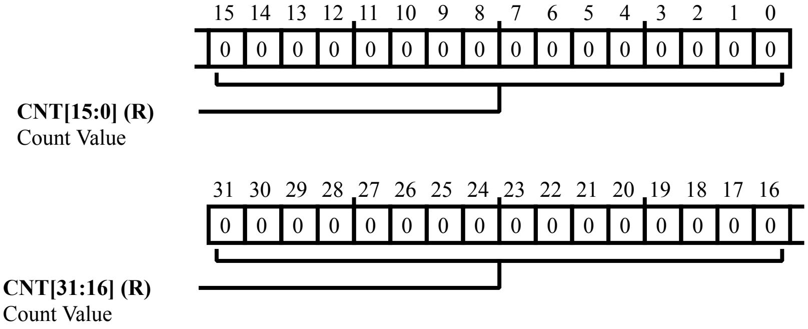

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-157: EMAC_RXUDP_ERR_FRM Register Diagram

Table 31-189: EMAC_RXUDP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Octets Register

The `EMAC_RXUDP_ERR_OCT` register contains a count of the number of bytes received in a UDP segment that had checksum errors.

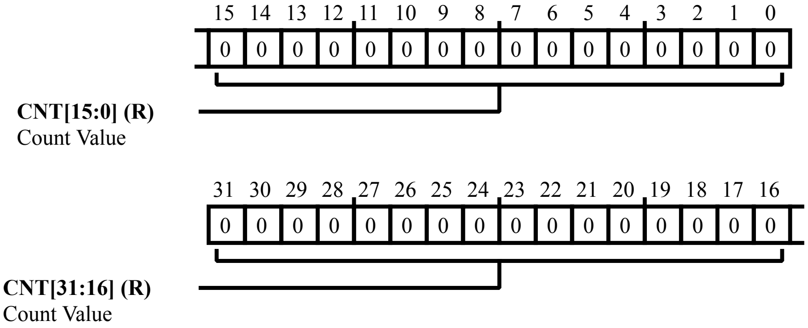

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-158: EMAC_RXUDP_ERR_OCT Register Diagram

Table 31-190: EMAC_RXUDP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Frames Register

The `EMAC_RXUDP_GD_FRM` register contains a count of the number of good IP datagrams with a good UDP payload. This counter is not updated when the rxipv4_udsbl_frms counter is incremented.

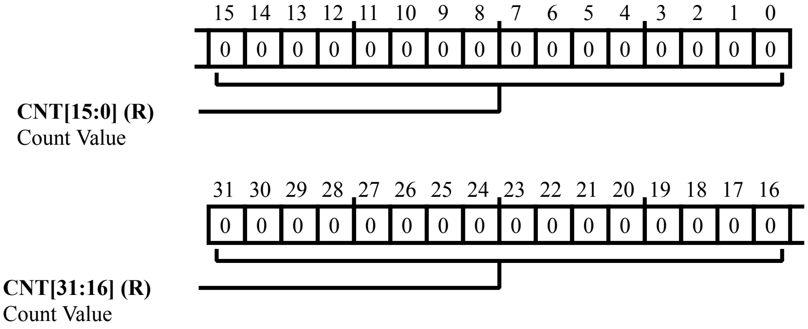

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-159: EMAC_RXUDP_GD_FRM Register Diagram

Table 31-191: EMAC_RXUDP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Octets Register

The `EMAC_RXUDP_GD_OCT` register contains a count of the number of bytes received in a good UDP segment. This counter (and the counters below) does not count IP header bytes.

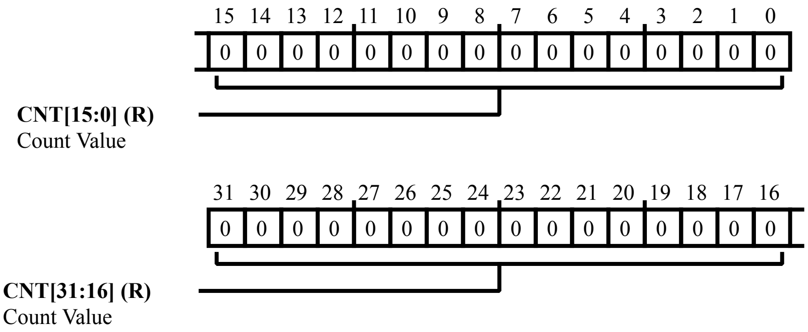

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-160: EMAC_RXUDP_GD_OCT Register Diagram

Table 31-192: EMAC_RXUDP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Undersize (Good) Register

The `EMAC_RXUSIZE_G` register contains a count of the number of frames received with length less than 64 bytes, without any errors.

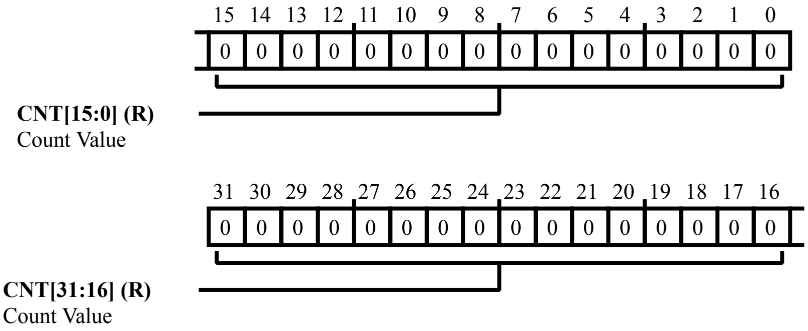

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-161: EMAC_RXUSIZE_G Register Diagram

Table 31-193: EMAC_RXUSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx VLAN Frames (Good/Bad) Register

The EMAC_RXVLANFRM_GB register contains a count of the number of good and bad VLAN frames received.

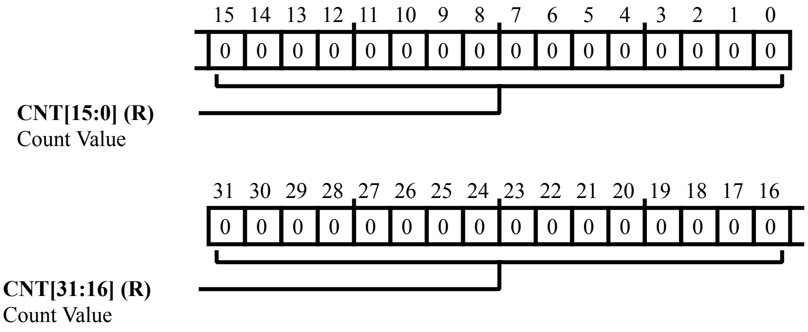

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-162: EMAC_RXVLANFRM_GB Register Diagram

Table 31-194: EMAC_RXVLANFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Watch Dog Error Register

The `EMAC_RXWDOG_ERR` register contains a count of the number of frames received with error due to watchdog timeout error (frames with a data load larger than 2,048 bytes).

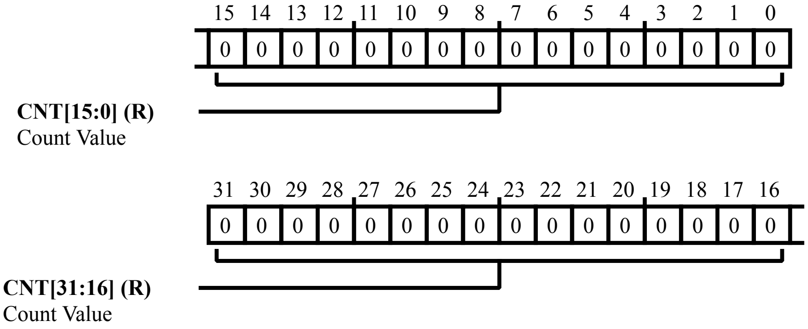

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-163: EMAC_RXWDOG_ERR Register Diagram

Table 31-195: EMAC_RXWDOG_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

SMI Address Register

The EMAC_SMI_ADDR register contains the station management interface address and feature settings.

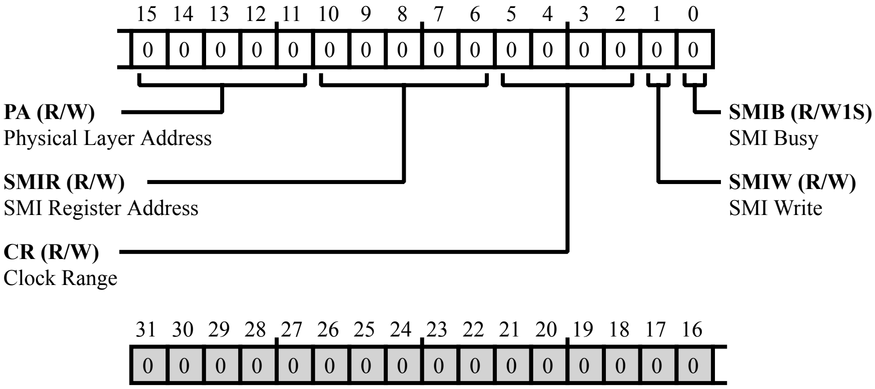

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PA (R/W)
Physical Layer Address
SMIR (R/W)
SMI Register Address
CR (R/W)
Clock Range
SMIB (R/W1S)
SMI Busy
SMIW (R/W)
SMI Write
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-164: EMAC_SMI_ADDR Register Diagram

Table 31-196: EMAC_SMI_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:11 (R/W)	PA	Physical Layer Address. The `EMAC_SMI_ADDR.PA` bits select the PHY. This field tells which of the 32 possible PHY devices are being accessed.	
10:6 (R/W)	SMIR	SMI Register Address. The `EMAC_SMI_ADDR.SMIR` bits select the desired Station Management Interface register in the selected PHY device.	
5:2 (R/W)	CR	Clock Range. The `EMAC_SMI_ADDR.CR` bits select the Clock Range, determining the frequency of the MDC clock as per the SCLK0_0 frequency. The suggested range of SCLK0_0 frequency applicable for each value below (when Bit[5] =0) ensures that the MDC clock is approximately between the frequency range 1.0 MHz - 2.5 MHz. When the MSB of this field is set, you can achieve MDC clock of frequency higher than the IEEE 802.3 specified frequency limit of 2.5 MHz and program a clock divider of lower value. For example, when SCLK0_0=100 MHz and you program these bits to b#1010, the resulting MDC clock is 12.5 MHz, which is outside the limit of IEEE 802.3 specified range. Use the values shown only if the interface chips support faster MDC clocks.	
		0	MDC Clock= SCLK0_0/42 (for SCLK0_0=60-100MHz)
		1	MDC Clock= SCLK0_0/62 (for SCLK0_0=100-125 MHz)
		2	MDC Clock= SCLK0_0/16 (for SCLK0_0=20-35 MHz)

Table 31-196: EMAC_SMI_ADDR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	MDC Clock= SCLK0_0/26 (for SCLK0_0=35-60 MHz)
		8	MDC Clock= SCLK0_0/4
		9	MDC Clock= SCLK0_0/6
		10	MDC Clock= SCLK0_0/8
		11	MDC Clock= SCLK0_0/10
		12	MDC Clock= SCLK0_0/12
		13	MDC Clock= SCLK0_0/14
		14	MDC Clock= SCLK0_0/16
		15	MDC Clock= SCLK0_0/18
1 (R/W)	SMIW	SMI Write. The `EMAC_SMI_ADDR.SMIW` bit, when set, tells the PHY this is a Write operation using the Station Management Interface Data register. If this bit is not set, this is a Read operation.	
0 (R/W1S)	SMIB	SMI Busy. The `EMAC_SMI_ADDR.SMIB` bit should read low (=0) before writing to the `EMAC_SMI_ADDR` and `EMAC_SMI_DATA` registers. This bit must also =0 during a Write to `EMAC_SMI_ADDR`. During a PHY register access, this bit is set (=1) by the Application to indicate that a Read or Write access is in progress. The `EMAC_SMI_DATA` register should be kept valid until this bit is cleared by the MAC during a PHY Write operation. `EMAC_SMI_DATA` is invalid until this bit is cleared by the MAC during a PHY Read operation. The `EMAC_SMI_ADDR` should not be written to until this bit is cleared.	

SMI Data Register

The EMAC_SMI_DATA register contains the station management interface data.

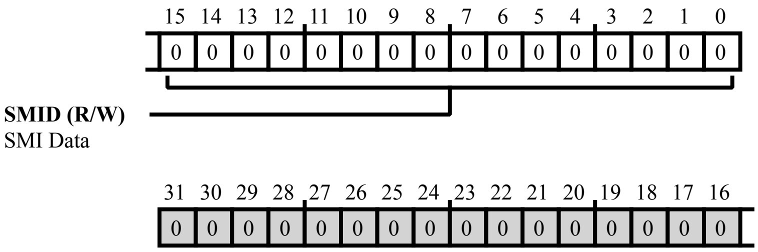

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SMID (R/W)
SMI Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-165: EMAC_SMI_DATA Register Diagram

Table 31-197: EMAC_SMI_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SMID	SMI Data. The `EMAC_SMI_DATA.SMID` bits contain the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.

Time Stamp Addend Register

The `EMAC_TM_ADDEND` register lets software adjust the clock frequency linearly to match the master clock frequency.

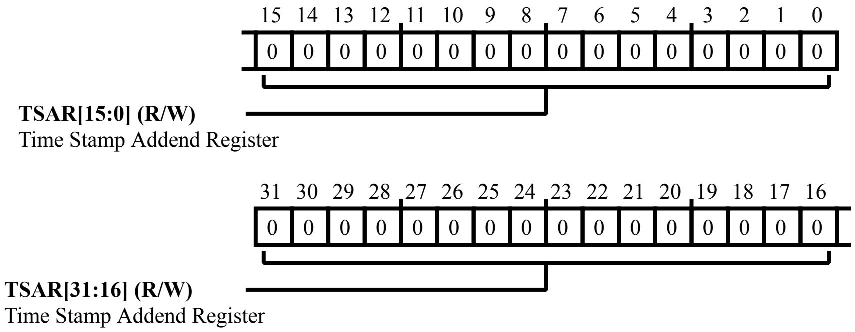

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSAR[15:0] (R/W)
Time Stamp Addend Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSAR[31:16] (R/W)
Time Stamp Addend Register

Figure 31-166: EMAC_TM_ADDEND Register Diagram

Table 31-198: EMAC_TM_ADDEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSAR	Time Stamp Addend Register. The `EMAC_TM_ADDEND.TSAR` bits indicate the 32-bit time value to be added to the Accumulator register to achieve time synchronization.

Time Stamp Auxiliary TS Nano Seconds Register

The `EMAC_TM_AUXSTMP_NSEC` register contains the low 32 bits (nanoseconds field) of the auxiliary time stamp.

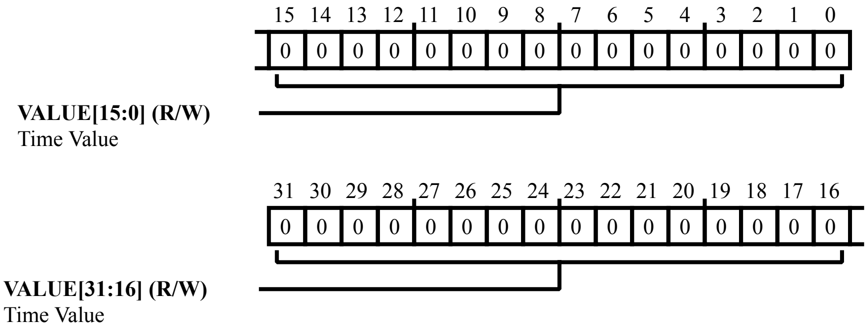

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Time Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Time Value

Figure 31-167: EMAC_TM_AUXSTMP_NSEC Register Diagram

Table 31-199: EMAC_TM_AUXSTMP_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp Auxiliary TM Seconds Register

The `EMAC_TM_AUXSTMP_SEC` register contains the low 32 bits of the seconds field of the auxiliary time stamp.

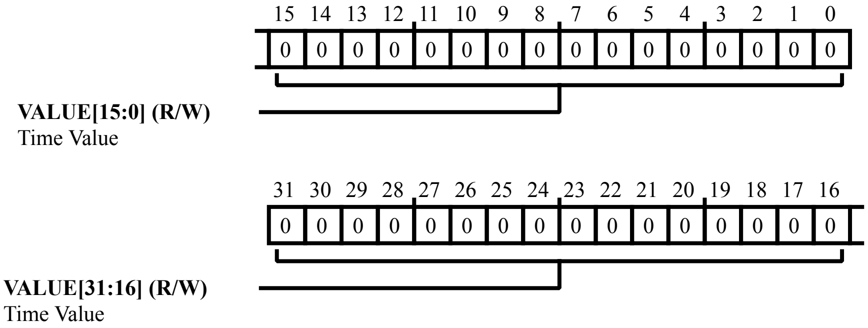

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Time Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Time Value

Figure 31-168: EMAC_TM_AUXSTMP_SEC Register Diagram

Table 31-200: EMAC_TM_AUXSTMP_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp Control Register

The `EMAC_TM_CTL` register controls time stamp generation and update. The `EMAC_TM_CTL.SNAPTYPSEL`, `EMAC_TM_CTL.TSMSTRENA`, and `EMAC_TM_CTL.TSEVNTENA` bits work together to decide the set of PTP packet types for which snapshot needs to be taken. (Encoding shown in table.)

SNAPTYPSEL()	TSMSTRENA	TSEVNTENA	Messages for which snapshot is taken
00	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
00	0	1	SYNC
00	1	1	Delay_Req
01	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
01	0	1	SYNC, Pdelay_Req, Pdelay_Resp
01	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
10	X	X	SYNC, Delay_Req
11	X	X	Pdelay_Req, Pdelay_Resp

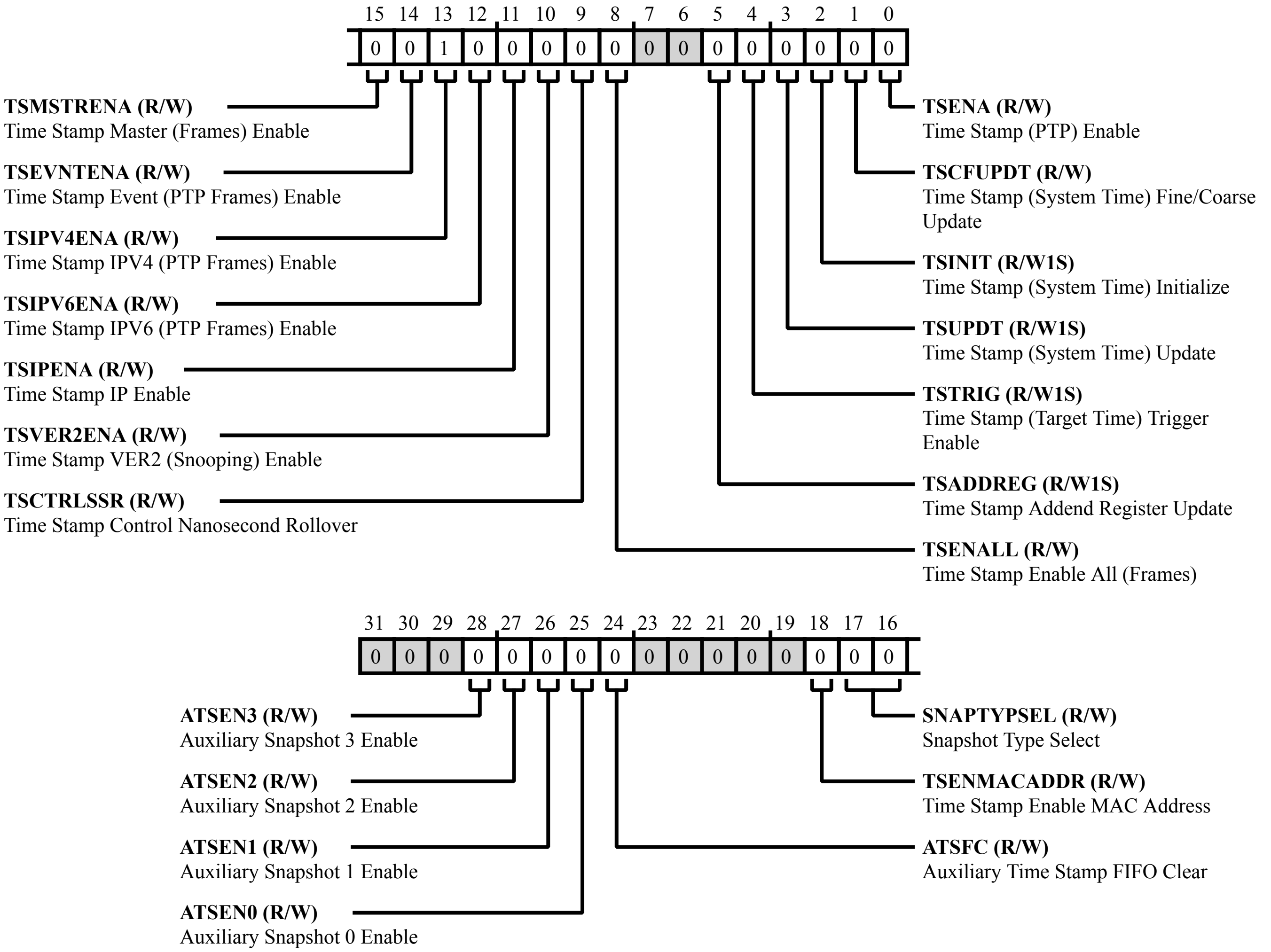

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
TSMSTRENA (R/W)
Time Stamp Master (Frames) Enable
TSEVNTENA (R/W)
Time Stamp Event (PTP Frames) Enable
TSIPV4ENA (R/W)
Time Stamp IPV4 (PTP Frames) Enable
TSIPV6ENA (R/W)
Time Stamp IPV6 (PTP Frames) Enable
TSIPENA (R/W)
Time Stamp IP Enable
TSVER2ENA (R/W)
Time Stamp VER2 (Snooping) Enable
TSCTRLSSR (R/W)
Time Stamp Control Nanosecond Rollover
TSENA (R/W)
Time Stamp (PTP) Enable
TSCFUPDT (R/W)
Time Stamp (System Time) Fine/Coarse Update
TSINIT (R/W1S)
Time Stamp (System Time) Initialize
TSUPDT (R/W1S)
Time Stamp (System Time) Update
TSTRIG (R/W1S)
Time Stamp (Target Time) Trigger Enable
TSADDREG (R/W1S)
Time Stamp Addend Register Update
TSENALL (R/W)
Time Stamp Enable All (Frames)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ATSEN3 (R/W)
Auxiliary Snapshot 3 Enable
ATSEN2 (R/W)
Auxiliary Snapshot 2 Enable
ATSEN1 (R/W)
Auxiliary Snapshot 1 Enable
ATSEN0 (R/W)
Auxiliary Snapshot 0 Enable
SNAPTYPSEL (R/W)
Snapshot Type Select
TSENMACADDR (R/W)
Time Stamp Enable MAC Address
ATSFC (R/W)
Auxiliary Time Stamp FIFO Clear

Figure 31-169: EMAC_TM_CTL Register Diagram

Table 31-201: EMAC_TM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	ATSEN3	Auxiliary Snapshot 3 Enable. The `EMAC_TM_CTL.ATSEN3` bit, controls capturing the Auxiliary Snapshot Trigger 3
27 (R/W)	ATSEN2	Auxiliary Snapshot 2 Enable. The `EMAC_TM_CTL.ATSEN2` bit, controls capturing the Auxiliary Snapshot Trigger 2
26 (R/W)	ATSEN1	Auxiliary Snapshot 1 Enable. The `EMAC_TM_CTL.ATSEN1` bit, controls capturing the Auxiliary Snapshot Trigger 1

Table 31-201: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	ATSEN0	Auxiliary Snapshot 0 Enable. The `EMAC_TM_CTL.ATSEN0` bit, controls capturing the Auxiliary Snapshot Trigger 0	
24 (R/W)	ATSFC	Auxiliary Time Stamp FIFO Clear. The `EMAC_TM_CTL.ATSFC` bit, when set, resets the pointers of the Auxiliary Snapshot FIFO. This bit is cleared when the pointers are reset and the FIFO is empty. When this bit is cleared, auxiliary snapshots gets stored in the FIFO.	
18 (R/W)	TSENMACADDR	Time Stamp Enable MAC Address. The `EMAC_TM_CTL.TSENMACADDR` bit, when set, uses the DA MAC address (that matches the `EMAC_ADDR0_LO` and `EMAC_ADDR0_HI` registers) to filter the PTP frames when PTP is sent directly over Ethernet.	
		0	Disable PTP MAC address filter
		1	Enable PTP MAC address filter
17:16 (R/W)	SNAPTYPSEL	Snapshot Type Select. The `EMAC_TM_CTL.SNAPTYPSEL` bits along with bit 15 and 14 decide the set of PTP packet types for which snapshot needs to be taken. (See the table in the `EMAC_TM_CTL` register description.)	
15 (R/W)	TSMSTRENA	Time Stamp Master (Frames) Enable. The `EMAC_TM_CTL.TSMSTRENA` bit, when set, takes the snapshot for messages relevant to master node only else snapshot is taken for PTP messages relevant to slave node.	
		0	Enable Snapshot for Slave Messages
		1	Enable Snapshot for Master Messages
14 (R/W)	TSEVNTENA	Time Stamp Event (PTP Frames) Enable. The `EMAC_TM_CTL.TSEVNTENA` bit, when set, takes the time stamp snapshot for PTP event messages only (SYNC, Delay_Req, Pdelay_Req, or Pdelay_Resp). When reset, the snapshot is taken for all PTP messages except Announce, Management, and Signaling.	
		0	Enable Time Stamp for All Messages
		1	Enable Time Stamp for Event Messages Only

Table 31-201: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	TSIPV4ENA	Time Stamp IPV4 (PTP Frames) Enable. The `EMAC_TM_CTL.TSIPV4ENA` bit, when set, directs the EMAC receiver to process the PTP packets encapsulated in UDP over IPv4 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv4 packets. This bit is set by default.	
		0	Disable Time Stamp for PTP Over IPv4 Frames
		1	Enable Time Stamp for PTP Over IPv4 Frames
12 (R/W)	TSIPV6ENA	Time Stamp IPV6 (PTP Frames) Enable. The `EMAC_TM_CTL.TSIPV6ENA` bit, when set, directs the EMAC receiver to process PTP packets encapsulated in UDP over IPv6 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv6 packets.	
		0	Disable Time Stamp for PTP Over IPv6 frames
		1	Enable Time Stamp for PTP Over IPv6 Frames
11 (R/W)	TSIPENA	Time Stamp IP Enable. The `EMAC_TM_CTL.TSIPENA` bit, when set, directs the EMAC receiver to process the PTP packets encapsulated directly in the Ethernet frames. When this bit is clear, the MAC ignores PTP over Ethernet packets.	
		0	Disable PTP Over Ethernet Frames
		1	Enable PTP Over Ethernet Frames
10 (R/W)	TSVER2ENA	Time Stamp VER2 (Snooping) Enable. The `EMAC_TM_CTL.TSVER2ENA` bit, when set, processes the PTP packets using the 1588 version 2 format (enables PTP packet snooping for VER2) else processed using the version 1 format.	
		0	Disable packet snooping for V2 frames
		1	Enable packet snooping for V2 frames
9 (R/W)	TSCTRLSSR	Time Stamp Control Nanosecond Rollover. The `EMAC_TM_CTL.TSCTRLSSR` bit, when set, rolls over the `EMAC_TM_NSEC` register after 0x3B9A_C9FF value (10^9-1) and increments the `EMAC_TM_SEC` register. When reset, the roll over value of `EMAC_TM_NSEC` register is 0x7FFF_FFFF. The nanosecond increment has to be programmed correctly depending on the PTP reference clock frequency and this bit value.	
		0	Roll Over Nanosecond After 0x7FFFFFFF
		1	Roll Over Nanosecond After 0x3B9AC9FF

Table 31-201: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	TSENALL	Time Stamp Enable All (Frames). The `EMAC_TM_CTL.TSENALL` bit, when set, enables the time stamp snapshot for all frames received by the core.	
		0	Disable timestamp for all frames
		1	Enable timestamp for all frames
5 (R/W1S)	TSADDREG	Time Stamp Addend Register Update. The `EMAC_TM_CTL.TSADDREG` bit, when set, updates the contents of the `EMAC_TM_ADDEND` register for fine correction. This bit is cleared when the update is completed. This bit should be zero before setting it.	
4 (R/W1S)	TSTRIG	Time Stamp (Target Time) Trigger Enable. The `EMAC_TM_CTL.TSTRIG` bit, when set, generates the time stamp interrupt when the System Time becomes greater than the value written in Time Stamp Target Time Seconds register. This bit is reset after the generation of the Time Stamp Trigger Interrupt.	
		1	Interrupt (TS) if system time is greater than target time register
3 (R/W1S)	TSUPDT	Time Stamp (System Time) Update. The `EMAC_TM_CTL.TSUPDT` bit, when set, updates (adds/subtracts) the system time with the value specified in the `EMAC_TM_SECUPDT` and `EMAC_TM_NSECUPDT` registers. This bit should read =0 before updating it. This bit is reset when the update is completed in hardware. The `EMAC_TM_NSEC` register is not updated.	
		1	System time updated with Time stamp register values
2 (R/W1S)	TSINIT	Time Stamp (System Time) Initialize. The `EMAC_TM_CTL.TSINIT` bit, when set, initializes (over-writes) the system time with the value specified in the `EMAC_TM_SECUPDT` and `EMAC_TM_NSECUPDT` registers. This bit should read =0 before updating it. This bit is reset when the initialization is complete. Only the `EMAC_TM_NSEC` register can be initialized.	
		1	System time initialized with Time stamp register values
1 (R/W)	TSCFUPDT	Time Stamp (System Time) Fine/Coarse Update. The `EMAC_TM_CTL.TSCFUPDT` bit, when set, indicates that the system time update is done using the fine correction method. When reset, it indicates the system time correction to be done using Coarse method.	
		0	Use Coarse Correction Method for System Time Update
		1	Use Fine Correction Method for System Time Update

Table 31-201: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	TSENA	Time Stamp (PTP) Enable. The `EMAC_TM_CTL.TSENA` bit, when set, enables PTP module for time stamping transmitted and received frames. It also enables System Time which is used for time stamping the frames. Programs should initialize the System Time after setting this bit.	
		0	Disable PTP Module
		1	Enable PTP Module

Time Stamp High Second Register

The EMAC_TM_HISEC register contains the upper 32 bits of the seconds field of the system time.

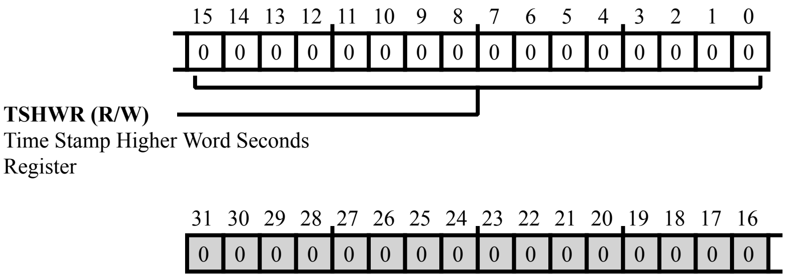

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSHWR (R/W)
Time Stamp Higher Word Seconds Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-170: EMAC_TM_HISEC Register Diagram

Table 31-202: EMAC_TM_HISEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSHWR	Time Stamp Higher Word Seconds Register. The EMAC_TM_HISEC.TSHWR bit field contains the most significant 16-bits of the time stamp seconds value. The register is directly written to initialize the value. This register is incremented when there is an overflow from the 32-bits of the EMAC_TM_SEC register.

Time Stamp Nanoseconds Register

The `EMAC_TM_NSEC` register contains the nanoseconds field of the system time.

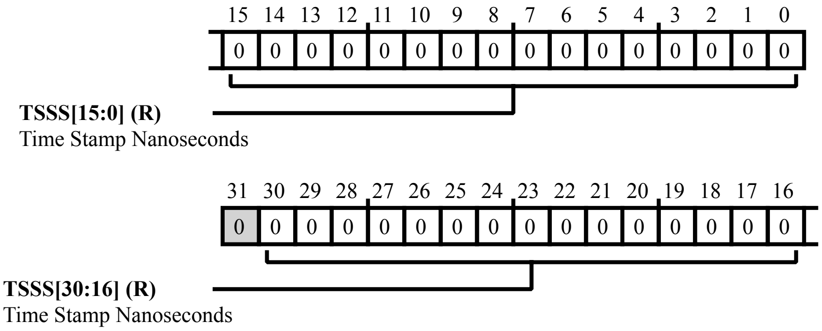

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSSS[15:0] (R)
Time Stamp Nanoseconds
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSSS[30:16] (R)
Time Stamp Nanoseconds

Figure 31-171: EMAC_TM_NSEC Register Diagram

Table 31-203: EMAC_TM_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:0 (R/NW)	TSSS	Time Stamp Nanoseconds. The value in the `EMAC_TM_NSEC.TSSS` bit field has the nanosecond representation of time, with an accuracy of 0.46 nanosecond. (When `EMAC_TM_CTL.TSCTRLSSR` is set, each bit represents 1 ns and the maximum value will be 0x3B9A_C9FF, after which it rolls-over to zero).

Time Stamp Nanoseconds Update Register

The `EMAC_TM_NSECUPDT` register contains the low 32 bits to be added to, subtracted from, or written to the nanoseconds field of the system time.

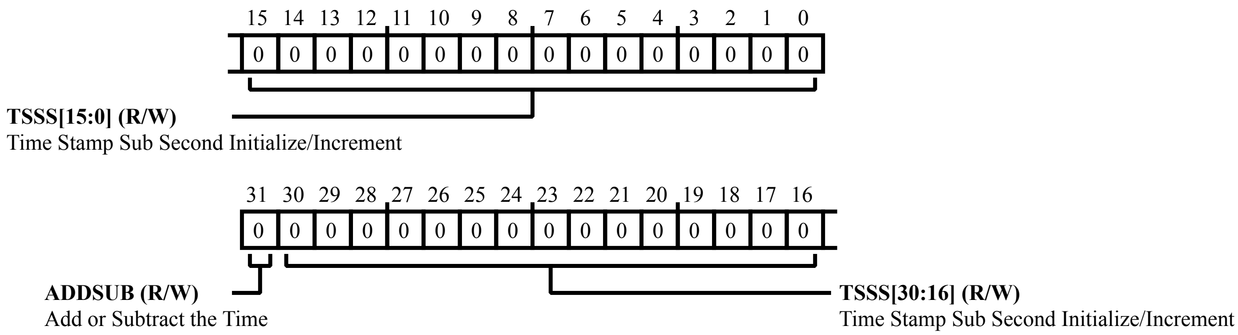

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSSS[15:0] (R/W)
Time Stamp Sub Second Initialize/Increment
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDSUB (R/W)
Add or Subtract the Time
TSSS[30:16] (R/W)
Time Stamp Sub Second Initialize/Increment

Figure 31-172: EMAC_TM_NSECUPDT Register Diagram

Table 31-204: EMAC_TM_NSECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ADDSUB	Add or Subtract the Time. The `EMAC_TM_NSECUPDT.ADDSUB` bit, when set, subtracts the time value with the contents of the update registers. When this bit is reset, the time value is added with the contents of the update registers.
30:0 (R/W)	TSSS	Time Stamp Sub Second Initialize/Increment. The value in the `EMAC_TM_NSECUPDT.TSSS` bit field indicates the time, in nanoseconds, to be initialized or added to or subtracted from the system time nanoseconds.

Time Stamp PPS Interval Register

The `EMAC_TM_PPS0INTVL` register contains the interval value for the time between rising edges (period) of PPS output.

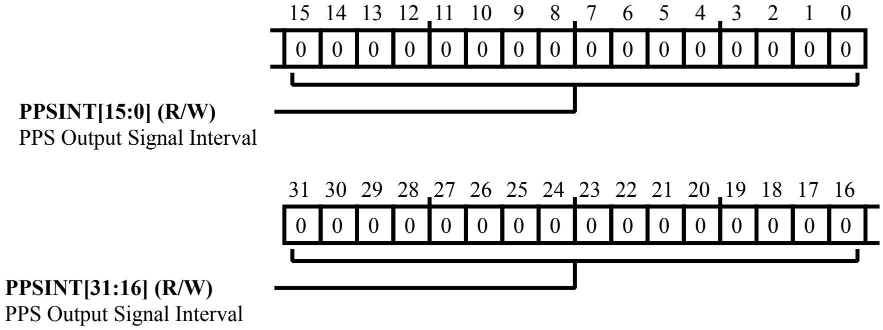

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Interval

Figure 31-173: EMAC_TM_PPS0INTVL Register Diagram

Table 31-205: EMAC_TM_PPS0INTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The `EMAC_TM_PPS0INTVL.PPSINT` bits store the interval between the rising edges of PPS signal output in terms of units of sub-second increment value. You need to program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20ns), and desired interval between rising edges of PPS signal output is 100ns (that is, 5 units of sub-second increment value), then you should program value 4 (5-1) in this register.

Time Stamp Target Time Nanoseconds Register

The EMAC_TM_PPS0NTGTM register contains the high 32 bits of the target nanoseconds field for comparison to the corresponding system time field.

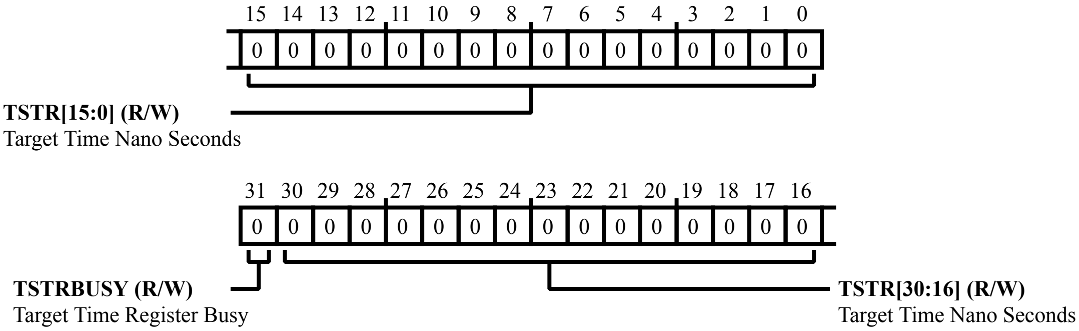

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target Time Nano Seconds
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRBUSY (R/W)
Target Time Register Busy
TSTR[30:16] (R/W)
Target Time Nano Seconds

Figure 31-174: EMAC_TM_PPS0NTGTM Register Diagram

Table 31-206: EMAC_TM_PPS0NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TSTRBUSY	Target Time Register Busy. The EMAC_TM_PPS0NTGTM.TSTRBUSY bit is set when Flexible PPS is enabled and the PPS Frequency Control bit field in the PPS Control register is programmed to 0001, 0010 or 0100. Programming the PPS Frequency Control bit field to 0001, 0010 or 0100, instructs the core to synchronize the EMAC_TM_PPS0TGTM and EMAC_TM_NSEC registers to the PTP clock domain. The EMAC clears this bit after synchronizing the EMAC_TM_PPS0TGTM and EMAC_TM_NSEC registers to the PTP clock domain The application must not update the EMAC_TM_PPS0TGTM and EMAC_TM_NSEC registers when this bit is read as 1. Otherwise, the synchronization of the previous programmed time gets corrupted.
30:0 (R/W)	TSTR	Target Time Nano Seconds. The EMAC_TM_PPS0NTGTM.TSTR bit field stores the time in (signed) nanoseconds. When the value of the time stamp matches the both EMAC_TM_PPS0TGTM and EMAC_TM_NSEC registers, based on the Target Time Register Mode bit field in the PPS control register, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled). This value should not exceed 0x3B9A_C9FF when the Target Time Register Mode bit field is set. The actual start or stop time of the PPS signal output may have an error margin up to one unit of sub-second increment value.

Time Stamp Target Time Seconds Register

The `EMAC_TM_PPS0TGTM` register contains the high 32 bits of the target seconds field for comparison to the corresponding system time field.

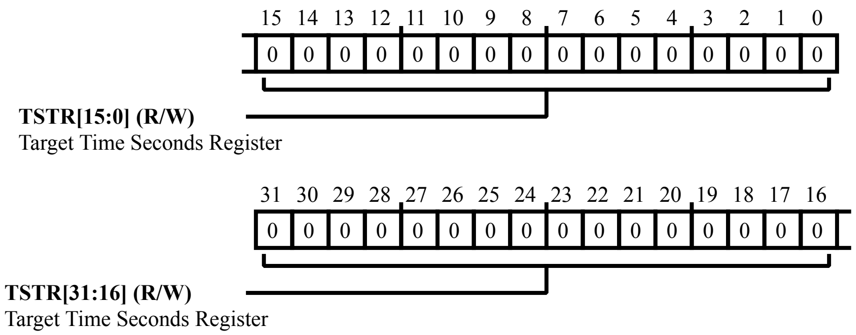

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target Time Seconds Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[31:16] (R/W)
Target Time Seconds Register

Figure 31-175: EMAC_TM_PPS0TGTM Register Diagram

Table 31-207: EMAC_TM_PPS0TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target Time Seconds Register. The `EMAC_TM_PPS0TGTM.TSTR` bit field stores the time in seconds. When the time stamp value matches or exceeds both the value in this field and the value in the `EMAC_TM_NSEC` register, based on the selection in the Target Time Register Mode bit field, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled).

PPS Width Register

The EMAC_TM_PPS0WIDTH register contains the interval value for the time between a rising and the next falling edge (width) of PPS output.

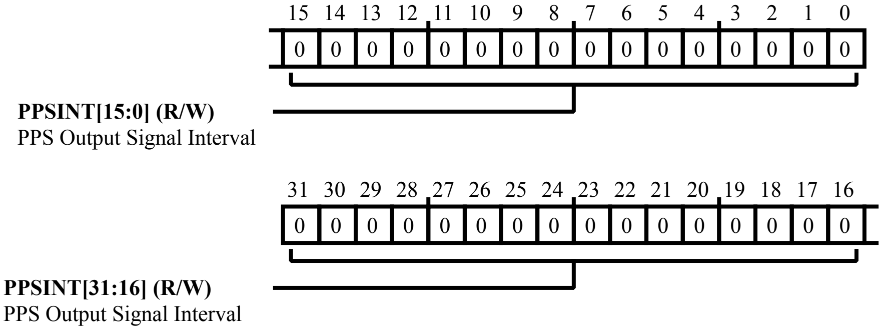

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Interval

Figure 31-176: EMAC_TM_PPS0WIDTH Register Diagram

Table 31-208: EMAC_TM_PPS0WIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The `EMAC_TM_PPS0WIDTH.PPSINT` bits store the interval between a rising edge and the next falling edge (width) of PPS output in terms of units of sub second increment value. Program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20 ns) and the desired width of the PPS signal output is 60 ns (3 units of sub-second increment value), program the value 2 (3-1) in this register.

PPS 1 Interval Register

The `EMAC_TM_PPS1INTVL` register contains the number of units of sub-second increment value between the rising edges of PPS0 signal output.

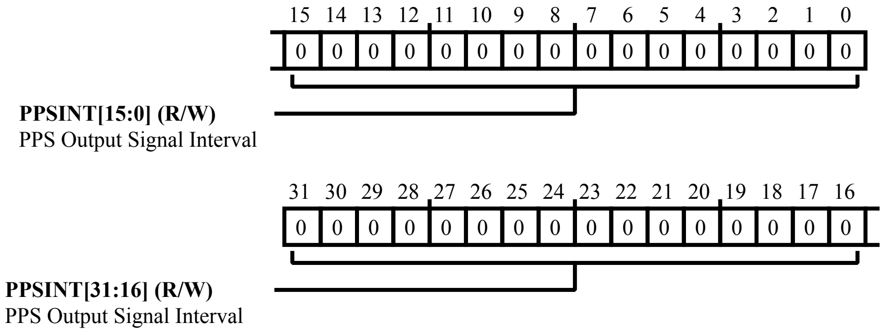

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Interval

Figure 31-177: EMAC_TM_PPS1INTVL Register Diagram

Table 31-209: EMAC_TM_PPS1INTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The `EMAC_TM_PPS1INTVL.PPSINT` bit field contains the number of units of sub-second increment value between the rising edges of PPS0 signal output.

PPS 1 Target Time Nanoseconds Register

The `EMAC_TM_PPS1NTGTM` register is present only when the IEEE 1588 time-stamp feature is selected without external time-stamp input.

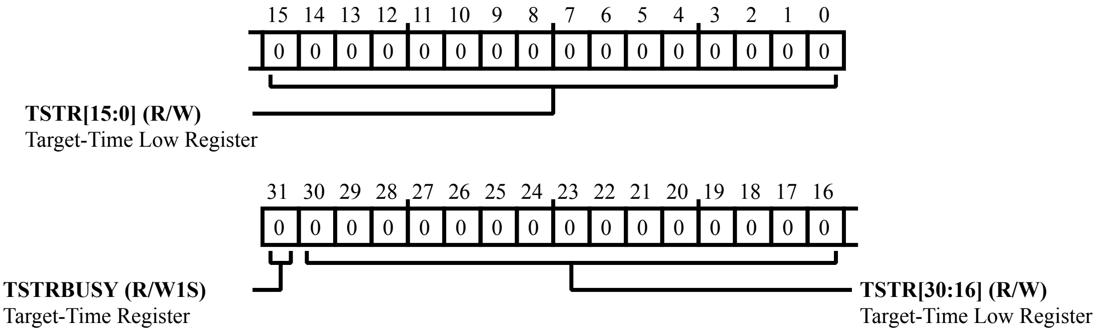

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target-Time Low Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRBUSY (R/W1S)
Target-Time Register
TSTR[30:16] (R/W)
Target-Time Low Register

Figure 31-178: EMAC_TM_PPS1NTGTM Register Diagram

Table 31-210: EMAC_TM_PPS1NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	TSTRBUSY	Target-Time Register. The `EMAC_TM_PPS1NTGTM.TSTRBUSY` bit field is cleared to 1b0 by the core (self clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.
30:0 (R/W)	TSTR	Target-Time Low Register.

PPS 1 Target Time Seconds Register

The `EMAC_TM_PPS1TGTM` register schedule an interrupt event when the system time exceeds the value programmed in these registers.

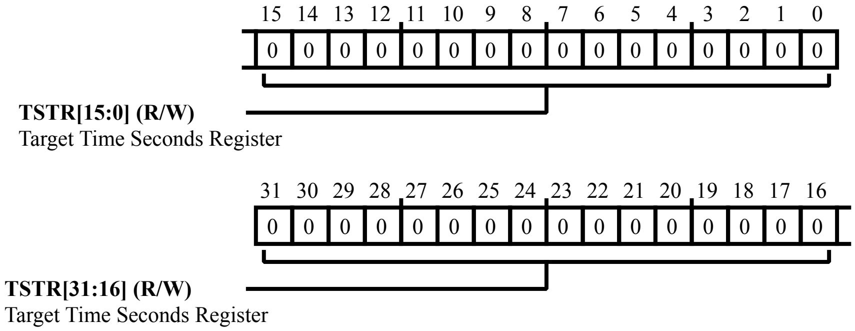

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target Time Seconds Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[31:16] (R/W)
Target Time Seconds Register

Figure 31-179: EMAC_TM_PPS1TGTM Register Diagram

Table 31-211: EMAC_TM_PPS1TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target Time Seconds Register. The `EMAC_TM_PPS1TGTM.TSTR` bit field stores the time in seconds. When the time-stamp value matches or exceeds both target time-stamp registers, the MAC starts or stops the PPS signal output and generates an interrupt.

PPS 1 Width Register

The `EMAC_TM_PPS1WIDTH` register contains the number of units of sub-second increment value between the rising and corresponding falling edges of the PPS0 signal output

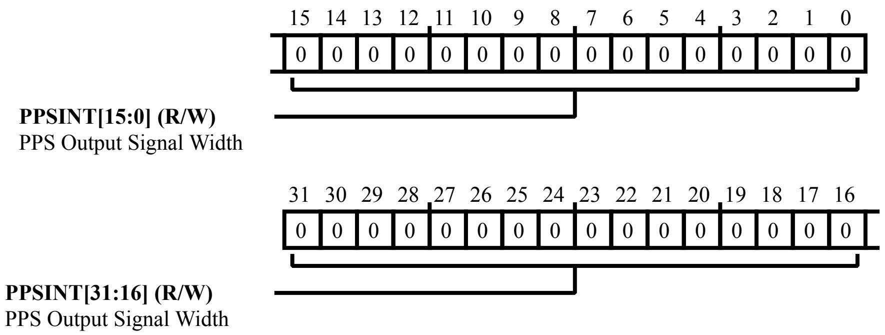

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Width

Figure 31-180: EMAC_TM_PPS1WIDTH Register Diagram

Table 31-212: EMAC_TM_PPS1WIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Width. The `EMAC_TM_PPS1WIDTH.PPSINT` bits store the interval between a rising edge and the next falling edge (width) of PPS output in terms of units of sub second increment value. Program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20 ns) and the desired width of the PPS signal output is 60 ns (3 units of sub-second increment value), program the value 2 (3-1) in this register.

PPS 2 Interval Register

The EMAC_TM_PPS2INTVL register contains the number of units of sub-second increment value between the rising edges of PPS0 signal output

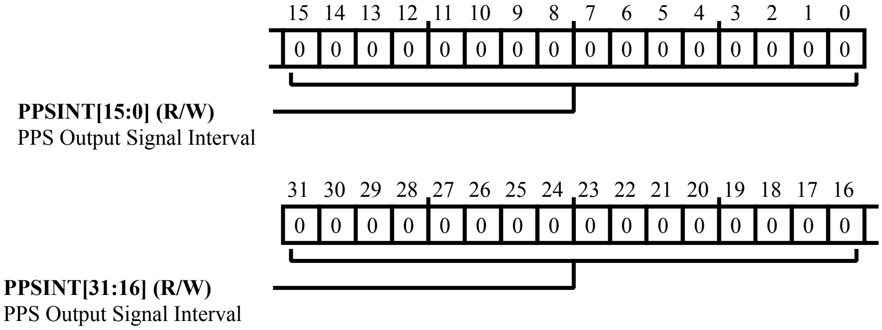

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Interval

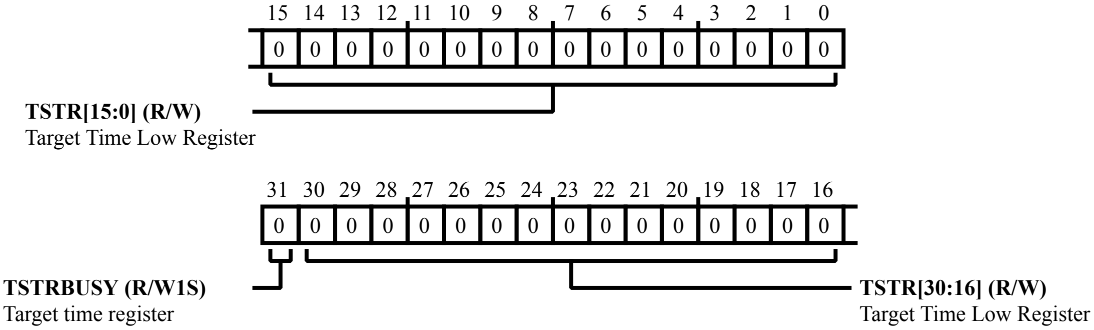
Figure 31-181: EMAC_TM_PPS2INTVL Register Diagram

Table 31-213: EMAC_TM_PPS2INTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The `EMAC_TM_PPS2INTVL.PPSINT` bit field contains the number of units of sub-second increment value between the rising edges of PPS2 signal output.

PPS 2 Target Time Nanoseconds Register

The EMAC_TM_PPS2NTGTM register is present only when the IEEE 1588 Time stamp feature is selected without external time stamp input.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target Time Low Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRBUSY (R/W1S)
Target time register
TSTR[30:16] (R/W)
Target Time Low Register

Figure 31-182: EMAC_TM_PPS2NTGTM Register Diagram

Table 31-214: EMAC_TM_PPS2NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	TSTRBUSY	Target time register. The EMAC_TM_PPS2NTGTM.TSTRBUSY bit field is cleared to 1b0 by the core (self clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.
30:0 (R/W)	TSTR	Target Time Low Register.

PPS 2 Target Time Seconds Register

The `EMAC_TM_PPS2TGTM` register schedule an interrupt event when the system time exceeds the value programmed in these registers.

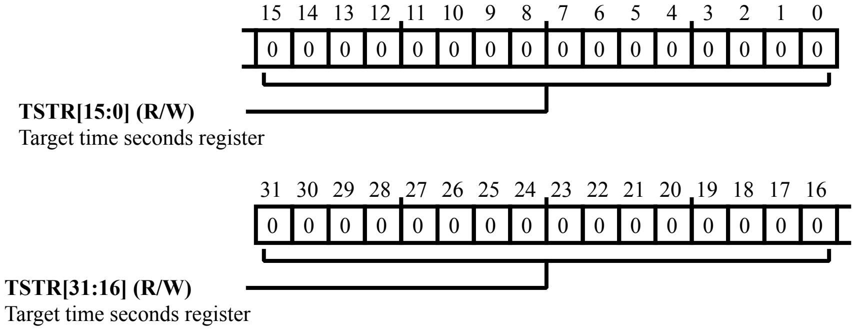

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target time seconds register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[31:16] (R/W)
Target time seconds register

Figure 31-183: EMAC_TM_PPS2TGTM Register Diagram

Table 31-215: EMAC_TM_PPS2TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target time seconds register. The `EMAC_TM_PPS2TGTM.TSTR` bit field stores the time in seconds. When the time-stamp value matches or exceeds both target time-stamp registers, the MAC starts or stops the PPS signal output and generates an interrupt.

PPS 2 Width Register

The `EMAC_TM_PPS2WIDTH` register contains the number of units of sub-second increment value between the rising and corresponding falling edges of the PPS0 signal output

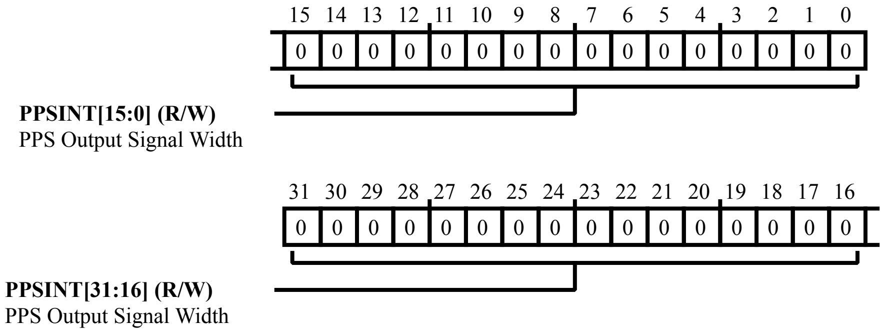

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Width

Figure 31-184: EMAC_TM_PPS2WIDTH Register Diagram

Table 31-216: EMAC_TM_PPS2WIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Width. The `EMAC_TM_PPS2WIDTH.PPSINT` bits store the interval between a rising edge and the next falling edge (width) of PPS output in terms of units of sub second increment value. Program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20 ns) and the desired width of the PPS signal output is 60 ns (3 units of sub-second increment value), program the value 2 (3-1) in this register.

PPS 3 Interval Register

The `EMAC_TM_PPS3INTVL` register contains the number of units of sub-second increment value between the rising edges of PPS0 signal output

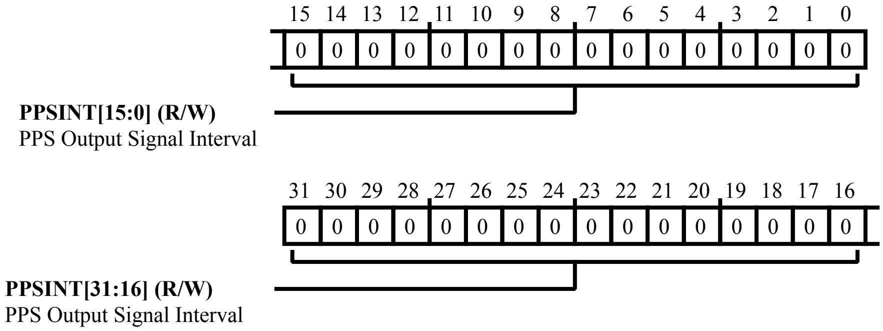

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Interval
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Interval

Figure 31-185: EMAC_TM_PPS3INTVL Register Diagram

Table 31-217: EMAC_TM_PPS3INTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The `EMAC_TM_PPS3INTVL.PPSINT` bit field contains the number of units of sub-second increment value between the rising edges of PPS3 signal output.

PPS 3 Target Time Nanoseconds Register

The `EMAC_TM_PPS3NTGTM` register is present only when the IEEE 1588 Time stamp feature is selected without external time stamp input.

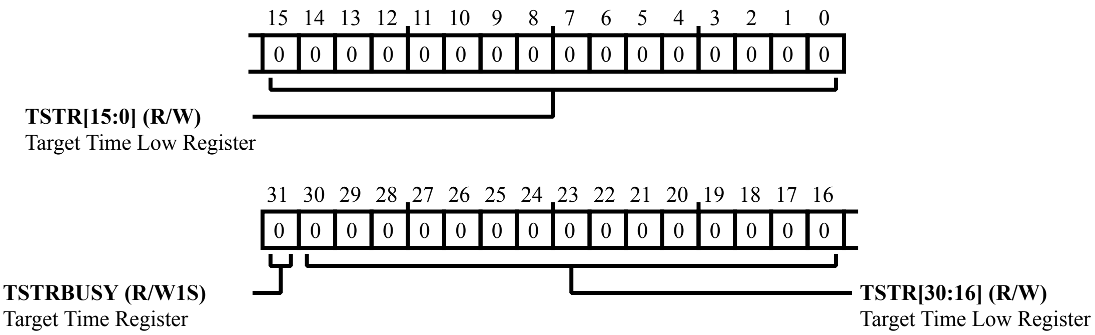

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target Time Low Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRBUSY (R/W1S)
Target Time Register
TSTR[30:16] (R/W)
Target Time Low Register

Figure 31-186: EMAC_TM_PPS3NTGTM Register Diagram

Table 31-218: EMAC_TM_PPS3NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	TSTRBUSY	Target Time Register. The `EMAC_TM_PPS3NTGTM.TSTRBUSY` bit field is cleared to 1b0 by the core (self clear). The application cannot clear this type of field, and a register write of 1b0 to this bit has no effect on this field.
30:0 (R/W)	TSTR	Target Time Low Register.

PPS 3 Target Time Seconds Register

The `EMAC_TM_PPS3TGTM` register schedule an interrupt event when the system time exceeds the value programmed in these registers.

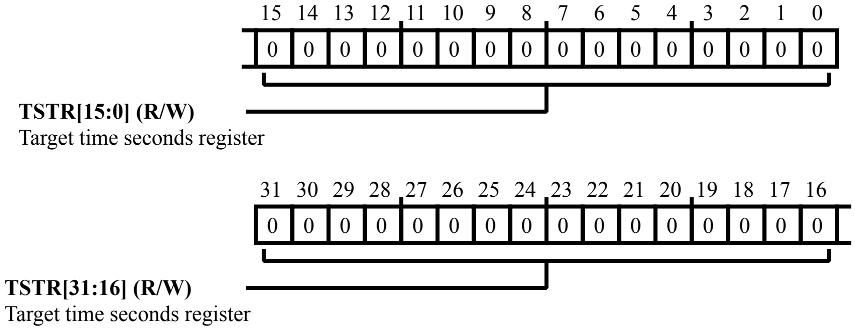

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[15:0] (R/W)
Target time seconds register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTR[31:16] (R/W)
Target time seconds register

Figure 31-187: EMAC_TM_PPS3TGTM Register Diagram

Table 31-219: EMAC_TM_PPS3TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target time seconds register. The `EMAC_TM_PPS3TGTM.TSTR` bit field stores the time in seconds. When the time-stamp value matches or exceeds both target time-stamp registers, the MAC starts or stops the PPS signal output and generates an interrupt.

PPS 3 Width Register

The `EMAC_TM_PPS3WIDTH` register contains the number of units of sub-second increment value between the rising and corresponding falling edges of the PPS0 signal output

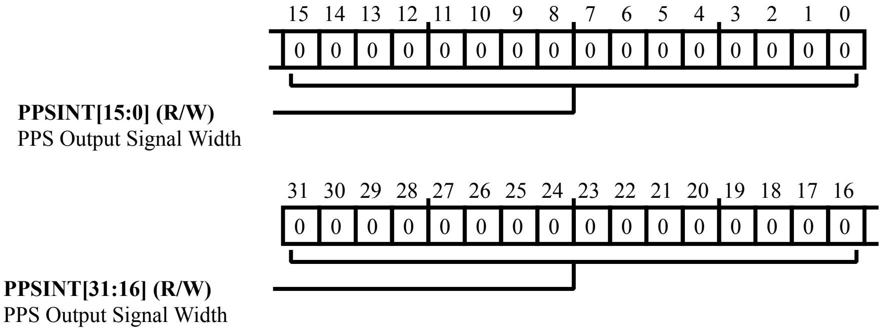

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[15:0] (R/W)
PPS Output Signal Width
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPSINT[31:16] (R/W)
PPS Output Signal Width

Figure 31-188: EMAC_TM_PPS3WIDTH Register Diagram

Table 31-220: EMAC_TM_PPS3WIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Width. The `EMAC_TM_PPS3WIDTH.PPSINT` bits store the interval between a rising edge and the next falling edge (width) of PPS output in terms of units of sub second increment value. Program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20 ns) and the desired width of the PPS signal output is 60 ns (3 units of sub-second increment value), program the value 2 (3-1) in this register.

PPS Control Register

The `EMAC_TM_PPSCTL` register controls the interval of PPS output.

When the `EMAC_TM_PPSCTL.PPSEN` bit is disabled (=0, fixed PPS output), the PPS Frequency Control (PPSCTL0) bits control the behavior of the PPS output signal. The default value of PPSCTRL is 0000 and the PPS output is 1 pulse every second. For other values of PPSCTRL, the PPS output becomes a generated clock. (See bit enumerations for frequencies.) In the binary rollover mode, the PPS output has a duty cycle of 50 percent with these frequencies. In the digital rollover mode, the PPS output frequency is an average number. The actual clock is of different frequency that gets synchronized every second. This behavior is because of the non-linear toggling of the bits in the digital rollover mode in System Time - Nanoseconds Register.

When the `EMAC_TM_PPSCTL.PPSEN` bit is enabled (=1, flexible PPS output), the PPS Frequency Control bits function as PPSCMD. (See bit enumerations for commands.) Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are all-zero.

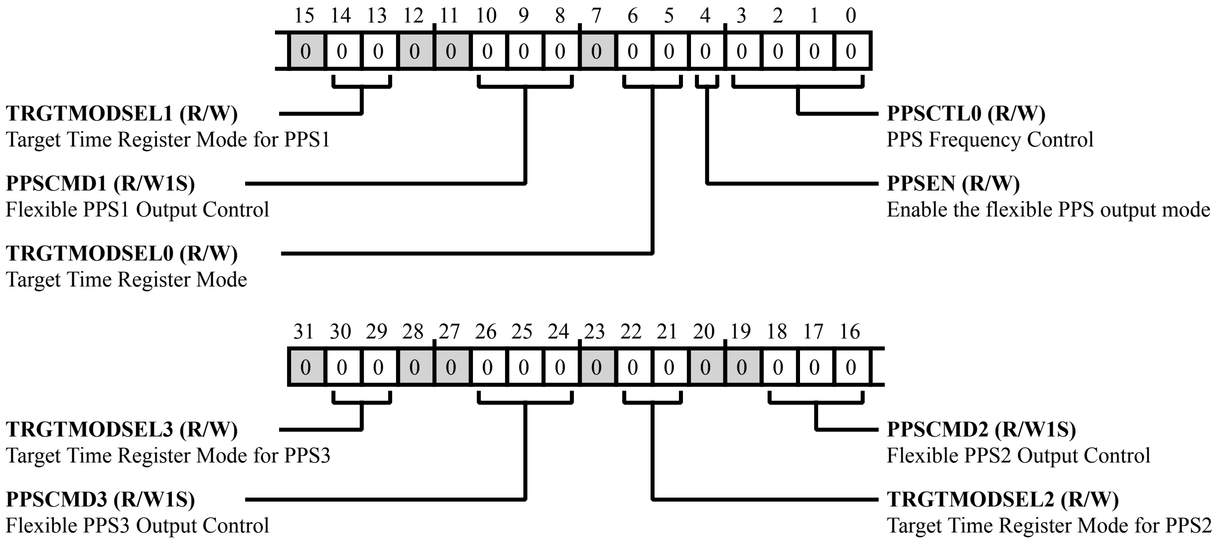

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRGTMODSEL1 (R/W)
Target Time Register Mode for PPS1
PPSCMD1 (R/W1S)
Flexible PPS1 Output Control
TRGTMODSEL0 (R/W)
Target Time Register Mode
PPSCTL0 (R/W)
PPS Frequency Control
PPSEN (R/W)
Enable the flexible PPS output mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRGTMODSEL3 (R/W)
Target Time Register Mode for PPS3
PPSCMD3 (R/W1S)
Flexible PPS3 Output Control
PPSCMD2 (R/W1S)
Flexible PPS2 Output Control
TRGTMODSEL2 (R/W)
Target Time Register Mode for PPS2

Figure 31-189: EMAC_TM_PPSCTL Register Diagram

Table 31-221: EMAC_TM_PPSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:29 (R/W)	TRGTMODSEL3	Target Time Register Mode for PPS3. The `EMAC_TM_PPSCTL.TRGTMODSEL3` bits indicates the Target Time registers mode for PPS3 output signal.

Table 31-221: EMAC_TM_PPSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26:24 (R/W1S)	PPSCMD3	Flexible PPS3 Output Control. The `EMAC_TM_PPSCTL.PPSCMD3` bits controls the flexible PPS3 output signal.	
22:21 (R/W)	TRGTMODSEL2	Target Time Register Mode for PPS2. The `EMAC_TM_PPSCTL.TRGTMODSEL2` bits indicates the Target Time registers mode for PPS2 output signal.	
18:16 (R/W1S)	PPSCMD2	Flexible PPS2 Output Control. The `EMAC_TM_PPSCTL.PPSCMD2` bits controls the flexible PPS3 output signal.	
14:13 (R/W)	TRGTMODSEL1	Target Time Register Mode for PPS1. The `EMAC_TM_PPSCTL.TRGTMODSEL1` bits indicates the Target Time registers mode for PPS1 output signal.	
10:8 (R/W1S)	PPSCMD1	Flexible PPS1 Output Control. The `EMAC_TM_PPSCTL.PPSCMD1` bits controls the flexible PPS1 output signal.	
6:5 (R/W)	TRGTMODSEL0	Target Time Register Mode. The `EMAC_TM_PPSCTL.TRGTMODSEL0` bits select the target time register mode.	
		0	Interrupt Only The Target Time registers are programmed only for interrupt event generation.
		1	Reserved
		2	Interrupt and PPS Start/Stop The Target Time registers are programmed for interrupt event and for starting or stopping the PPS output signal generation.
		3	PPS Start/Stop Only The Target Time registers are programmed only for starting or stopping the PPS output signal generation. No interrupt is asserted.
4 (R/W)	PPSEN	Enable the flexible PPS output mode. The `EMAC_TM_PPSCTL.PPSEN` bit enables PPS operation. When set low, the PPS Frequency Control field controls frequency of Fixed PPS output. When set high, PPS Frequency Control field is used to command Flexible PPS output.	

Table 31-221: EMAC_TM_PPSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	PPSCTL0	PPS Frequency Control. When the `EMAC_TM_PPSCTL.PPSEN` bit is disabled (=0, fixed PPS output), the `EMAC_TM_PPSCTL.PPSCTL0` bits control the behavior of the PPS output signal. When the `EMAC_TM_PPSCTL.PPSEN` bit is enabled (=1, flexible PPS output), the `EMAC_TM_PPSCTL.PPSCTL0` bits function as PPSCMD. (See bit enumerations for PPS output frequency, rollover, and PPS commands.) Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are all-zero. All values not shown in the bit enumerations are reserved. For more information about the `EMAC_TM_PPSCTL.PPSCTL0` bits, see the pulse-per-second functional description.	
		0	CMD=No Command
		1	CMD=START Single; BR=2kHz; DR=1kHz For more info, see register description.
		2	CMD=START Pulse; BR=4kHz; DR=2kHz For more info, see register description.
		3	CMD=Cancel START; BR=8kHz; DR=4kHz For more info, see register description.
		4	CMD=STOP Pulse Time; BR=16kHz; DR=8kHz For more info, see register description.
		5	CMD=STOP Pulse Now For more info, see register description.
		6	CMD=Cancel STOP Pulse For more info, see register description.

Time Stamp Low Seconds Register

The `EMAC_TM_SEC` register contains the lower 32 bits of the seconds field of the system time.

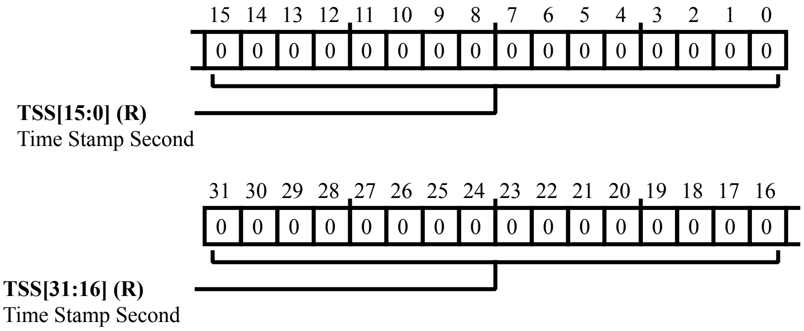

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSS[15:0] (R)
Time Stamp Second
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSS[31:16] (R)
Time Stamp Second

Figure 31-190: EMAC_TM_SEC Register Diagram

Table 31-222: EMAC_TM_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TSS	Time Stamp Second. The value in the `EMAC_TM_SEC.TSS` bit field indicates the current value in seconds of the System Time maintained by the core.

Time Stamp Seconds Update Register

The `EMAC_TM_SECUPDT` register contains the low 32 bits to be added to, subtracted from, or written to the seconds field of the system time.

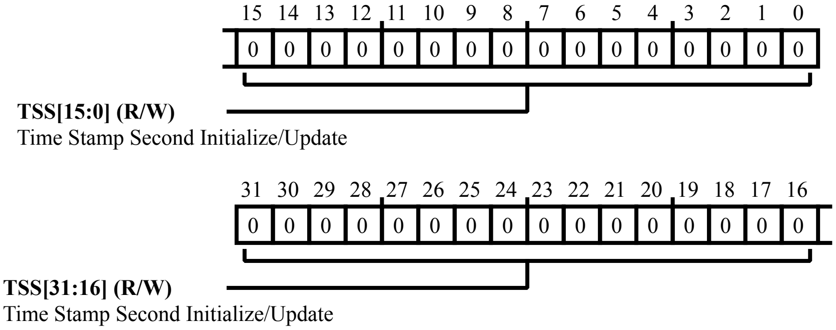

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSS[15:0] (R/W)
Time Stamp Second Initialize/Update
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSS[31:16] (R/W)
Time Stamp Second Initialize/Update

Figure 31-191: EMAC_TM_SECUPDT Register Diagram

Table 31-223: EMAC_TM_SECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSS	Time Stamp Second Initialize/Update. The value in the `EMAC_TM_SECUPDT.TSS` bit field indicates the time, in seconds, to be initialized or added to or subtracted from the system time seconds.

Time Stamp Status Register

The `EMAC_TM_STMPSTAT` register contains the PTP status.

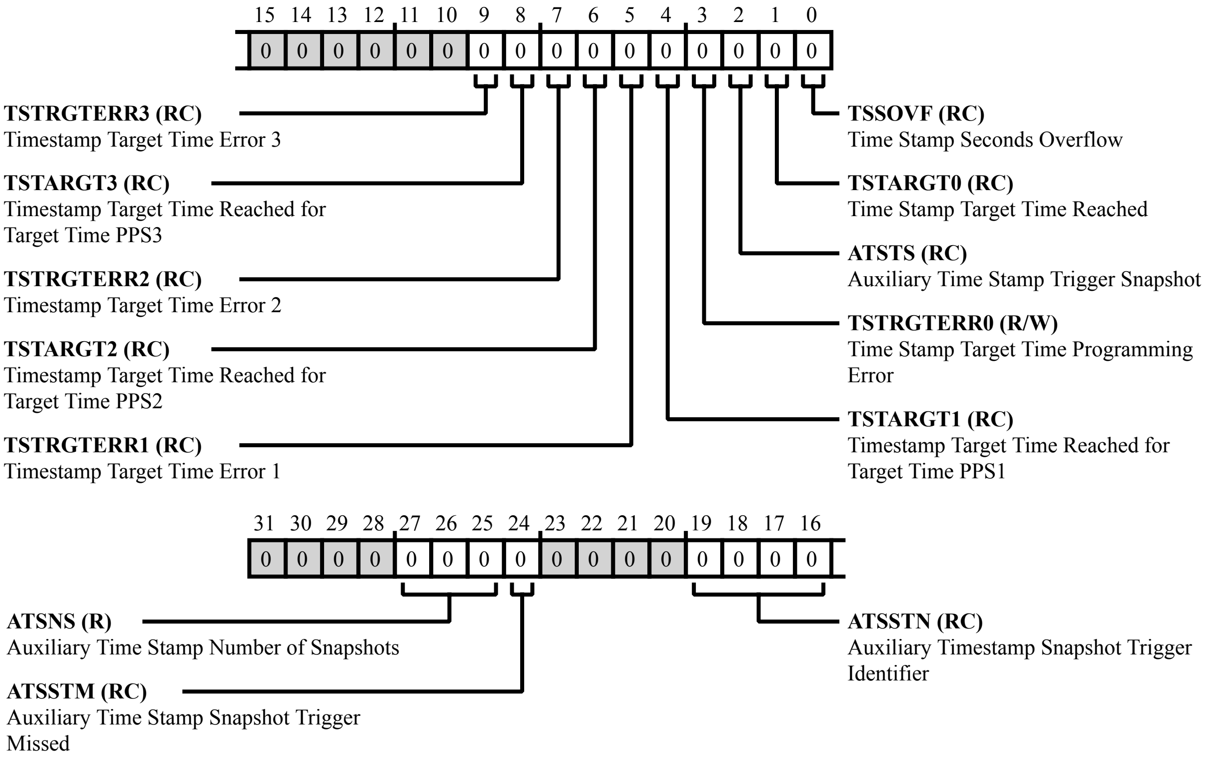

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRGTERR3 (RC)
Timestamp Target Time Error 3
TSTARGT3 (RC)
Timestamp Target Time Reached for Target Time PPS3
TSTRGTERR2 (RC)
Timestamp Target Time Error 2
TSTARGT2 (RC)
Timestamp Target Time Reached for Target Time PPS2
TSTRGTERR1 (RC)
Timestamp Target Time Error 1
TSSOVF (RC)
Time Stamp Seconds Overflow
TSTARGT0 (RC)
Time Stamp Target Time Reached
ATSTS (RC)
Auxiliary Time Stamp Trigger Snapshot
TSTRGTERR0 (R/W)
Time Stamp Target Time Programming Error
TSTARGT1 (RC)
Timestamp Target Time Reached for Target Time PPS1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ATSNS (R)
Auxiliary Time Stamp Number of Snapshots
ATSSTM (RC)
Auxiliary Time Stamp Snapshot Trigger Missed
ATSSTN (RC)
Auxiliary Timestamp Snapshot Trigger Identifier

Figure 31-192: EMAC_TM_STMPSTAT Register Diagram

Table 31-224: EMAC_TM_STMPSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:25 (R/NW)	ATSNS	Auxiliary Time Stamp Number of Snapshots. The `EMAC_TM_STMPSTAT.ATSNS` bits indicate the number of Snapshots available in the FIFO. A value of 4 (100) indicates that the Auxiliary Snapshot FIFO is full. These bits are cleared (to 000) when the Auxiliary snapshot FIFO clear bit is set.
24 (RC/NW)	ATSSTM	Auxiliary Time Stamp Snapshot Trigger Missed. The `EMAC_TM_STMPSTAT.ATSSTM` bit is set when the Auxiliary time stamp snapshot FIFO is full and external trigger was set. This indicates that the latest snapshot was not stored in the FIFO.
19:16 (RC/NW)	ATSSTN	Auxiliary Timestamp Snapshot Trigger Identifier. The `EMAC_TM_STMPSTAT.ATSSTN` bit identify the Auxiliary trigger inputs for which the timestamp available in the Auxiliary Snapshot Register is applicable.

Table 31-224: EMAC_TM_STMPSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (RC/NW)	TSTRGTERR3	Timestamp Target Time Error 3. The `EMAC_TM_STMPSTAT.TSTRGTERR3` bit when set the target time being programmed in PPS3 Target Time High Register and PPS3 Target Time Low Register, is already elapsed.
8 (RC/NW)	TSTARGT3	Timestamp Target Time Reached for Target Time PPS3. The `EMAC_TM_STMPSTAT.TSTARGT3` bit indicates that the value of system time is greater than or equal to the value specified in PPS3 Target Time High Register and PPS3 Target Time Low Register.
7 (RC/NW)	TSTRGTERR2	Timestamp Target Time Error 2. The `EMAC_TM_STMPSTAT.TSTRGTERR2` bit when set the target time being programmed in PPS2 Target Time High Register and PPS2 Target Time Low Register, is already elapsed.
6 (RC/NW)	TSTARGT2	Timestamp Target Time Reached for Target Time PPS2. The `EMAC_TM_STMPSTAT.TSTARGT2` bit indicates that the value of system time is greater than or equal to the value specified in PPS2 Target Time High Register and PPS2 Target Time Low Register.
5 (RC/NW)	TSTRGTERR1	Timestamp Target Time Error 1. The `EMAC_TM_STMPSTAT.TSTRGTERR1` bit when set the target time being programmed in PPS1 Target Time High Register and PPS1 Target Time Low Register, is already elapsed.
4 (RC/NW)	TSTARGT1	Timestamp Target Time Reached for Target Time PPS1. The `EMAC_TM_STMPSTAT.TSTARGT1` bit indicates that the value of system time is greater than or equal to the value specified in PPS1 Target Time High Register and PPS1 Target Time Low Register.
3 (R/W)	TSTRGTERR0	Time Stamp Target Time Programming Error. The `EMAC_TM_STMPSTAT.TSTRGTERR0` bit is set when the target time, which is being programmed in the `EMAC_TM_SEC` and `EMAC_TM_NSEC` registers, has already elapsed. This bit is cleared when read by the application.
2 (RC/NW)	ATSTS	Auxiliary Time Stamp Trigger Snapshot. The `EMAC_TM_STMPSTAT.ATSTS` bit is set high when the auxiliary snapshot is written to the FIFO.
1 (RC/NW)	TSTARGT0	Time Stamp Target Time Reached. The `EMAC_TM_STMPSTAT.TSTARGT0` bit, when set, indicates the value of system time has reached or passed the value specified in the `EMAC_TM_PPS0TGTM` and `EMAC_TM_NSEC` registers.

Table 31-224: EMAC_TM_STMPSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (RC/NW)	TSSOVF	Time Stamp Seconds Overflow. The `EMAC_TM_STMPSTAT.TSSOVF` bit, when set, indicates that the seconds value of the time stamp (when supporting PTP version 2 format) has overflowed beyond 0xFFFF_FFFF.

Time Stamp Sub Second Increment Register

The EMAC_TM_SUBSEC register contains the value by which the system time nano second is incremented.

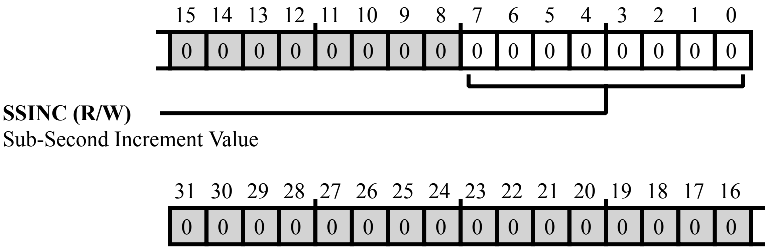

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SSINC (R/W)
Sub-Second Increment Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-193: EMAC_TM_SUBSEC Register Diagram

Table 31-225: EMAC_TM_SUBSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SSINC	Sub-Second Increment Value. The value in the `EMAC_TM_SUBSEC.SSINC` bits is accumulated every PTP clock cycle with the contents of the nanosecond register. For example, when PTP clock is 50 MHz (period is 20 ns), the processor should program 20 (0x14) when the EMAC_TM_NSEC register has an accuracy of 1 ns (`EMAC_TM_CTL.TSCTRLSSR` bit is set). When `EMAC_TM_CTL.TSCTRLSSR` is clear, the EMAC_TM_NSEC register has a resolution of ~0.465ns. In this case, the processor should program a value of 43 (0x2B) that is derived by 20ns/0.465.

Tx 1024- to Max-Byte Frames (Good/Bad) Register

The `EMAC_TX1024TOMAX_GB` register contains the count of the number of good and bad frames transmitted with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.

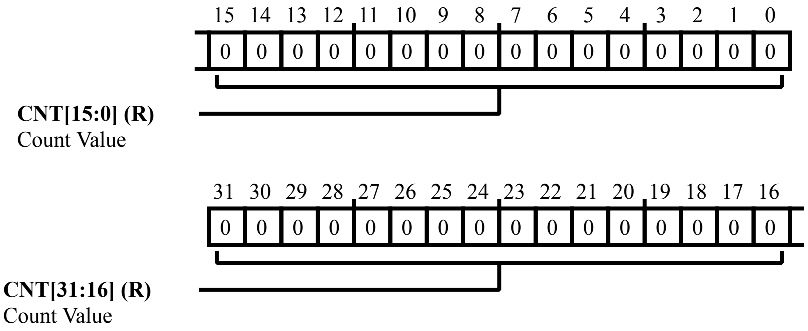

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-194: EMAC_TX1024TOMAX_GB Register Diagram

Table 31-226: EMAC_TX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 128- to 255-Byte Frames (Good/Bad) Register

The `EMAC_TX128TO255_GB` register contains the count of the number of good and bad frames transmitted with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.

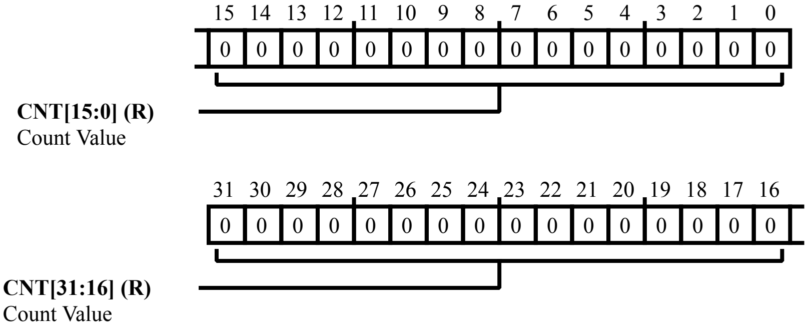

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-195: EMAC_TX128TO255_GB Register Diagram

Table 31-227: EMAC_TX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 256- to 511-Byte Frames (Good/Bad) Register

The `EMAC_TX256TO511_GB` register contains the count of the number of good and bad frames transmitted with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.

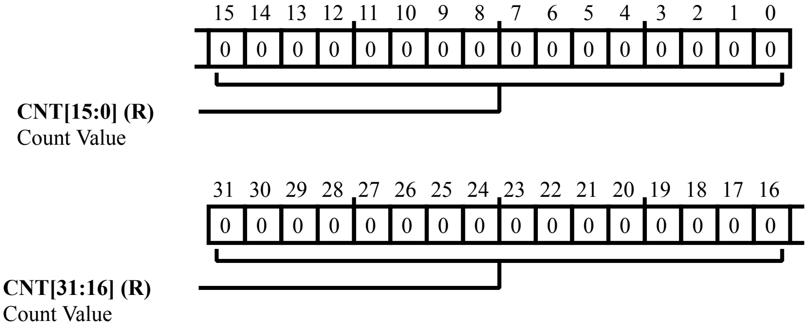

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-196: EMAC_TX256TO511_GB Register Diagram

Table 31-228: EMAC_TX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 512- to 1023-Byte Frames (Good/Bad) Register

The `EMAC_TX512TO1023_GB` register contains the count of the number of good and bad frames transmitted with length between 512 and 1023 (inclusive) bytes, exclusive of preamble and retried frames.

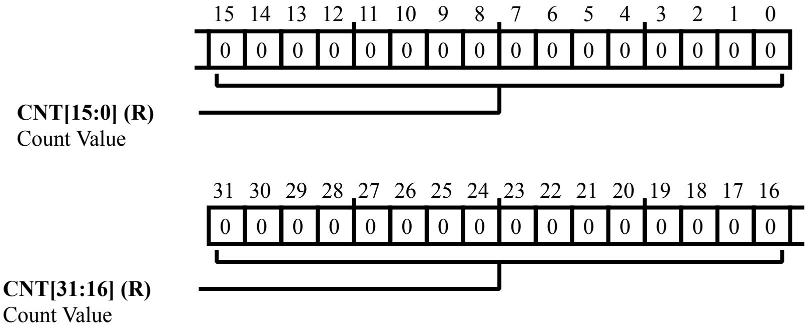

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-197: EMAC_TX512TO1023_GB Register Diagram

Table 31-229: EMAC_TX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 64-Byte Frames (Good/Bad) Register

The `EMAC_TX64_GB` register contains the count of the number of good and bad frames transmitted with length 64 bytes, exclusive of preamble and retried frames.

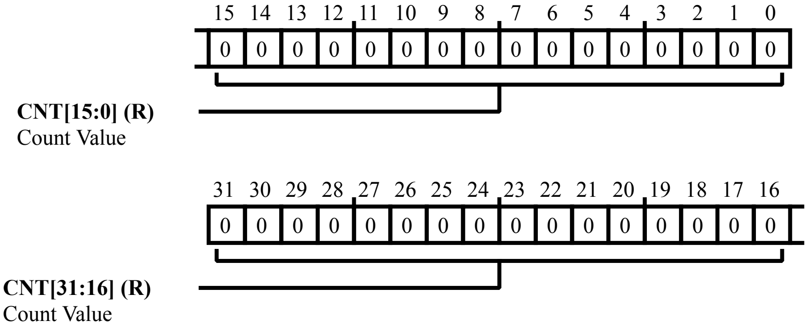

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-198: EMAC_TX64_GB Register Diagram

Table 31-230: EMAC_TX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 65- to 127-Byte Frames (Good/Bad) Register

The `EMAC_TX65TO127_GB` register contains the count of the number of good and bad frames transmitted with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.

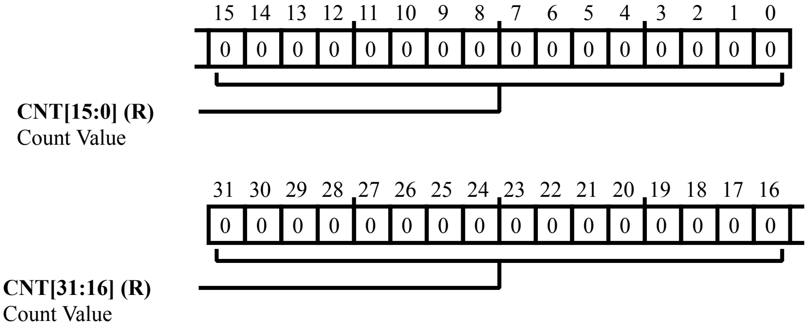

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-199: EMAC_TX65TO127_GB Register Diagram

Table 31-231: EMAC_TX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good) Register

The EMAC_TXBCASTFRM_G register contains the count of the number of good broadcast frames transmitted.

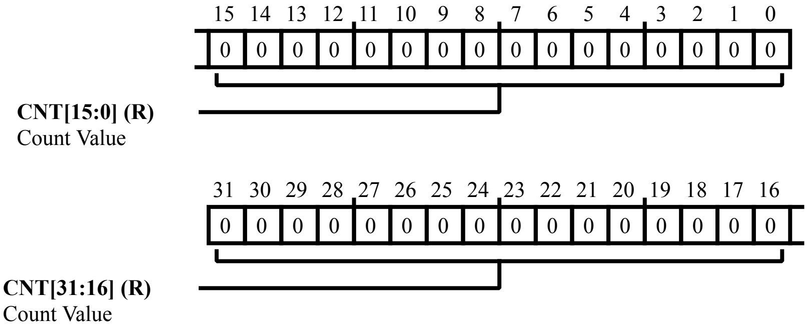

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-200: EMAC_TXBCASTFRM_G Register Diagram

Table 31-232: EMAC_TXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good/Bad) Register

The `EMAC_TXBCASTFRM_GB` register contains the count of the number of good and bad broadcast frames transmitted.

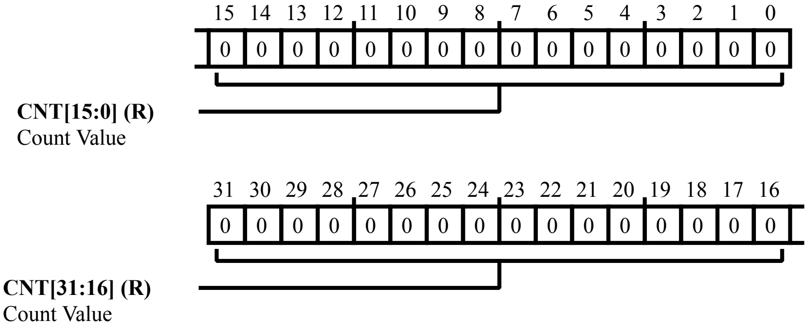

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-201: EMAC_TXBCASTFRM_GB Register Diagram

Table 31-233: EMAC_TXBCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Carrier Error Register

The `EMAC_TXCARR_ERR` register contains a count of the number of frames aborted due to carrier sense error (no carrier or loss of carrier).

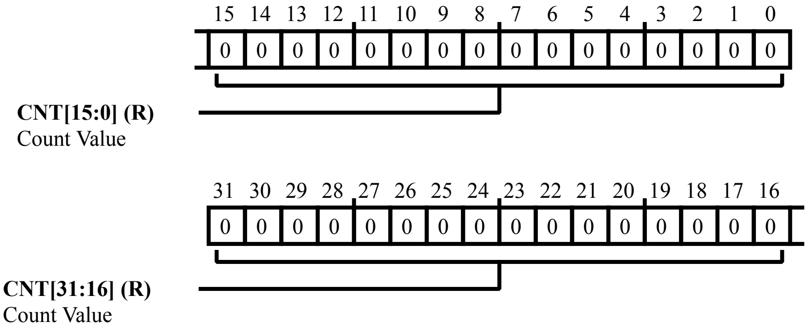

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-202: EMAC_TXCARR_ERR Register Diagram

Table 31-234: EMAC_TXCARR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Deferred Register

The `EMAC_TXDEFERRED` register contains a count of the number of successfully transmitted frames after a deferral in Half-duplex mode.

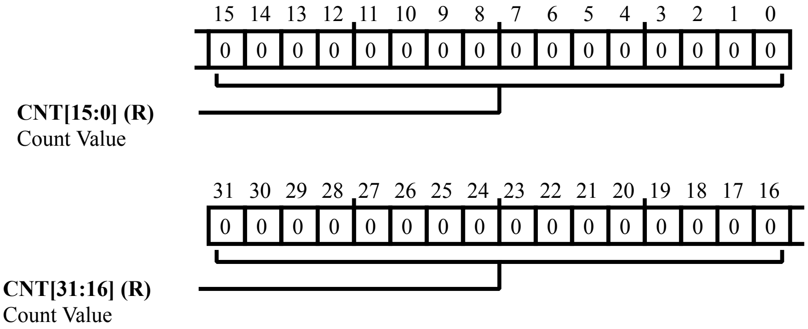

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-203: EMAC_TXDEFERRED Register Diagram

Table 31-235: EMAC_TXDEFERRED Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Collision Register

The `EMAC_TXEXCESSCOL` register contains a count of the number of frames aborted due to excessive (16) collision errors.

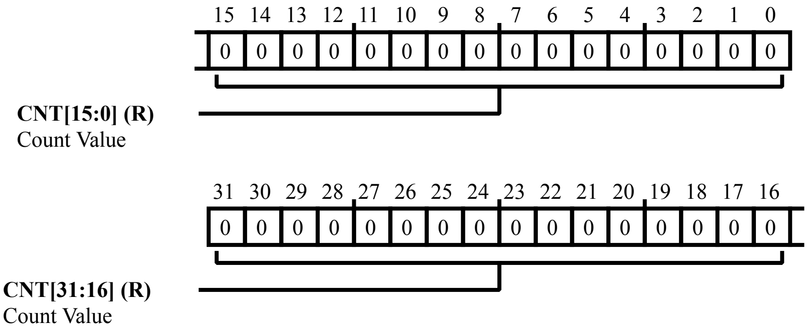

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-204: EMAC_TXEXCESSCOL Register Diagram

Table 31-236: EMAC_TXEXCESSCOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Deferral Register

The `EMAC_TXEXCESSDEF` register contains a count of the number of frames aborted due to excessive deferral error (deferred for more than two max-sized frame times).

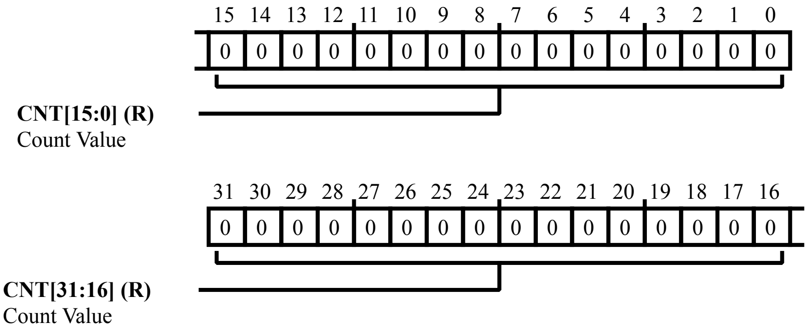

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-205: EMAC_TXEXCESSDEF Register Diagram

Table 31-237: EMAC_TXEXCESSDEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good) Register

The `EMAC_TXFRMCNT_G` register contains a count of the number of good frames transmitted.

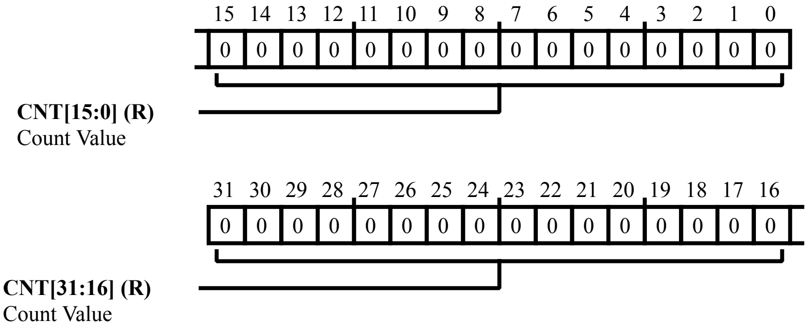

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-206: EMAC_TXFRMCNT_G Register Diagram

Table 31-238: EMAC_TXFRMCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good/Bad) Register

The EMAC_TXFRMCNT_GB register contains the count of the number of good and bad frames transmitted, exclusive of retried frames.

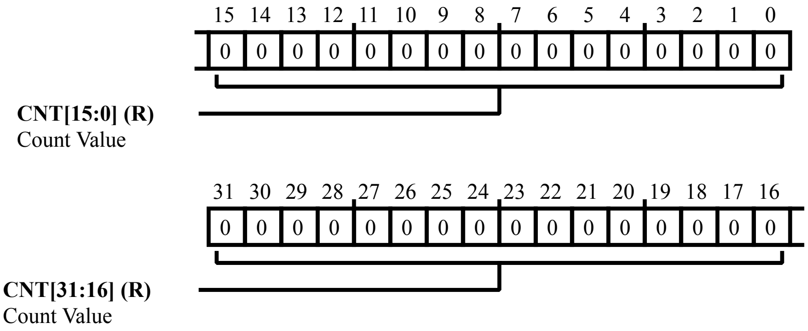

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-207: EMAC_TXFRMCNT_GB Register Diagram

Table 31-239: EMAC_TXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Late Collision Register

The EMAC_TXLATECOL register contains a count of the number of frames aborted due to late collision error.

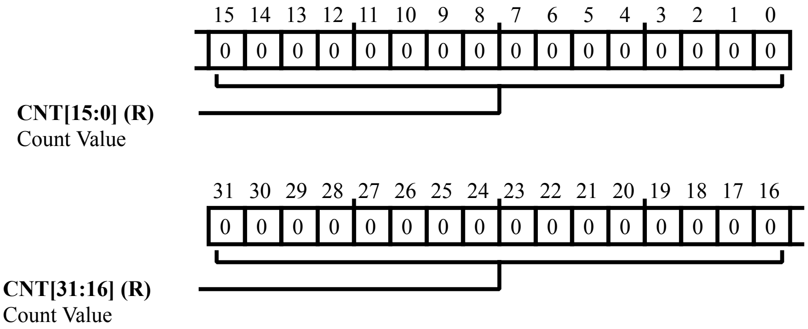

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-208: EMAC_TXLATECOL Register Diagram

Table 31-240: EMAC_TXLATECOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good) Register

The `EMAC_TXMCASTFRM_G` register contains the count of the number of good multicast frames transmitted.

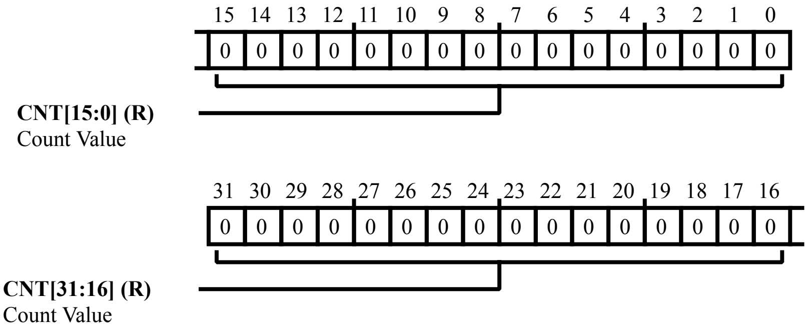

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-209: EMAC_TXMCASTFRM_G Register Diagram

Table 31-241: EMAC_TXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good/Bad) Register

The EMAC_TXMCASTFRM_GB register contains the count of the number of good and bad multicast frames transmitted.

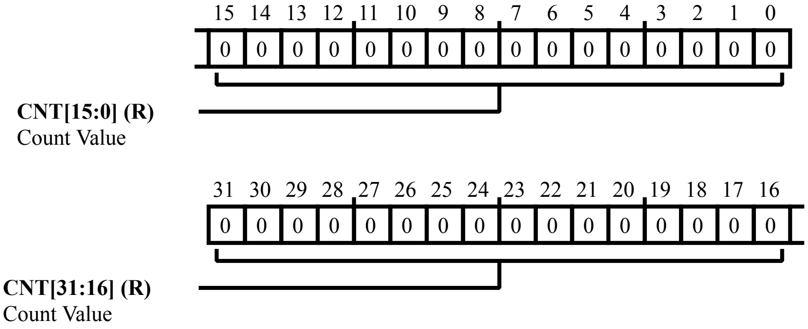

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-210: EMAC_TXMCASTFRM_GB Register Diagram

Table 31-242: EMAC_TXMCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multiple Collision (Good) Register

The EMAC_TXMULTCOL_G register contains a count of the number of successfully transmitted frames after more than a single collision in Half-duplex mode.

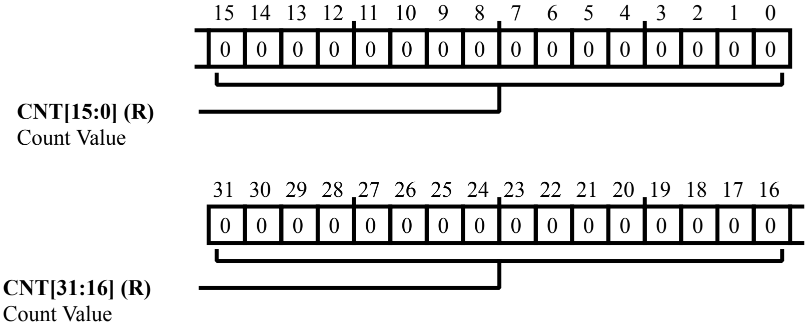

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-211: EMAC_TXMULTCOL_G Register Diagram

Table 31-243: EMAC_TXMULTCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Octet Count (Good) Register

The EMAC_TXOCTCNT_G register contains a count of the number of bytes transmitted, exclusive of preamble, in good frames only.

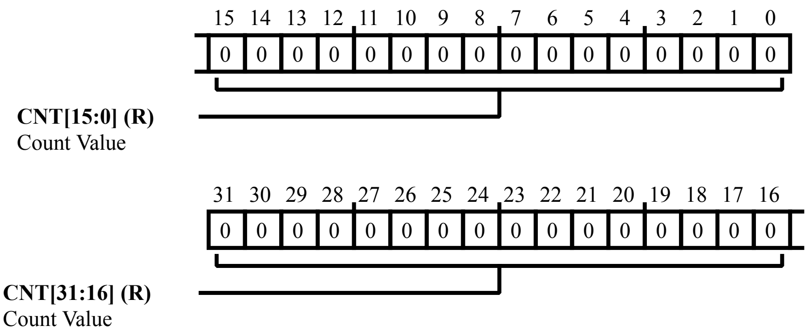

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-212: EMAC_TXOCTCNT_G Register Diagram

Table 31-244: EMAC_TXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx OCT Count (Good/Bad) Register

The `EMAC_TXOCTCNT_GB` register contains the count of the number of bytes transmitted, exclusive of the preamble and retried bytes, in good and bad frames.

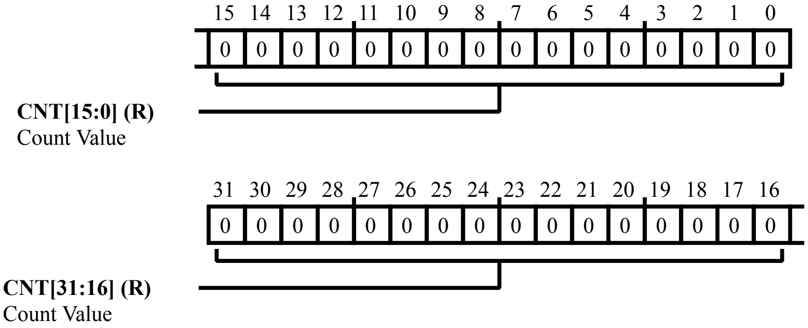

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-213: EMAC_TXOCTCNT_GB Register Diagram

Table 31-245: EMAC_TXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Number of Tx Frames (Good) greater than maxsize

The `EMAC_TXOVRSIZE_G` register contains a count of the number of good frames transmitted with length greater than the maxsize.

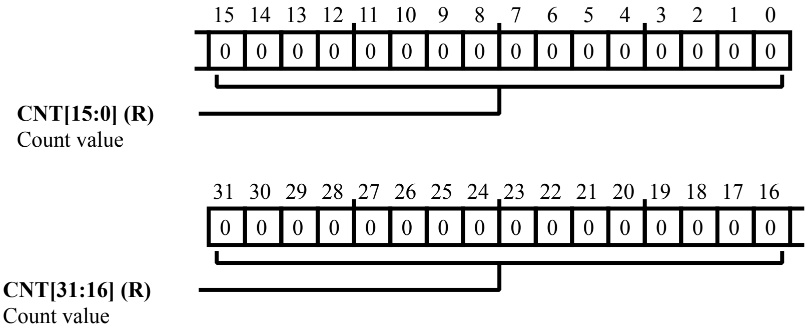

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count value

Figure 31-214: EMAC_TXOVRSIZE_G Register Diagram

Table 31-246: EMAC_TXOVRSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count value.

Tx Pause Frame Register

The EMAC_TXPAUSEFRM register contains a count of the number of good PAUSE frames transmitted.

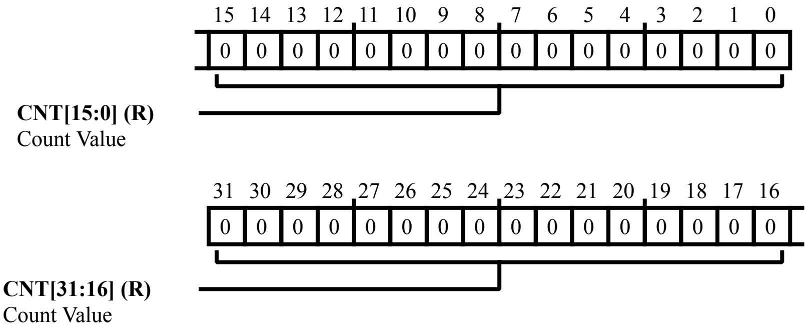

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-215: EMAC_TXPAUSEFRM Register Diagram

Table 31-247: EMAC_TXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Single Collision (Good) Register

The `EMAC_TXSNGCOL_G` register contains a count of the number of successfully transmitted frames after a single collision in Half-duplex mode.

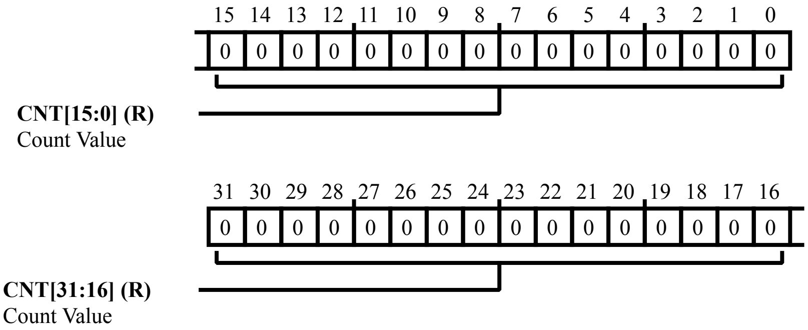

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-216: EMAC_TXSNGCOL_G Register Diagram

Table 31-248: EMAC_TXSNGCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Unicast Frames (Good/Bad) Register

The `EMAC_TXUCASTFRM_GB` register contains the count of the number of good and bad unicast frames transmitted.

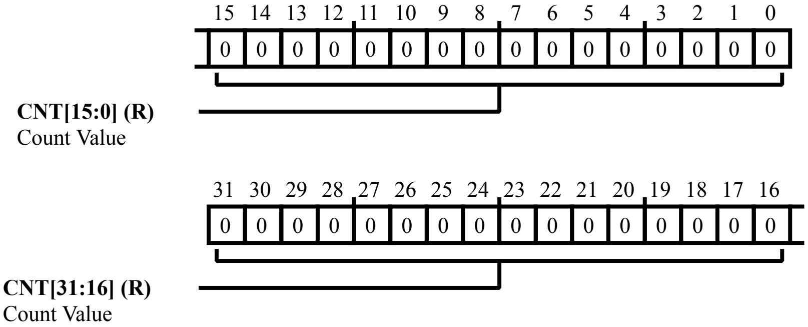

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-217: EMAC_TXUCASTFRM_GB Register Diagram

Table 31-249: EMAC_TXUCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Underflow Error Register

The EMAC_TXUNDR_ERR register contains a count of the number of frames aborted due to frame underflow error.

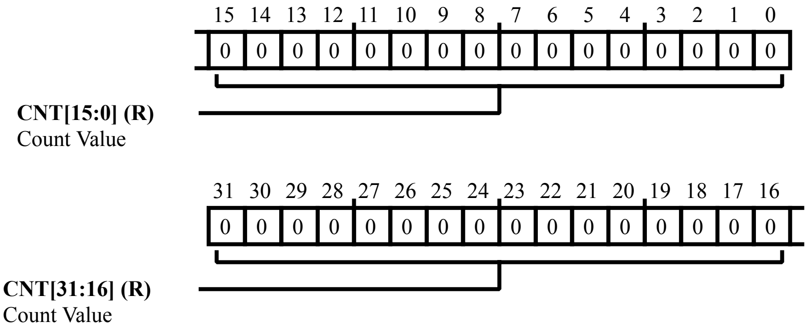

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-218: EMAC_TXUNDR_ERR Register Diagram

Table 31-250: EMAC_TXUNDR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx VLAN Frames (Good) Register

The `EMAC_TXVLANFRM_G` register contains a count of the number of good VLAN frames transmitted, exclusive of retried frames.

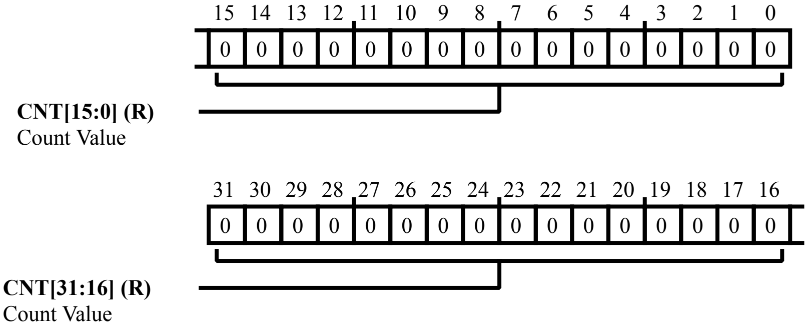

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[15:0] (R)
Count Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT[31:16] (R)
Count Value

Figure 31-219: EMAC_TXVLANFRM_G Register Diagram

Table 31-251: EMAC_TXVLANFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

VLAN Tag Register

The EMAC_VLANTAG register contains the VLAN tag.

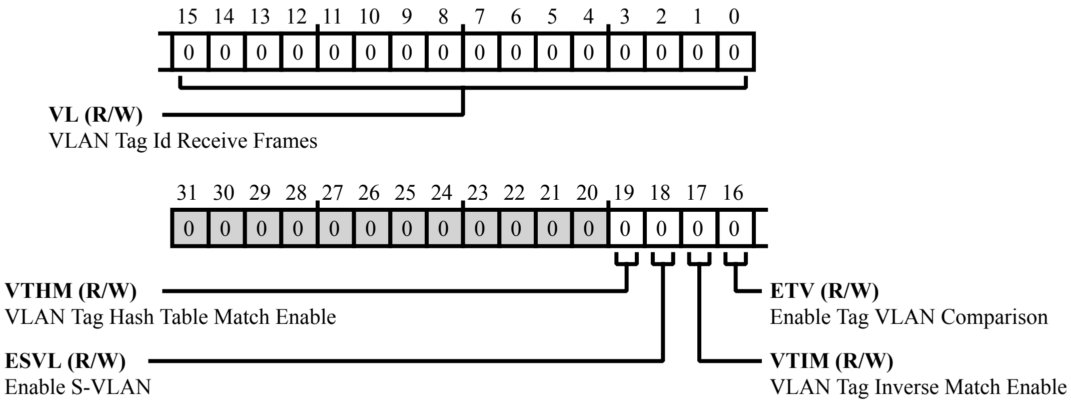

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VL (R/W)
VLAN Tag Id Receive Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VTHM (R/W)
VLAN Tag Hash Table Match Enable
ESVL (R/W)
Enable S-VLAN
ETV (R/W)
Enable Tag VLAN Comparison
VTIM (R/W)
VLAN Tag Inverse Match Enable

Figure 31-220: EMAC_VLANTAG Register Diagram

Table 31-252: EMAC_VLANTAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	VTHM	VLAN Tag Hash Table Match Enable. The EMAC_VLANTAG.VTHM bit, When set, the most significant four bits of the VLAN tags CRC are used to index the content of VLAN Hash Table Register. A value of 1 in the VLAN Hash Table register, corresponding to the index, indicates that the frame matched the VLAN hash table.
18 (R/W)	ESVL	Enable S-VLAN. The EMAC_VLANTAG.ESVL bit, When this bit is set, the MAC transmitter and receiver also consider the S-VLAN (Type = 0x88A8) frames as valid VLAN tagged frames.
17 (R/W)	VTIM	VLAN Tag Inverse Match Enable. The EMAC_VLANTAG.VTIM bit, enables the VLAN Tag inverse matching. The frames that do not have matching VLAN Tag are marked as matched.
16 (R/W)	ETV	Enable Tag VLAN Comparison. The EMAC_VLANTAG.ETV bit, when set, directs the EMAC to use a 12-bit VLAN identifier, rather than the complete 16-bit VLAN tag, for comparison and filtering. Bits[11:0] of the VLAN tag are compared with the corresponding field in the received VLAN-tagged frame. When this bit is reset, all 16 bits of the received VLAN frame's fifteenth and sixteenth bytes are used for comparison.

Table 31-252: EMAC_VLANTAG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VL	VLAN Tag Id Receive Frames. The `EMAC_VLANTAG.VL` bits contain the 802.1Q VLAN tag to identify VLAN frames, and is compared to the fifteenth and sixteenth bytes of the frames being received for VLAN frames. Bits[15:13] are the User Priority, Bit[12] is the Canonical Format Indicator (CFI) and bits[11:0] are the VLAN tag's VLAN Identifier (VID) field. When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison. If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and sixteenth bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 to be VLAN frames.

VLAN Hash Table Register

The `EMAC_VLAN_HSHTBL` register contains the VLAN hash table

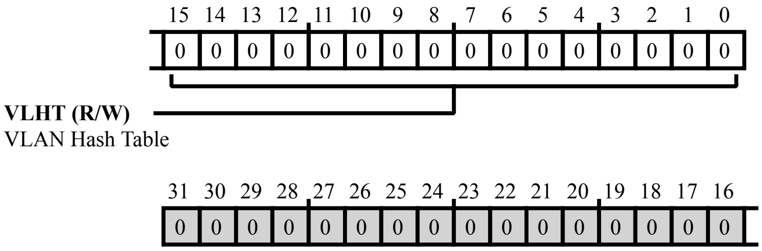

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VLHT (R/W)
VLAN Hash Table
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 31-221: EMAC_VLAN_HSHTBL Register Diagram

Table 31-253: EMAC_VLAN_HSHTBL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VLHT	VLAN Hash Table. This field contains the 16-bit VLAN Hash Table.

VLAN Tag Inclusion or Replacement Register

The `EMAC_VLAN_INCL` register contains the VLAN tag for insertion into or replacement in the transmit frames.

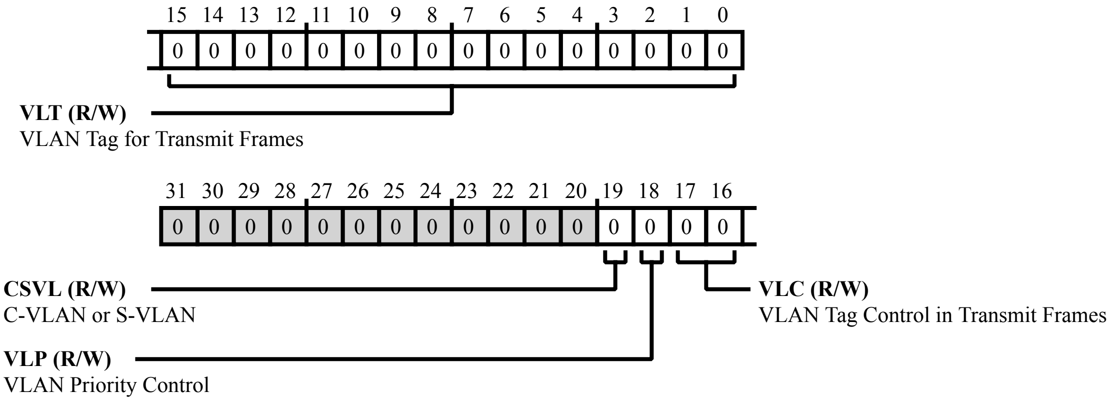

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VLT (R/W)
VLAN Tag for Transmit Frames
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CSVL (R/W)
C-VLAN or S-VLAN
VLP (R/W)
VLAN Priority Control
VLC (R/W)
VLAN Tag Control in Transmit Frames

Figure 31-222: EMAC_VLAN_INCL Register Diagram

Table 31-254: EMAC_VLAN_INCL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/W)	CSVL	C-VLAN or S-VLAN. The `EMAC_VLAN_INCL.CSVL` bit, When this bit is set, S-VLAN type (0x88A8) is inserted or replaced in the 13th and 14th bytes of transmitted frames.
18 (R/W)	VLP	VLAN Priority Control. The `EMAC_VLAN_INCL.VLP` bit, When this bit is set, the control Bits [17:16] are used for VLAN deletion, insertion, or replacement
17:16 (R/W)	VLC	VLAN Tag Control in Transmit Frames. The `EMAC_VLAN_INCL.VLC` bit, 2'b00:No VLAN tag deletion, insertion, or replacement 2'b01:VLAN tag deletion. The MAC removes the VLAN type (bytes 13 and 14) and VLAN tag (bytes 15 and 16) of all transmitted frames with VLAN tags. 2'b10:VLAN tag insertion. The MAC inserts VLT in bytes 15 and 16 of the frame after inserting the Type value (0x8100/0x88a8) in bytes 13 and 14. This operation is performed on all transmitted frames, irrespective of whether they already have a VLAN tag. 2'b11:VLAN tag replacement. The MAC replaces VLT in bytes 15 and 16 of all VLAN-type transmitted frames (Bytes 13 and 14 are 0x8100/0x88a8).
15:0 (R/W)	VLT	VLAN Tag for Transmit Frames. The `EMAC_VLAN_INCL.VLT` bit, contains the value of the VLAN tag to be inserted or replaced.

Watchdog Timeout Register

The `EMAC_WDOG_TIMOUT` register controls the watchdog timeout for received frames.

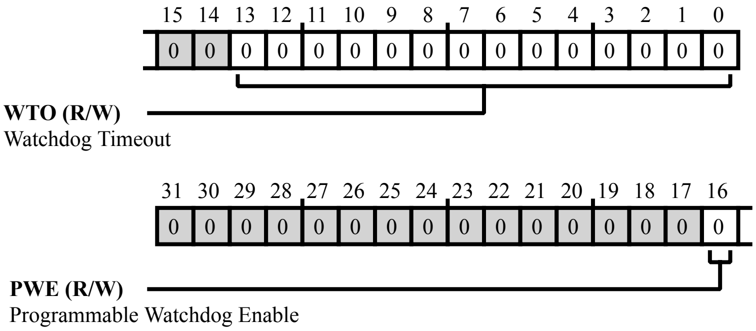

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WTO (R/W)
Watchdog Timeout
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PWE (R/W)
Programmable Watchdog Enable

Figure 31-223: EMAC_WDOG_TIMOUT Register Diagram

Table 31-255: EMAC_WDOG_TIMOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	PWE	Programmable Watchdog Enable. The `EMAC_WDOG_TIMOUT.PWE` bit, When this bit is set and Bit 23 (WD) of Register 0 (MAC Configuration Register) is reset, the WTO field (Bits[13:0]) is used as watchdog timeout for a received frame.
13:0 (R/W)	WTO	Watchdog Timeout. The `EMAC_WDOG_TIMOUT.WTO` bit, When Bit 16 (PWE) is set and Bit 23 (WD) of Register 0 (MAC Configuration Register) is reset, this field is used as watchdog timeout for a received frame. If the length of a received frame exceeds the value of this field, such frame is terminated and declared as an error frame.

32 Sinus Cardinalis (SINC) Filter

The sinus cardinalis (SINC) filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator.

The filter consists of a set of integration and decimation stages implemented directly in logic for efficient execution. The SINC filter supports capture of current or voltage feedback signals from an isolating analog-to-digital converter (ADC). Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback; a secondary filter for overcurrent detection. The SINC module includes four filter channels and two modulator clock generators.

SINC Filter Features

The SINC features include:

- Four-bit stream filter channels for current or voltage feedback signal processing
- Each channel includes two SINC filter pairs:
  - A primary filter for feedback signal processing
  - A secondary filter for overload detection
- Up to two modulator clock sources with phase control options
- Configuration of SINC filter channels according to a modulator clock selection
- Programmable order and decimation rates
- Primary filters:
  - Programmable bias and gain with output saturation
  - Dedicated direct memory access (DMA) channels with data interleaving and programmable data ready output triggers
- Secondary filters:
  - Detecting a fault when signals exceed amplitude and duration values

  - Registers preserving the eight most recent samples before a fault event
- Multiple interrupt trigger sources for overload fault and data overflow events

SINC Functional Description

The SINC filter has the following functionality:

Digital filter

The filter removes the modulator sample clock and recovers a digital value of the sampled signal.

DC gain and data resolution

The DC gain of the digital filter is a function of the order and decimation rate.

Frequency response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency.

Output scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function.

ADSP-SC58x SINC Register List

The SINC filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator. Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback, and a secondary filter for overcurrent detection. A set of registers governs SINC operations. For more information on SINC functionality, see the SINC register descriptions.

Table 32-1: ADSP-SC58x SINC Register List

Name	Description
`SINC_BIAS0`	Bias for Group 0 Register
`SINC_BIAS1`	Bias for Group 1 Register
`SINC_CLK`	Clock Control Register
`SINC_CTL`	Control Register
`SINC_HIS_STAT`	History Status Register
`SINC_LEVEL0`	Level Control for Group 0 Register
`SINC_LEVEL1`	Level Control for Group 1 Register

Table 32-1: ADSP-SC58x SINC Register List (Continued)

Name	Description
SINC_LIMIT0	(Amplitude) Limits for Secondary Filter 0 Register
SINC_LIMIT1	(Amplitude) Limits for Secondary Filter 1 Register
SINC_LIMIT2	(Amplitude) Limits for Secondary Filter 2 Register
SINC_LIMIT3	(Amplitude) Limits for Secondary Filter 3 Register
SINC_P0SEC_HIST[n]	Pair 0 Secondary (Filter) History n Register
SINC_P1SEC_HIST[n]	Pair 1 Secondary (Filter) History n Register
SINC_P2SEC_HIST[n]	Pair 2 Secondary (Filter) History n Register
SINC_P3SEC_HIST[n]	Pair 3 Secondary (Filter) History n Register
SINC_PHEAD0	Primary (Filters) Head for Group 0 Register
SINC_PHEAD1	Primary (Filters) Head for Group 1 Register
SINC_PPTR0	Primary (Filters) Pointer for Group 0 Register
SINC_PPTR1	Primary (Filters) Pointer for Group 1 Register
SINC_PTAIL0	Primary (Filters) Tail for Group 0 Register
SINC_PTAIL1	Primary (Filters) Tail for Group 1 Register
SINC_RATE0	Rate Control for Group 0 Register
SINC_RATE1	Rate Control for Group 1 Register
SINC_STAT	Status Register

ADSP-SC58x SINC Interrupt List

Table 32-2: ADSP-SC58x SINC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
144	SINC0_STAT	SINC0 Status	Level	

ADSP-SC58x SINC Trigger List

Table 32-3: ADSP-SC58x SINC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
49	SINC0_P0_OVLD	SINC0 Pair 0 Overload Indicator	Edge
50	SINC0_P1_OVLD	SINC0 Pair 1 Overload Indicator	Edge
51	SINC0_P2_OVLD	SINC0 Pair 2 Overload Indicator	Edge
52	SINC0_P3_OVLD	SINC0 Pair 3 Overload Indicator	Edge

Table 32-3: ADSP-SC58x SINC Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
53	`SINC0_DATA0`	SINC0 Data Move 0	Edge
54	`SINC0_DATA1`	SINC0 Data Move 1	Edge

Table 32-4: ADSP-SC58x SINC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
42	`SINC0_SYNC0`	SINC0 Synchronization Input 0	Pulse
43	`SINC0_SYNC1`	SINC0 Synchronization Input 1	Pulse

SINC Definitions

To make the best use of the SINC, it is useful to understand the following terms.

Decimation

Decimation is the process of discarding samples from a data stream.

Decimation Rate

The decimation rate is the ratio of the filter input data rate to the filter output data rate.

Filter Order

The SINC filter order is the number of integration and decimation stages in the filter.

Modulator Order

The modulator order is the number of comparator and integrator stages in a sigma-delta modulator.

Sigma-Delta Modulator

The sigma-delta modulator is an oversampling analog-to-digital conversion circuit that generates a digital bit stream whose pulse density is proportional to the analog voltage presented to the input.

SINC Block Diagram

The *SINC Block Diagram* figure shows the functional blocks within the SINC module.

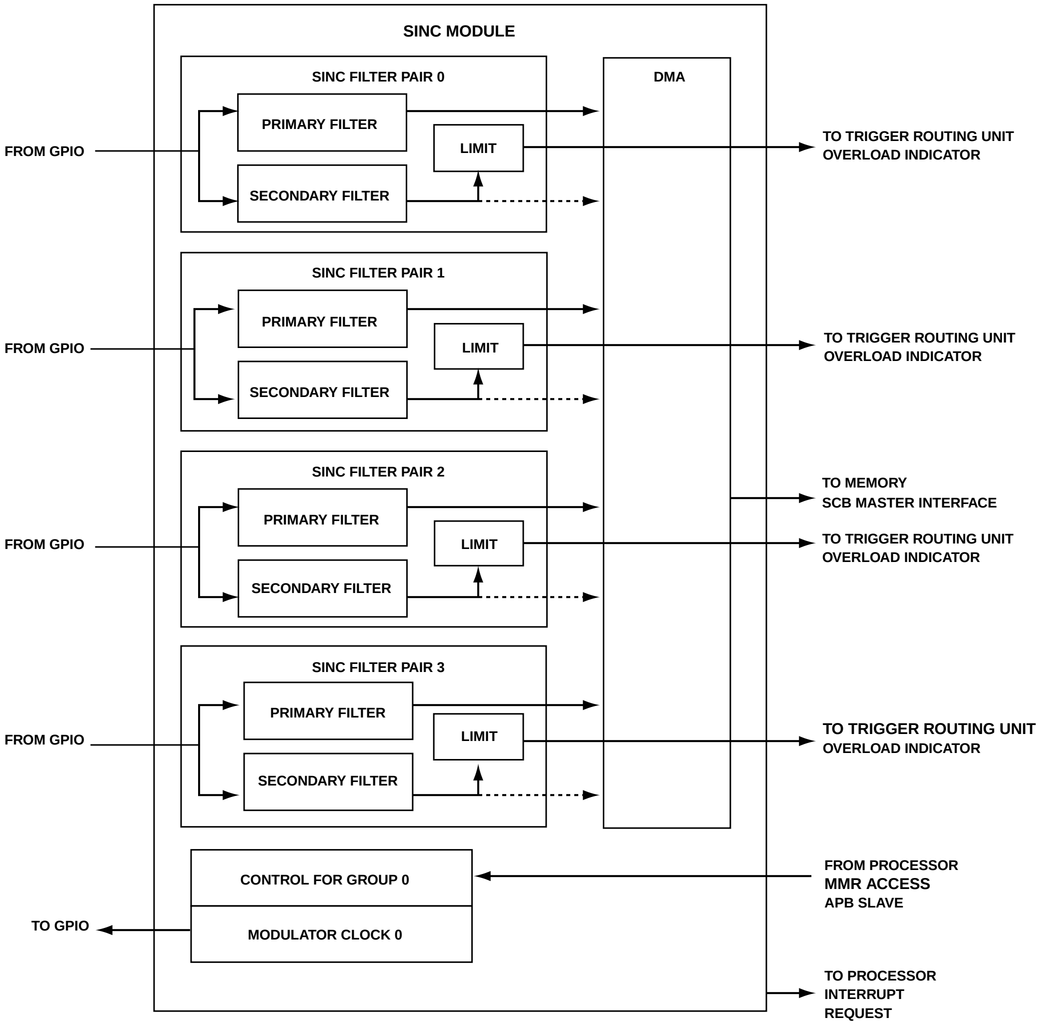

SINC MODULE
SINC FILTER PAIR 0
PRIMARY FILTER
LIMIT
SECONDARY FILTER
SINC FILTER PAIR 1
PRIMARY FILTER
LIMIT
SECONDARY FILTER
SINC FILTER PAIR 2
PRIMARY FILTER
LIMIT
SECONDARY FILTER
SINC FILTER PAIR 3
PRIMARY FILTER
LIMIT
SECONDARY FILTER
DMA
FROM GPIO
FROM GPIO
FROM GPIO
FROM GPIO
TO TRIGGER ROUTING UNIT OVERLOAD INDICATOR
TO TRIGGER ROUTING UNIT OVERLOAD INDICATOR
TO MEMORY SCB MASTER INTERFACE
TO TRIGGER ROUTING UNIT OVERLOAD INDICATOR
TO TRIGGER ROUTING UNIT OVERLOAD INDICATOR
CONTROL FOR GROUP 0
MODULATOR CLOCK 0
TO GPIO
FROM PROCESSOR MMR ACCESS APB SLAVE
TO PROCESSOR INTERRUPT REQUEST

Figure 32-1: SINC Block Diagram

The block diagram shows four SINC filter pairs (SINC0–3), one modulator clock source, and one bank of control registers (units). The module accepts four sigma-delta bit streams from the GPIO input pins and directs the modulator clock source of GROUP 0 to the GPIO output pin. A pulse-width modulation (PWM) signal synchronizes the modulator clock to optimize system performance. Each SINC filter pair includes the primary filter, secondary filter, DMA interface, and overload limit detection functions.

The primary SINC filter transmits its data to memory using DMA. The secondary SINC filter generates overload signals, which can be routed through the trigger routing unit (TRU) to trip a PWM modulator and generate an interrupt request.

The SINC filter pairs are assigned to control unit 0, where multiple channels of current or voltage-feedback share common filter parameters. The primary filters generate high-resolution signals for closing the feedback control loop. The secondary filters are for rapid-overload fault detection, require lower resolution, but a faster response. The primary and secondary filters have programmable order and decimation rates. The primary filters also have the programmable output gain stage, while the secondary filters have the programmable overload limit thresholds.

To use the primary and secondary filters, set up the filter parameters once, prior to using the filters. The feedback control algorithm reads the data from the primary filter directly from memory. A PWM interrupt request signal can generate the algorithm timing signal, or the SINC module generates a data trigger. The data history of the secondary filter is saved in buffer registers once an overload fault signal is detected. The data history supports fault diagnostics.

SINC Architectural Concepts

The architecture of the SINC includes the following:

- Digital Filter
- DC Gain and Data Resolution
- Frequency Response
- Output Scaling

Digital Filter

The SINC filter has a transfer function that lends itself to an implementation in digital logic, using a series of summation and decimation functions. The filter removes the modulator sample clock and recovers a digital value of the sampled signal. The filter design matches a bipolar SD modulator. The design produces a 50% pulse density for a 0V input, over 50% for positive inputs and less than 50% for negative inputs.

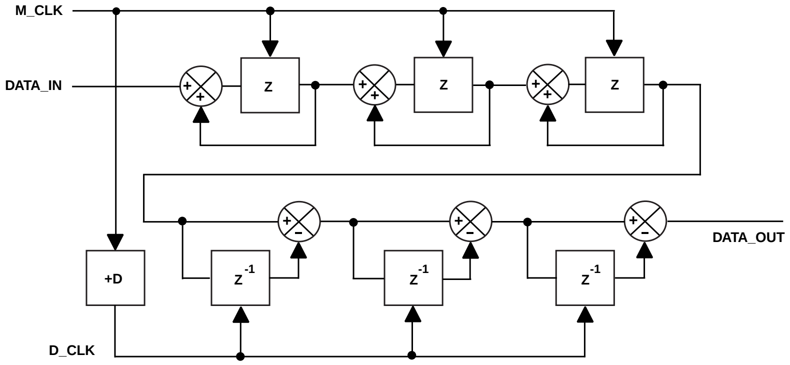


Figure 32-2: SINC Digital Filter

The digital filter is a set of accumulators driven by the modulator clock (`M_CLK`), followed by a set of differentiators driven by the decimation clock (`D_CLK`). The input accumulators convert the input bit stream into a multibyte word, while the output differentiators derive the average 1's density of the bit stream. The number of accumulator and differentiator stages can be three or four, depending on the order of the filter. The DC gain and bandwidth of the filter are functions of the filter order (`O`) and the decimation rate (`D`), which is the ratio of the modulator to the decimation clock.

The calculation of the transfer function of the SINC filter includes the product of the transfer functions for the accumulators and differentiators, and in the `z` domain. The following equation gives the calculation:

$$H(z) = \left[\frac{1}{D} \text{ x } \frac{1 - z^{-D}}{1 - z^{-1}}\right]^{O}$$

DC Gain and Data Resolution

The DC gain of the digital filter is a function of the order and decimation rate. At 100% ones density input, each accumulator stage counts `D` pulses, and the gain of the filter is given as follows:

$$G_{dc} = D^{o}$$

The higher the decimation rate, the higher the resolution of the output data. The number of usable data bits is a function of the `SNR`; the ***Filter Order versus Decimation*** table shows `ENOB` versus the decimation rate.

Table 32-5: ENOB versus Decimation

Decimation		4	5	6	7	8	16	32	64	128	256
O = 3	SNR (dB)	6.42	11.47	16.41	20.57	23.55	35.02	48.59	62.26	76.46	89.59
	ENOB	0.8	1.6	2.4	3.1	3.6	5.5	7.8	10.0	12.4	14.6
O = 4	SNR (dB)	9.08	14.77	19.78	23.41	25.9	38.05	51.29	64.67	79.15	
	ENOB	1.2	2.1	3.1	3.6	4.0	6.0	8.2	10.4	12.8	

Notes: ENOB versus order and decimation rate.

Test conditions are for a 1.22 kHz tone and a 10 MHz modulator.

Frequency Response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency, f_M. The equation is obtained by substituting $e^{j \text{ x } Ts}$ for `z` in the transfer function, where `Ts` is the period of the modulator clock:

$$H\left(e^{j\frac{\omega}{f_M}}\right) = \left[\frac{1}{D} \times \frac{\sin\left(D\frac{\omega}{2f_M}\right)}{\sin\left(\frac{\omega}{2f_M}\right)} \times e^{-j(D-1)\frac{\omega}{2f_M}}\right]^{O}$$

The filter has a linear phase response with a constant group delay given by:

$$\tau_d = \left(\frac{D-1}{2}\right)\frac{O}{f_M}$$

The ***Frequency Response*** plots show zeros at multiples the decimation frequency, where the `sin` term in the numerator goes to zero. This response makes it possible to remove some PWM ripple components from the motor current waveform by matching the decimation frequency to the PWM switching frequency. There are some limitations at lower PWM frequencies based on available decimation rates. High decimation rates limit the bandwidth of the control loop because of the phase delay, which is 3π radians at the decimation frequency.

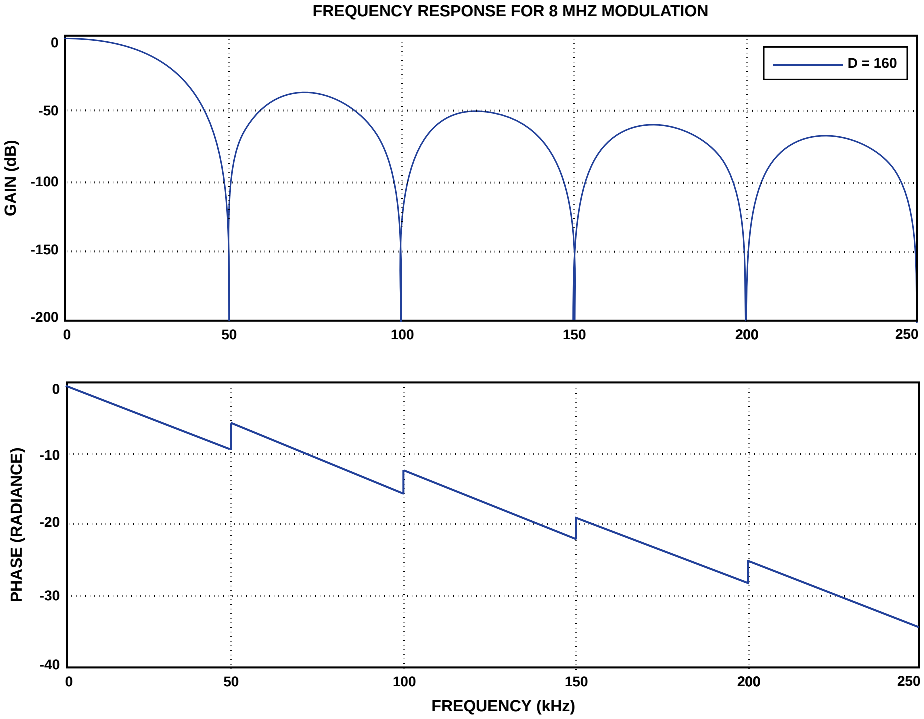


Figure 32-5: Frequency Response

Output Scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function. The primary filter used for feedback signal processing includes the output bias and scaling blocks to present a 16-bit signed integer to the control code. The scaling is required at decimation rates higher than 32 to keep the lower 16 bits of the output word.

The secondary filter supports overload detection functions. The secondary filter can detect signals crossing maximum and minimum thresholds. It has a glitch filter that only accepts faults with a minimum number of counts (`c`) within a certain count window (`w`). The secondary filter has no output scaling, so the minimum and maximum values in the overload registers must be calculated from the DC gain of the secondary filter. The response time to a step input is approximately 2 x O decimation clock cycles.

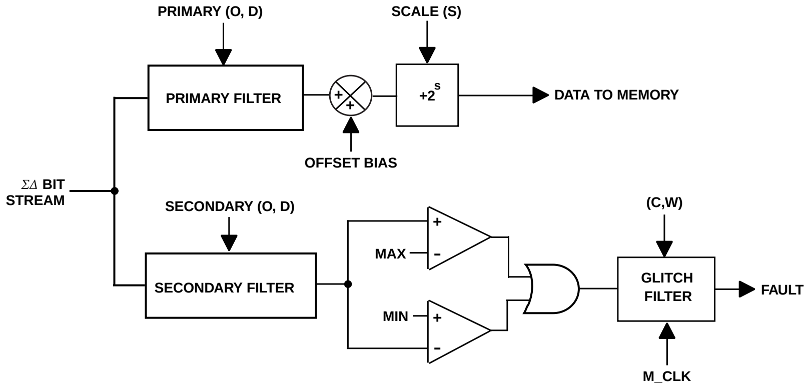


Figure 32-4: Output Scaling

SINC Operating Modes

The SINC filter module has only one operating mode. The module generates the clock source for a sigma-delta modulator analog front end and filters the output data stream for the modulator. The primary SINC filter transfers its data to memory through DMA. The secondary SINC filter output generates an overload trigger signal that the SINC filter module can use as a PWM trip signal. The SINC control registers enable the module and set up the modulator clock sources, filter parameters, DMA transfers, and interrupts masks, as described in SINC Programming Model.

For self-testing purposes, the SINC data inputs can be driven by bit streams transferred from the SPORT0A port through DMA from a slave memory space (typically M4 main memory). The following programming selections are required to enable SINC self-test mode:

- [3:0] – set each bit to 1 for each SINC channel to be selected for test. If 1, the SINC channel input will be driven by the SPORT, not the associated GPIO.
- Enable filters to route SPORT data to SINC unit with the write: `*pREG_TESYS1_SINC_TEST = 0xf`
- The SINC unit must be configured as follows.
  - Timer Group 0; MCLK0
- The SPORT (A) must be configured as follows.
  - External Clock
  - 16 bit data recommended
  - No multichannel mode
  - Enable SPORT0_A in transmit mode, external clock, internal FS, and I2S Mode `*pREG_SPORT0_CTL_A = 0x021548f1`
- Any data transmitted on SPORT0 Data A channel is sent to SINC block.

SINC Data Transfer Modes

The only mode of data transfer between the primary SINC filter and memory is through DMA (See Primary DMA Configuration and Data Interrupts for more information. Reading the history registers for the secondary filter is the only way to transfer data between the secondary SINC filter and memory. See Overload Detection for more information.

SINC Signal Modes

The SINC filter has an interrupt request signal and a number of triggers and status signals to indicate system events and errors.

- Primary data transfer trigger:

  The SINC filter can generate a trigger after a user-specified number of primary output sets are transferred to memory. There is one trigger source for each filter group. See Primary DMA Configuration and Data Interrupts for more information.

- Secondary data overload trigger:

  The SINC filter can generate a trigger when one of the secondary filters detects an overload condition. There is one trigger source for each secondary filter. See Overload Detection for more information.

- SINC status bits:

  The SINC status bits indicate secondary filter overload errors, primary filter saturation errors, primary filter transfer count exceeded, and primary filter data buffer errors.

  - Secondary filter overload errors:

    A number of status bits indicate the type of error and the filter channel when a secondary filter detects an overload condition. The status bits `SINC_STAT.GLIM0` and `SINC_STAT.GLIM1` indicate the control group of the secondary filter that detected the overload. The status bits `SINC_STAT.MAX0` through `SINC_STAT.MAX3` indicate when the error a maximum limit on one of the secondary filter channels is passed, causing the error. The status bits `SINC_STAT.MIN0` through `SINC_STAT.MIN3` indicate when a minimum limit on one of the secondary filter channels is passed, causing the error.

  - Primary filter data saturation errors:

    A number of status bits indicate the group and filter channel when the SINC filter detects data saturation . The status bits `SINC_STAT.GSAT0` and `SINC_STAT.GSAT1` indicate the filter control group when the SINC filter detects data saturation. The status bits `SINC_STAT.PSAT0` through `SINC_STAT.PSAT3` indicate a primary filter channel that detects data saturation.

  - Primary filter transfer count exceeded:

    The status bits `SINC_STAT.PCNT0` and `SINC_STAT.PCNT1` are set every time a specified number of primary filter data sets for that filter group are transferred to memory. The primary filter data set for a group is the data for all the channels in the group. The specified number of data sets is the value in the

`SINC_LEVEL0.PCNT`-`SINC_LEVEL1.PCNT` bits. Write 1 to clear the bits before the next data transfer to generate a trigger.

- Primary filter data buffer errors:

  A number of status bits indicate data buffer errors. The status bits `SINC_STAT.FOVF0` and `SINC_STAT.FOVF1` indicate the filter control group when there is a SINC data buffer overflow. This error occurs when a third sample is presented to the buffer before the first sample transfers to memory. The status bits `SINC_STAT.PFAB0` and `SINC_STAT.PFAB1` indicate the filter group when an error occurs while writing the data to memory.

- SINC status interrupt request:

  There is a single SINC filter interrupt request output that can indicate secondary filter overload errors, primary filter data saturation, or primary filter data buffer overrun. There is one interrupt mask bit for each of these conditions per filter group. See Interrupt Masking for more information.

SINC Event Control

The SINC provides status and error bits through different registers to signal the core about its state and various error conditions that occur during its operation. These conditions include:

- Interrupt status related to data overload, data saturation, data FIFO fault conditions
- Error status related to SINC operations
- History status (which do not generate interrupts) related to data FIFO operations

SINC Interrupt Signals

The interrupt request and trigger signals to the SINC filter module include:

- One interrupt request signal, SINC_STAT, triggered by fault events, such as detected overload limits and data transfer errors. Manage interrupt request generation with the masking bits in the `SINC_CTL` register:

  - Bits `SINC_CTL.ELIM0`-`SINC_CTL.ELIM1` can enable (unmask) interrupt request generation on overload faults when the `SINC_STAT.GLIM0`-`SINC_STAT.GLIM1` bit is set, respectively.
  - Bits `SINC_CTL.ESAT0`-`SINC_CTL.ESAT1` can mask interrupt request generation on data saturation faults when the `SINC_STAT.GSAT0`-`SINC_STAT.GSAT1` bit is set, respectively.
  - Bits `SINC_CTL.EFOVF0`-`SINC_CTL.EFOVF1` can mask interrupt request generation on data buffer overruns when the `SINC_STAT.FOVF0`-`SINC_STAT.FOVF1` bit is set, respectively.

  The fault bits in the `SINC_STAT` register must be cleared to clear the interrupt request.

- Two data count triggers, one trigger per each control group. The SINC filter module regularly uses the data count triggers to generate a software interrupt request or trigger an event. First, set the `SINC_CTL.EPCNT0` or `SINC_CTL.EPCNT1` masking bit to enable the data count trigger. Then, the TRU must assign the data count master (`SINC0_DATA0-1`) to an interrupt request input.

- Four overload triggers, one trigger per each channel. The SINC filter module can use overload triggers to trip the appropriate PWM block in the case of a fault. The overload trigger is always enabled, and the TRU must assign the masters (`SINC0_P0_OVLD` through `SINC0_P4_OVLD`) to the appropriate PWM trip input slave (`PWMn_TRIP_TRIGn`).

SINC Status and Error Signals

The status and error signals related to SINC operations are as follows:

- `SINC_STAT` signals:
  - The amplitude and duration limit error signals for secondary SINC filters: `SINC_STAT.MAX0` through `SINC_STAT.MAX3`, `SINC_STAT.MIN0` through `SINC_STAT.MIN3`, and `SINC_STAT.GLIM0-SINC_STAT.GLIM1`.
  - The output saturation error signals for primary SINC filters: `SINC_STAT.MAX0` through `SINC_STAT.MAX3`, `SINC_STAT.MIN0` through `SINC_STAT.MIN3`, and `SINC_STAT.GLIM0-SINC_STAT.GLIM1`.
  - The output FIFO overflow error signals for primary SINC filters: `SINC_STAT.FOVF0` and `SINC_STAT.FOVF1`.
  - The output count error signals for primary SINC filters: `SINC_STAT.PCNT0` and `SINC_STAT.PCNT1`.
  - The SCB fabric-related error signals for primary SINC filters: `SINC_STAT.PFAB0-SINC_STAT.PFAB1`.
- `SINC_CLK` signals:
  - The phase shift signals for SINC modulator clocks: `SINC_CLK.MREQ0-SINC_CLK.MREQ1`.
- `SINC_HIS_STAT` signals:
  - The history saved signals for secondary SINC filters: `SINC_HIS_STAT.P0HISPTR` through `SINC_HIS_STAT.P3HISPTR`, which indicate that the data history of the filter is saved in buffer registers due to a detected overload error signal.

SINC Programming Model

The pin multiplexer enables the device input and output pins and connects the signals to the SINC module. Decide the filter grouping in advance. The filter parameters are defined according to the control register group.

Follow these steps to configure the filters:

1. Define the primary and secondary filter parameters by setting the appropriate bits in the control register for each filter channel group.

2. Set the upper and lower overload limits to maximum for each channel to avoid overload trips due to the filter startup transient.
3. Define the modulator clock frequency and startup mode.
4. Enable the SINC channels and assign them to the selected group of control registers.

Set the running overload limits after the filter settles, which is (order × decimation) modulator clock cycles after startup. When the filters are running, the module transfers its data to data RAM on the dedicated DMA channels. Once configured, the control registers do not need accessing, but the status and some data buffer registers typically are read after fault events.

In general, adjusting filter parameters during operation leads to unpredictable results. However, programs can write to the trigger and interrupt mask registers, as well as to the secondary threshold level registers during operation.

The DC gain of the converter subsystem depends on the gain of the input modulator (G_M), filter order (`O`), and decimation rate (`D`). The primary filter has an output binary scalar (`S`) to fit data into a 16-bit range:

$$G_M = 0.625 \times (D^O \div 2^S)$$

SINC Programming Concepts

Using the features and event control for the SINC to their greatest potential requires an understanding of some SINC-related concepts:

- Channel Configuration
- Trigger Masking
- Interrupt Masking
- Modulator Clock
- Filter Configuration
- Primary Filter Parameters
- Primary DMA Configuration and Data Interrupts
- Secondary Filter Parameters
- Overload Detection

Channel Configuration

The control bits, `SINC_CTL.EN0` through `SINC_CTL.EN3`, configure SINC module channels. These control bits enable or disable the selected SINC filter channel and assign the channel to control register group 0.

Trigger Masking

The SINC module has one data count trigger. The module can use the data count trigger to generate a software interrupt regularly or trigger an event. First, set the `SINC_CTL.EPCNT0` masking bit to enable the data count trigger. Then, the TRU must assign the data count master (`SINC_DATn`) to an interrupt input.

There are also four overload triggers, one trigger per each channel. The SINC module can use overload triggers to trip the appropriate PWM block when there is a fault. The overload trigger is always enabled, and the TRU must assign the masters (`SINC0_Pn_OVLD`) to the appropriate PWM trip input slave (`PWMn_TRIP_TRIGn`).

Interrupt Masking

The SINC filter can generate a `SINC_STAT` interrupt signal when triggered by fault events, such as detected overload limits or data transfer errors.

Enable (unmask) interrupt generation with the `SINC_CTL` register bits:

- The `SINC_CTL.ELIM0` bit can enable interrupt generation on overload faults when the `SINC_STAT.GLIM0` bit is set.
- The `SINC_CTL.ESAT0` bit can enable interrupt generation on data saturation faults when the `SINC_STAT.GSAT0` bit is set.
- The `SINC_CTL.EFOVF0` bit can enable interrupt generation on data buffer overruns when the `SINC_STAT.FOVF0` bit is set.

The fault bits in the `SINC_STAT` register must be cleared to clear the interrupt.

Modulator Clock

The SINC filter has one modulator clock source. The clock source can be set with an output frequency in the range of 1–20 MHz. The SINC module uses bits in the `SINC_CLK` register to control the modulator clock output, frequency, and phase. The SINC module uses the `SINC_CLK.MCEN0` bit field to enable the modulator clock and control the startup behavior of the clock. Start the clock immediately or enable the clock on the first rising edge of an external trigger connected to the `SINC0_SYNCn` input of the module. This action synchronizes the modulator clock with a PWM waveform source by routing a `PWMn_SYNC` master to the `SINC0_SYNC0` slave using the TRU.

The target frequency is in the range and derived from `SCLK0_0` using an integer divisor in the `SINC_CLK.MDIV0` bits. Write to the `SINC_CLK.MREQ0` bit to adjust the phase of the clock. This adjustment lengthens the next clock period by the number of `SCLK0_0` periods stored in the respective `SINC_CLK.MADJ0` bit field. The `SINC_CLK.MREQ0` bit is cleared automatically once the adjustment is complete.

Filter Configuration

Configure the primary and secondary filter parameters by setting the appropriate bits in the `SINC_RATE0`, `SINC_LEVEL0`, and `SINC_BIAS0` control registers. Configure the DMA transfers by setting the appropriate bits in the `SINC_PHEAD0` and `SINC_PTAIL0` registers. Set the maximum and minimum levels for overload detection in the four limit registers, `SINC_LIMIT0` – `SINC_LIMIT3`. Set the overload filtering parameters in the `SINC_LEVEL0` register.

Primary Filter Parameters

Set the primary filter to the 3rd or 4th order by the `SINC_LEVEL0.PORD` bit assigned to the channel. Set the decimation rate for the primary filter using the `SINC_RATE0.PDEC` bits assigned to the channel. Valid decimation rates are in the range 4–256. Set the phase of the primary filter output relative to the number of modulator clocks after enabling the filter using the `SINC_RATE0.PADJ` bits assigned to the channel. Valid `SINC_RATE0.PADJ` values are in the range 0 to `SINC_RATE0.PDEC` – 1.

The raw filter output is a 32-bit wide integer, has an offset added, and is scaled to a 16-bit number before transfer to memory. Store the 32-bit two's compliment offset value in the `SINC_BIAS0` register of the channel. Set the binary scale factor by a mantissa in the range 4–32 stored in the `SINC_LEVEL0.PSCALE` bits. The output is a valid 16-bit signed number. If the number is outside of the valid range, the output is saturated to 0x8000 or 0x7FFF, while the `SINC_STAT.PSAT0` or `SINC_STAT.GSAT0` fault bit (according to the channel group) is set.

Primary DMA Configuration and Data Interrupts

Transfer the primary SINC filter outputs to a circular buffer in data memory using DMA. The output from the primary filter is interleaved with outputs from other primary filters. The interleaving order is from the lowest to the highest numbered filter.

The SINC module stores the circular buffer head address in the `SINC_PHEAD0` register of the channel. It stores the tail address in the `SINC_PTAIL0` register of the channel. The data address wraps around to the head address after the tail address is reached. The head and tail addresses must be 16-bit aligned and can be set to the same address. The `SINC_PPTR0` register of the channel is a read-only register that contains the address of the most recent primary SINC filter data. If there is an overflow condition in the SINC filter output data FIFO due to a delay DMA transfer, the `SINC_STAT.FOVF0` fault bit (according to the channel group) is set.

A SINC data trigger can be generated after a user-specified number of primary filter outputs (data transfers) completes. Specify the data count value by the `SINC_LEVEL0.PCNT` bits assigned to the channel, and the trigger is generated every `SINC_LEVEL0.PCNT` + 1 data transfers.

The ***SINC Data Buffer Organization*** figure shows the SINC data buffer organization. In the figure, `SINC_OUT_0_M[n]` is the data for the n th most recent sample in the m th channel in the filter group 0, and n = 0 is the most recent data.

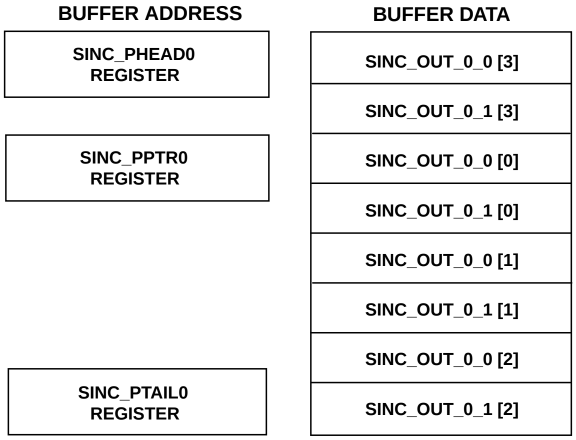


Figure 32-5: SINC Data Buffer Organization

Secondary Filter Parameters

Set the secondary filter to the 3rd or 4th order by the `SINC_LEVEL0.SORD` bit assigned to the channel. Set the decimation rate for the secondary filter using the `SINC_RATE0.SDEC` bits assigned to the channel. The secondary filter outputs are limited to 16-bit values. Limit the decimation rate according to the filter order:

- Valid decimation rates are in the range 4–40 for the 3rd order filters
- Valid decimation rates are in the range of 4–16 for the 4th order filters

Set the phase of the primary filter output relative to the number of modulator clocks after using the `SINC_RATE0.PADJ` and `SINC_RATE0.SDEC` bits to enable the filter. Valid `SINC_RATE0.PADJ` values are in the range 0 to – 1.

Overload Detection

The function of the secondary SINC filter is to detect AC current overload conditions and set up the upper and lower limit detection thresholds. There are event count filters on the overload detector outputs to reject short-term transients, if desired. Define the overload thresholds in four 32-bit registers `SINC_LIMIT0`, according to the channel number. Each register contains the 16-bit `SINC_LIMIT0.LMAX` and `SINC_LIMIT0.LMIN` overload threshold values. The SINC filter module detects an overload condition when the secondary filter output exceeds the threshold for a minimum number of counts (`SINC_LEVEL0.LCNT`) within the detection window (`SINC_LEVEL0.LWIN`). When the SINC filter module detects an overload condition, the appropriate `SINC0_Px_OVLD` trigger is generated, and the `SINC_STAT.GLIM0` fault bit is set.

The SINC filter module saves the eight most recent data samples for the secondary filter in a local circular buffer to support diagnostics after a fault is triggered. Since 16-bit data is saved, only four buffer registers are required per channel. For example, the `SINC_P1SEC_HIST[n]`[0–3] registers store the eight most recent 16-bit secondary filter outputs from channel 1. The `SINC_HIS_STAT` register contains four pointers (`SINC_HIS_STAT.P0HISPTR` through `SINC_HIS_STAT.P3HISPTR`) to the buffer location of the most recent secondary current samples, per channel.

ADSP-SC58x SINC Register Descriptions

SINC (SINC) contains the following registers.

Table 32-6: ADSP-SC58x SINC Register List

Name	Description
`SINC_BIAS0`	Bias for Group 0 Register
`SINC_BIAS1`	Bias for Group 1 Register
`SINC_CLK`	Clock Control Register
`SINC_CTL`	Control Register
`SINC_HIS_STAT`	History Status Register
`SINC_LEVEL0`	Level Control for Group 0 Register
`SINC_LEVEL1`	Level Control for Group 1 Register
`SINC_LIMIT0`	(Amplitude) Limits for Secondary Filter 0 Register
`SINC_LIMIT1`	(Amplitude) Limits for Secondary Filter 1 Register
`SINC_LIMIT2`	(Amplitude) Limits for Secondary Filter 2 Register
`SINC_LIMIT3`	(Amplitude) Limits for Secondary Filter 3 Register
`SINC_P0SEC_HIST[n]`	Pair 0 Secondary (Filter) History n Register
`SINC_P1SEC_HIST[n]`	Pair 1 Secondary (Filter) History n Register
`SINC_P2SEC_HIST[n]`	Pair 2 Secondary (Filter) History n Register
`SINC_P3SEC_HIST[n]`	Pair 3 Secondary (Filter) History n Register
`SINC_PHEAD0`	Primary (Filters) Head for Group 0 Register
`SINC_PHEAD1`	Primary (Filters) Head for Group 1 Register
`SINC_PPTR0`	Primary (Filters) Pointer for Group 0 Register
`SINC_PPTR1`	Primary (Filters) Pointer for Group 1 Register
`SINC_PTAIL0`	Primary (Filters) Tail for Group 0 Register
`SINC_PTAIL1`	Primary (Filters) Tail for Group 1 Register
`SINC_RATE0`	Rate Control for Group 0 Register
`SINC_RATE1`	Rate Control for Group 1 Register
`SINC_STAT`	Status Register

Bias for Group 0 Register

The SINC_BIAS0 register controls an output bias added to primary SINC filters of group 0.

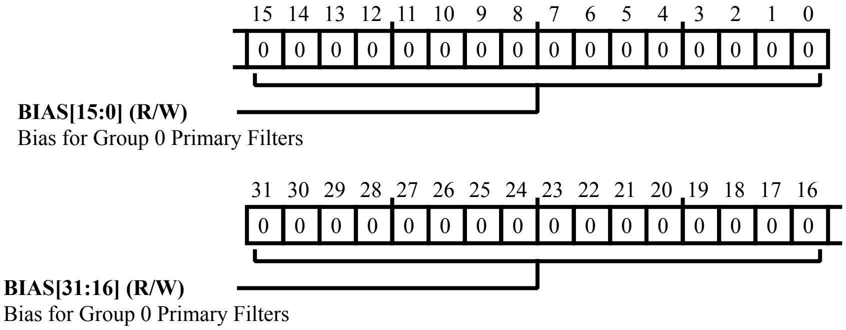

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BIAS[15:0] (R/W)
Bias for Group 0 Primary Filters
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BIAS[31:16] (R/W)
Bias for Group 0 Primary Filters

Figure 32-6: SINC_BIAS0 Register Diagram

Table 32-7: SINC_BIAS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group 0 Primary Filters. The `SINC_BIAS0.BIAS` bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where d = `SINC_RATE0.PDEC` and o = `SINC_LEVEL0.PORD`.

Bias for Group 1 Register

The `SINC_BIAS1` register controls an output bias added to primary SINC filters of group 1.

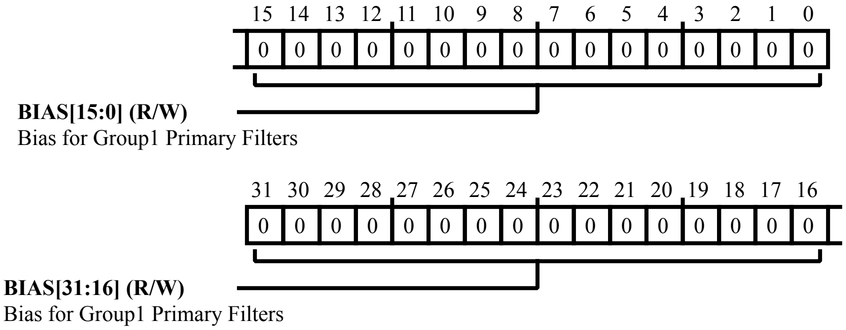

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BIAS[15:0] (R/W)
Bias for Group1 Primary Filters
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BIAS[31:16] (R/W)
Bias for Group1 Primary Filters

Figure 32-7: SINC_BIAS1 Register Diagram

Table 32-8: SINC_BIAS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group1 Primary Filters. The `SINC_BIAS1.BIAS` bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where where d = `SINC_RATE1.PDEC` and o = `SINC_LEVEL1.PORD`.

Clock Control Register

The SINC_CLK register generates and enables two SINC modulator clocks. The register also controls each clock's output, frequency, phase, and start-up behavior.

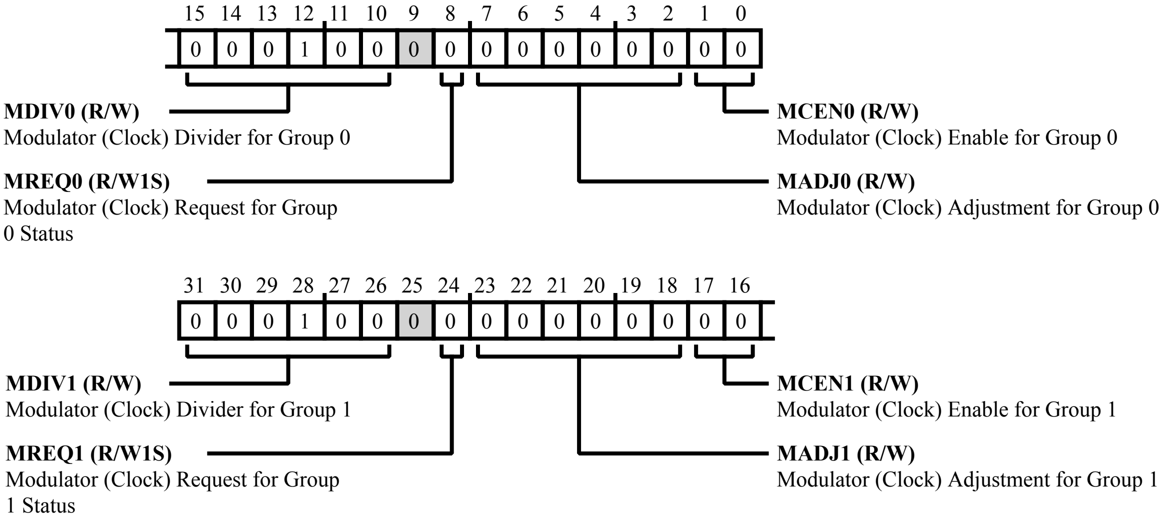

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
MDIV0 (R/W)
Modulator (Clock) Divider for Group 0
MREQ0 (R/W1S)
Modulator (Clock) Request for Group 0 Status
MCEN0 (R/W)
Modulator (Clock) Enable for Group 0
MADJ0 (R/W)
Modulator (Clock) Adjustment for Group 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
MDIV1 (R/W)
Modulator (Clock) Divider for Group 1
MREQ1 (R/W1S)
Modulator (Clock) Request for Group 1 Status
MCEN1 (R/W)
Modulator (Clock) Enable for Group 1
MADJ1 (R/W)
Modulator (Clock) Adjustment for Group 1

Figure 32-8: SINC_CLK Register Diagram

Table 32-9: SINC_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:26 (R/W)	MDIV1	Modulator (Clock) Divider for Group 1. The SINC_CLK.MDIV1 bits provide the SCLK0_0 divider to generate the modulator clock for group 1. The valid value is between 1 and 63.	
24 (R/W1S)	MREQ1	Modulator (Clock) Request for Group 1 Status. The SINC_CLK.MREQ1 bit indicates status for a phase shift request of the modulator clock for group 1. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 1 period is lengthened by the number of SCLK0_0 periods specified by the SINC_CLK.MADJ1 bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete.	
		0	Inactive
		1	Active
23:18 (R/W)	MADJ1	Modulator (Clock) Adjustment for Group 1. The SINC_CLK.MADJ1 bits provide the adjustment value for the modulator clock of group 1. The valid value is between 1 and 63 when SINC_CLK.MREQ1 is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the SINC_CLK.MREQ1 bit filed description.	

Table 32-9: SINC_CLK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17:16 (R/W)	MCEN1	Modulator (Clock) Enable for Group 1. The `SINC_CLK.MCEN1` bits enable/disable the modulator clock for group 1 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock). 0 Disable 1 Reserved 2 Enable and Commence 3 Enable and Commence on Next Rising Edge
15:10 (R/W)	MDIV0	Modulator (Clock) Divider for Group 0. The `SINC_CLK.MDIV0` bits provide the SCLK0_0 divider to generate the modulator clock for group 0. The valid value is between 1 and 63.
8 (R/W1S)	MREQ0	Modulator (Clock) Request for Group 0 Status. The `SINC_CLK.MREQ0` bit indicates status for a phase shift request of the modulator clock for group 0. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 0 period is lengthened by the number of SCLK0_0 periods specified by the `SINC_CLK.MADJ0` bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete. 0 Inactive 1 Active
7:2 (R/W)	MADJ0	Modulator (Clock) Adjustment for Group 0. The `SINC_CLK.MADJ0` bits provide the adjustment value for the modulator clock of group 0. The valid value is between 1 and 63 when `SINC_CLK.MREQ1` is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the `SINC_CLK.MREQ1` bit filed description.
1:0 (R/W)	MCEN0	Modulator (Clock) Enable for Group 0. The `SINC_CLK.MCEN0` bits enable/disable the modulator clock for group 0 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock). 0 Disable 1 Reserved 2 Enable and Commence 3 Enable and Commence on Next Rising Edge

Control Register

The `SINC_CTL` register masks (disables) and unmasks (enables) SINC high-level interrupt request output signals triggered by fault events. The register also enables and assigns SINC filter pairs to one of two control groups.

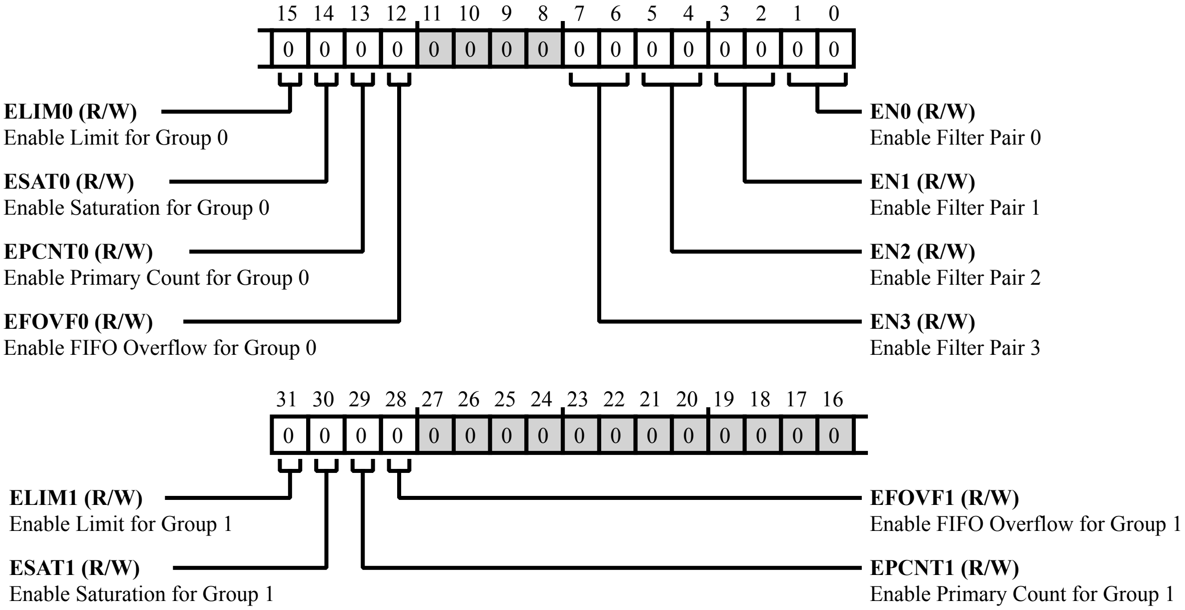

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ELIM0 (R/W)
Enable Limit for Group 0
ESAT0 (R/W)
Enable Saturation for Group 0
EPCNT0 (R/W)
Enable Primary Count for Group 0
EFOVF0 (R/W)
Enable FIFO Overflow for Group 0
EN0 (R/W)
Enable Filter Pair 0
EN1 (R/W)
Enable Filter Pair 1
EN2 (R/W)
Enable Filter Pair 2
EN3 (R/W)
Enable Filter Pair 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ELIM1 (R/W)
Enable Limit for Group 1
ESAT1 (R/W)
Enable Saturation for Group 1
EFOVF1 (R/W)
Enable FIFO Overflow for Group 1
EPCNT1 (R/W)
Enable Primary Count for Group 1

Figure 32-9: SINC_CTL Register Diagram

Table 32-10: SINC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	ELIM1	Enable Limit for Group 1. The `SINC_CTL.ELIM1` bit enables (unmasks) the SINC_STAT interrupt request on overload conditions if this bit and status bit `SINC_STAT.GLIM1` are set (=1).	
		0	Disable
		1	Enable
30 (R/W)	ESAT1	Enable Saturation for Group 1. The `SINC_CTL.ESAT1` bit enables (unmasks) the SINC_STAT interrupt request on output saturation conditions if this bit and bit `SINC_STAT.GSAT1` are set (=1).	
		0	Disable
		1	Enable

Table 32-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	EPCNT1	Enable Primary Count for Group 1. The `SINC_CTL.EPCNT1` bit enables a trigger event on each SINC_DATA1 interrupt request if this bit and status bit `SINC_STAT.PCNT1` are set (=1).	
		0	Disable
		1	Enable
28 (R/W)	EFOVF1	Enable FIFO Overflow for Group 1. The `SINC_CTL.EFOVF1` bit enables (unmasks) the SINC_STAT interrupt request on data FIFO overflow conditions if this bit and status bit `SINC_STAT.FOVF1` are set (=1). The `SINC_STAT.FOVF1` bit is set (=1) when the group 1 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable
15 (R/W)	ELIM0	Enable Limit for Group 0. The `SINC_CTL.ELIM0` bit enables (unmasks) the SINC_STAT interrupt request on overload conditions if this bit and status bit `SINC_STAT.GLIM0` are set (=1).	
		0	Disable
		1	Enable
14 (R/W)	ESAT0	Enable Saturation for Group 0. The `SINC_CTL.ESAT0` bit enables (unmasks) the SINC_STAT interrupt request on output saturation conditions if this bit and status bit `SINC_STAT.GSAT0` are set (=1).	
		0	Disable
		1	Enable
13 (R/W)	EPCNT0	Enable Primary Count for Group 0. The `SINC_CTL.EPCNT0` bit enables a trigger event on each SINC_DATA0 interrupt request if this bit and status bit `SINC_STAT.PCNT0` are set (=1).	
		0	Disable
		1	Enable

Table 32-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	EFOVF0	Enable FIFO Overflow for Group 0. The `SINC_CTL.EFOVF0` bit enables (unmasks) the SINC_STAT interrupt request on data FIFO overflow conditions if this bit and status bit `SINC_STAT.FOVF0` are set (=1). The `SINC_STAT.FOVF0` bit is set (=1) when the group 0 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable
7:6 (R/W)	EN3	Enable Filter Pair 3. The `SINC_CTL.EN3` bits enable/disable and assign SINC filter pair 3 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
5:4 (R/W)	EN2	Enable Filter Pair 2. The `SINC_CTL.EN2` bits enable/disable and assign SINC filter pair 2 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
3:2 (R/W)	EN1	Enable Filter Pair 1. The `SINC_CTL.EN1` bits enable/disable and assign SINC filter pair 1 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved

Table 32-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	EN0	Enable Filter Pair 0. The `SINC_CTL.EN0` bits enable/disable and assign SINC filter pair 0 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved

History Status Register

The `SINC_HIS_STAT` provides status for data histories of secondary SINC filters, in the corresponding history buffer registers. The SINC history buffer registers save the most recent filter samples once an overload fault signal is detected.

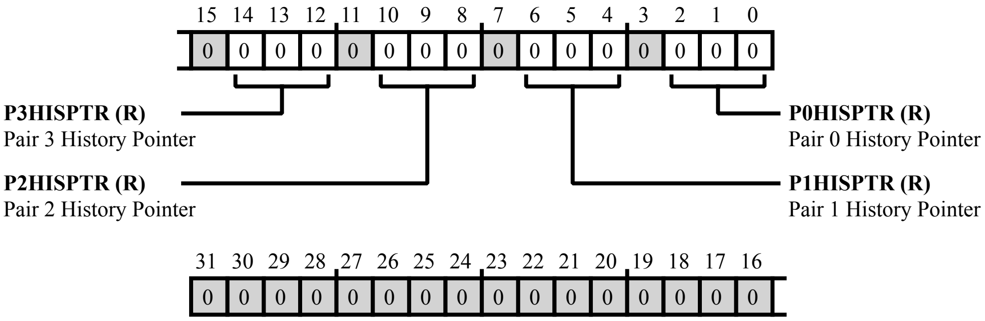

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P3HISPTR (R)
Pair 3 History Pointer
P2HISPTR (R)
Pair 2 History Pointer
P0HISPTR (R)
Pair 0 History Pointer
P1HISPTR (R)
Pair 1 History Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 32-10: SINC_HIS_STAT Register Diagram

Table 32-11: SINC_HIS_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/NW)	P3HISPTR	Pair 3 History Pointer. The `SINC_HIS_STAT.P3HISPTR` bits indicate the position for the most recent data sample of secondary SINC filter 3 in the corresponding `SINC_P3SEC_HIST[n]` register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
10:8 (R/NW)	P2HISPTR	Pair 2 History Pointer. The `SINC_HIS_STAT.P2HISPTR` bits indicate the position for the most recent data sample of secondary SINC filter 2 in the corresponding `SINC_P2SEC_HIST[n]` register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS

Table 32-11: SINC_HIS_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
6:4 (R/NW)	P1HISPTR	Pair 1 History Pointer. The `SINC_HIS_STAT.P1HISPTR` bits indicate the position for the most recent data sample of secondary SINC filter 1 in the corresponding `SINC_P1SEC_HIST[n]` register block.	
		0	History Register 3, MS
		1	History Register, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
2:0 (R/NW)	P0HISPTR	Pair 0 History Pointer. The `SINC_HIS_STAT.P0HISPTR` bits indicate the position for the most recent data sample of secondary SINC filter 0 in the corresponding `SINC_P0SEC_HIST[n]` register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS

Level Control for Group 0 Register

The `SINC_LEVEL0` register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 0.

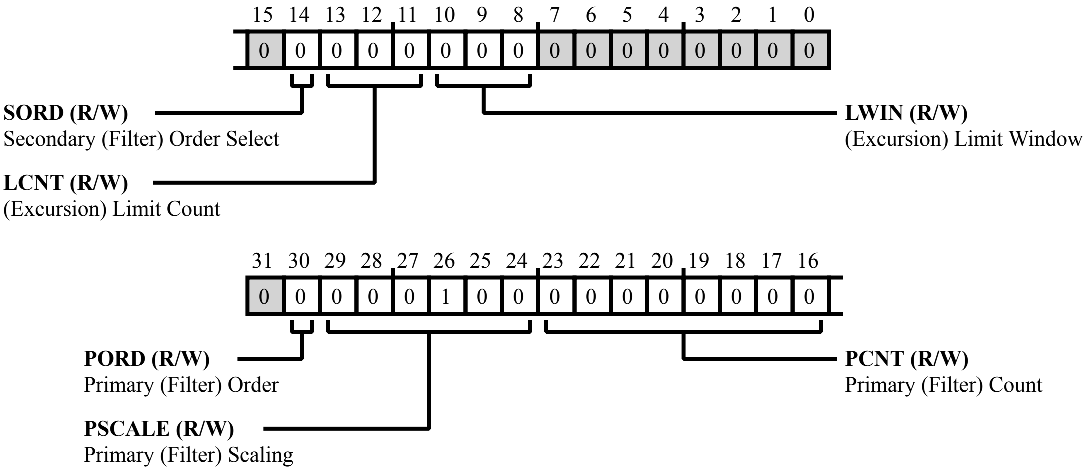

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SORD (R/W)
Secondary (Filter) Order Select
LCNT (R/W)
(Excursion) Limit Count
LWIN (R/W)
(Excursion) Limit Window
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
PORD (R/W)
Primary (Filter) Order
PSCALE (R/W)
Primary (Filter) Scaling
PCNT (R/W)
Primary (Filter) Count

Figure 32-11: SINC_LEVEL0 Register Diagram

Table 32-12: SINC_LEVEL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The `SINC_LEVEL0.PORD` bit determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order

Table 32-12: SINC_LEVEL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The `SINC_LEVEL0.PSCALE` bits specify the scaling applied to the output of group 0 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * `SINC_RATE0.PDEC`) ^ order)). The value requires about (ln2(`SINC_RATE0.PDEC`) * order) bits of precision (where 'order' is 3 or 4, as specified by the `SINC_LEVEL0.PORD` bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (`SINC_LEVEL0.PSCALE` - 16) bits if `SINC_LEVEL0.PSCALE` >= 16, or left-shifted by (16 - `SINC_LEVEL0.PSCALE`) bits if `SINC_LEVEL0.PSCALE` < 16. If `SINC_LEVEL0.PSCALE` >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (`SINC_STAT.PSAT3`, `SINC_STAT.PSAT2`, `SINC_STAT.PSAT1`, or `SINC_STAT.PSAT0` is set.
23:16 (R/W)	PCNT	Primary (Filter) Count. The `SINC_LEVEL0.PCNT` bits specify the modulo number of outputs for group 0 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the `SINC_STAT.PCNT0` status bit is set (=1). When the `SINC_STAT.PCNT0` bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the `SINC_LEVEL0.PCNT` bit field sets bit `SINC_STAT.PCNT0` to 1 after every primary SINC filter output is transferred. 8'hFF written to the `SINC_LEVEL0.PCNT` bit field sets bit `SINC_STAT.PCNT0` to 1 after every 256 primary SINC filter outputs transferred.
14 (R/W)	SORD	Secondary (Filter) Order Select. The `SINC_LEVEL0.SORD` bit determines the order for group 0 secondary filters. 0 Third Order 1 Fourth Order

Table 32-12: SINC_LEVEL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:11 (R/W)	LCNT	(Excursion) Limit Count. The `SINC_LEVEL0.LCNT` bits specify the number (count) of output excursions beyond the amplitude specified for group 0 secondary filters. The number of excursions greater than specified by registers `SINC_LIMIT3`, `SINC_LIMIT2`, `SINC_LIMIT2`, and `SINC_LIMIT0` is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the `SINC_STAT` register. The valid count is between 1 to 8. If the count is greater than `SINC_LEVEL0.LWIN`, the bit fields behavior is as it is equal to `SINC_LEVEL0.LWIN`. See `SINC_LEVEL0.LWIN` for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The `SINC_LEVEL0.LWIN` bits specify the window size for excursion checking for group 0 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the `SINC_LEVEL0.LCNT` bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

Level Control for Group 1 Register

The SINC_LEVEL1 register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 1.

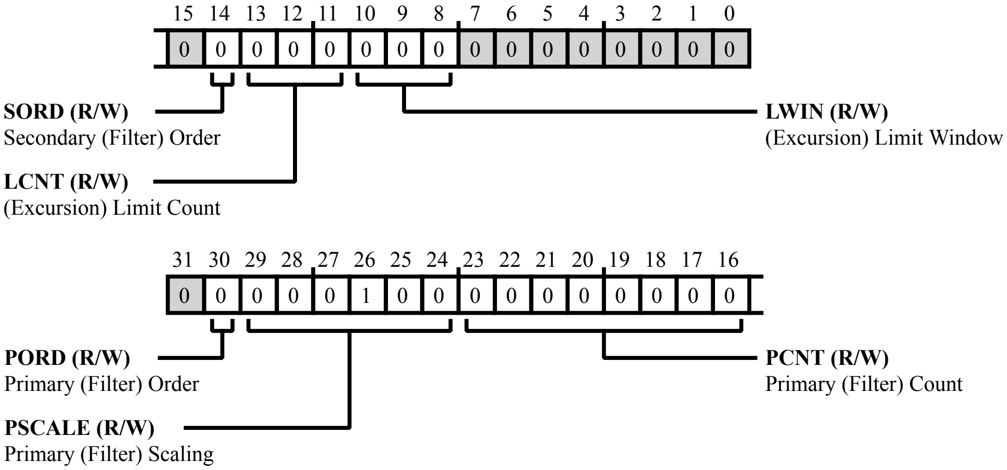

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SORD (R/W)
Secondary (Filter) Order
LCNT (R/W)
(Excursion) Limit Count
LWIN (R/W)
(Excursion) Limit Window
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
PORD (R/W)
Primary (Filter) Order
PSCALE (R/W)
Primary (Filter) Scaling
PCNT (R/W)
Primary (Filter) Count

Figure 32-12: SINC_LEVEL1 Register Diagram

Table 32-13: SINC_LEVEL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The `SINC_LEVEL1.PORD` bits determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order

Table 32-13: SINC_LEVEL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The `SINC_LEVEL1.PSCALE` bits specify the scaling applied to the output of group 1 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * `SINC_RATE1.PDEC`) ^ order)). The value requires about (ln2(`SINC_RATE1.PDEC`) * order) bits of precision (where 'order' is 3 or 4, as specified by the `SINC_LEVEL1.PORD` bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (`SINC_LEVEL1.PSCALE` - 16) bits if `SINC_LEVEL1.PSCALE` >= 16, or left-shifted by (16 - `SINC_LEVEL1.PSCALE`) bits if `SINC_LEVEL1.PSCALE` < 16. If `SINC_LEVEL1.PSCALE` >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (`SINC_STAT.PSAT3`, `SINC_STAT.PSAT2`, `SINC_STAT.PSAT1`, or `SINC_STAT.PSAT0` is set.
23:16 (R/W)	PCNT	Primary (Filter) Count. The `SINC_LEVEL1.PCNT` bits specify the modulo number of outputs for group 1 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the `SINC_STAT.PCNT1` status bit is set (=1). When the `SINC_STAT.PCNT1` bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the `SINC_LEVEL1.PCNT` bit field sets bit `SINC_STAT.PCNT1` to 1 after every primary SINC filter output is transferred. 8'hFF written to the `SINC_LEVEL1.PCNT` bit field sets bit `SINC_STAT.PCNT1` to 1 after every 256 primary SINC filter outputs transferred.
14 (R/W)	SORD	Secondary (Filter) Order. The `SINC_LEVEL1.SORD` bit determines the order for group 1 secondary filters. The `SINC_LEVEL1.SORD` bit determines the order for group 1 secondary filters. 0 Third Order 1 Fourth Order

Table 32-13: SINC_LEVEL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:11 (R/W)	LCNT	(Excursion) Limit Count. The `SINC_LEVEL1.LCNT` bits specify the number (count) of output excursions beyond the amplitude specified for group 1 secondary filters. The number of excursions greater than specified by registers `SINC_LIMIT3`, `SINC_LIMIT2`, `SINC_LIMIT2`, and `SINC_LIMIT0` is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the `SINC_STAT` register. The valid count is between 1 to 8. If the count is greater than `SINC_LEVEL1.LWIN`, the bit fields behavior is as it is equal to `SINC_LEVEL1.LWIN`. See `SINC_LEVEL1.LWIN` for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The `SINC_LEVEL1.LWIN` bits specify the window size for excursion checking for group 1 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the `SINC_LEVEL1.LCNT` bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

(Amplitude) Limits for Secondary Filter 0 Register

The SINC_LIMIT0 register controls amplitude limits for a secondary filter of SINC pair 0.

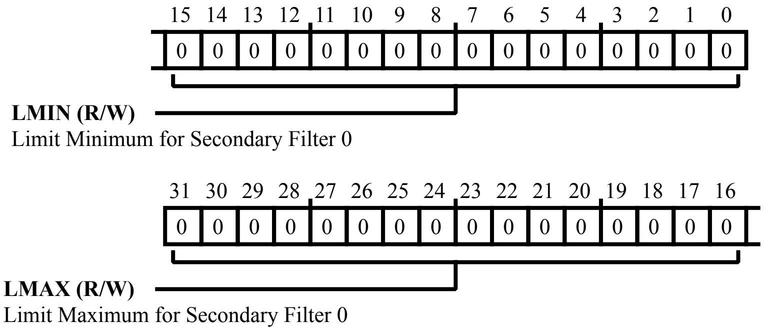

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMIN (R/W)
Limit Minimum for Secondary Filter 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMAX (R/W)
Limit Maximum for Secondary Filter 0

Figure 32-13: SINC_LIMIT0 Register Diagram

Table 32-14: SINC_LIMIT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 0. The `SINC_LIMIT0.LMAX` bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 0. The `SINC_LIMIT0.LMIN` bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 1 Register

The SINC_LIMIT1 register controls amplitude limits for a secondary filter of SINC pair 1.

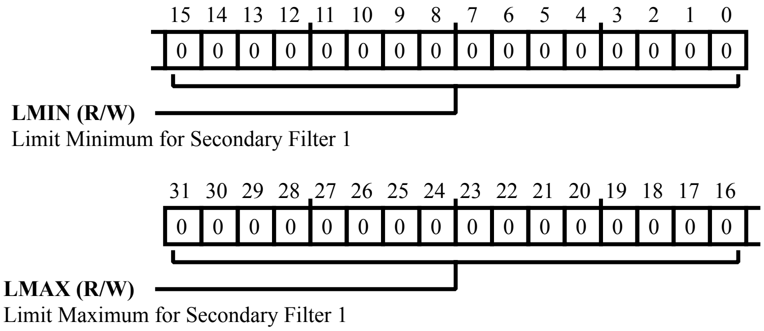

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMIN (R/W)
Limit Minimum for Secondary Filter 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMAX (R/W)
Limit Maximum for Secondary Filter 1

Figure 32-14: SINC_LIMIT1 Register Diagram

Table 32-15: SINC_LIMIT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 1. The `SINC_LIMIT1.LMAX` bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 1. The `SINC_LIMIT1.LMIN` bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 2 Register

The `SINC_LIMIT2` register controls amplitude limits for a secondary filter of SINC pair 2.

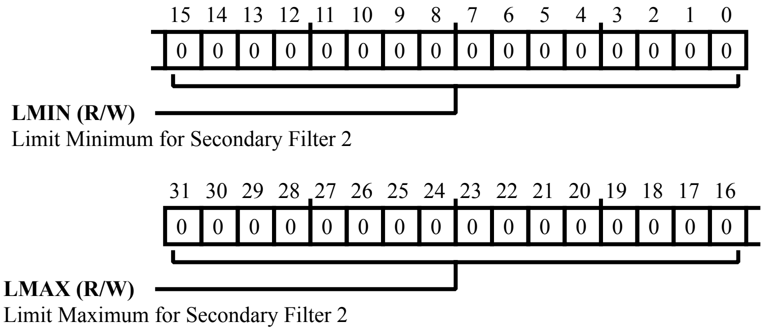

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMIN (R/W)
Limit Minimum for Secondary Filter 2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMAX (R/W)
Limit Maximum for Secondary Filter 2

Figure 32-15: SINC_LIMIT2 Register Diagram

Table 32-16: SINC_LIMIT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 2. The `SINC_LIMIT2.LMAX` bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register `SINC_LEVEL1` or `SINC_LEVEL0`, this bit field specifies conditions for an associated maximum limit warning bit in register `SINC_STAT`.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 2. The `SINC_LIMIT2.LMIN` bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register `SINC_LEVEL1` or `SINC_LEVEL0`, this bit field specifies conditions for an associated minimum limit warning bit in register `SINC_STAT`.

(Amplitude) Limits for Secondary Filter 3 Register

The SINC_LIMIT3 register controls amplitude limits for a secondary filter of SINC pair 3.

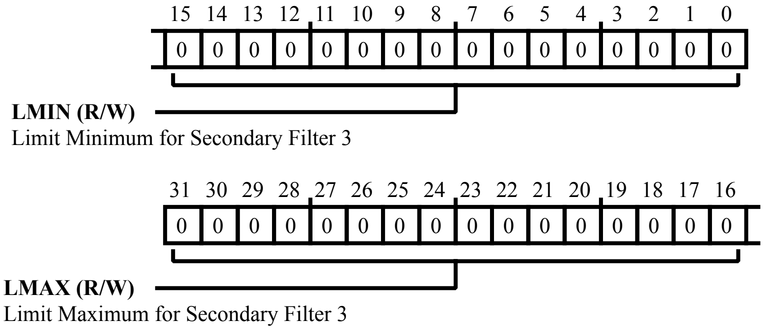

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMIN (R/W)
Limit Minimum for Secondary Filter 3
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LMAX (R/W)
Limit Maximum for Secondary Filter 3

Figure 32-16: SINC_LIMIT3 Register Diagram

Table 32-17: SINC_LIMIT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 3. The `SINC_LIMIT3.LMAX` bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register `SINC_LEVEL1` or `SINC_LEVEL0`, this bit field specifies conditions for an associated maximum limit warning bit in register `SINC_STAT`.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 3. The `SINC_LIMIT3.LMIN` bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register `SINC_LEVEL1` or `SINC_LEVEL0`, this bit field specifies conditions for an associated minimum limit warning bit in register `SINC_STAT`.

Pair 0 Secondary (Filter) History n Register

The `SINC_P0SEC_HIST[n]` read-only register provides the eight most recent samples produced by secondary SINC filter 0. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first `SINC_P0SEC_HIST[n]` register. The stored values, one compared to the limit, count, and window settings, set the `SINC_STAT.MAX0` and `SINC_STAT.MIN0` bits.

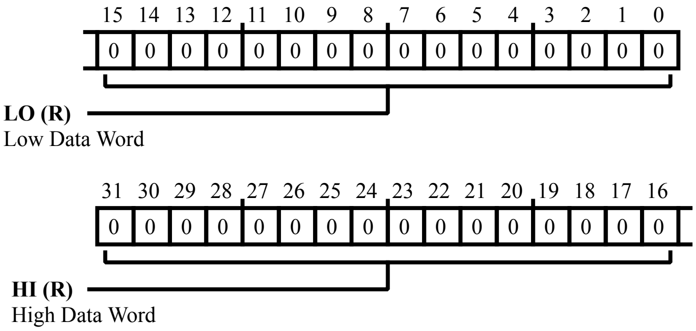

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LO (R)
Low Data Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HI (R)
High Data Word

Figure 32-17: SINC_P0SEC_HIST[n] Register Diagram

Table 32-18: SINC_P0SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The `SINC_P0SEC_HIST[n].HI` bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The `SINC_P0SEC_HIST[n].LO` bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 1 Secondary (Filter) History n Register

The `SINC_P1SEC_HIST[n]` read-only register provides the eight most recent samples produced by secondary SINC filter 1. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first `SINC_P1SEC_HIST[n]` register. The stored values, compared to the limit, count, and window settings, set the `SINC_STAT.MAX1` and `SINC_STAT.MIN1` bits.

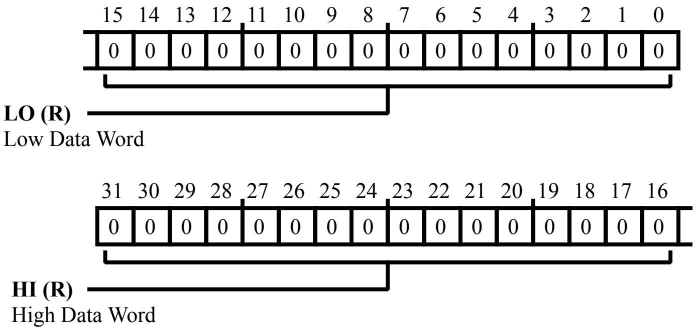

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LO (R)
Low Data Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HI (R)
High Data Word

Figure 32-18: SINC_P1SEC_HIST[n] Register Diagram

Table 32-19: SINC_P1SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The `SINC_P1SEC_HIST[n].HI` bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The `SINC_P1SEC_HIST[n].LO` bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 2 Secondary (Filter) History n Register

The `SINC_P2SEC_HIST[n]` read-only register provides the eight most recent samples produced by secondary SINC filter 2. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first `SINC_P2SEC_HIST[n]` register. The stored values, compared to the limit, count, and window settings, set the `SINC_STAT.MAX2` and `SINC_STAT.MIN2` bits.

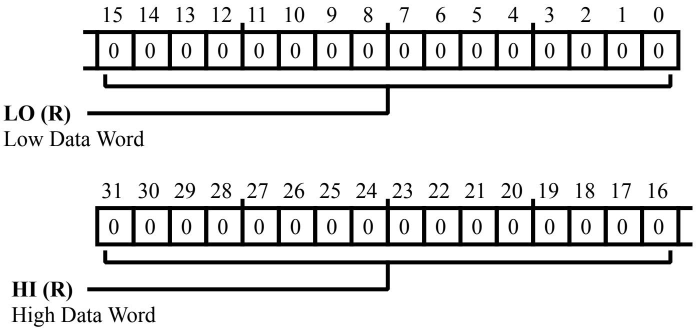

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LO (R)
Low Data Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HI (R)
High Data Word

Figure 32-19: SINC_P2SEC_HIST[n] Register Diagram

Table 32-20: SINC_P2SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The `SINC_P2SEC_HIST[n].HI` bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The `SINC_P2SEC_HIST[n].LO` bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 3 Secondary (Filter) History n Register

The `SINC_P3SEC_HIST[n]` read-only register provides the eight most recent samples produced by secondary SINC filter 3. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first `SINC_P3SEC_HIST[n]` register. The stored values, compared to the limit, count, and window settings, set the `SINC_STAT.MAX3` and `SINC_STAT.MIN3` bits.

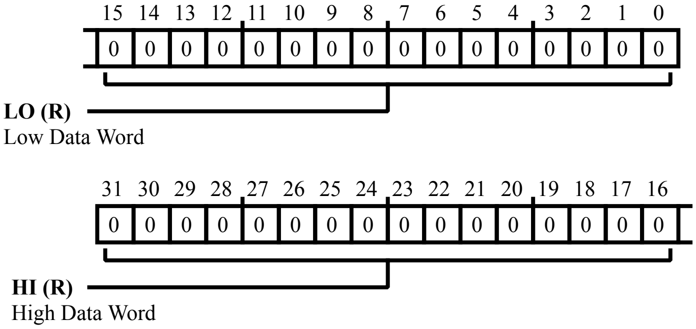

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LO (R)
Low Data Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HI (R)
High Data Word

Figure 32-20: SINC_P3SEC_HIST[n] Register Diagram

Table 32-21: SINC_P3SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The `SINC_P3SEC_HIST[n].HI` bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The `SINC_P3SEC_HIST[n].LO` bits provide the 16-bit sample in the least significant half of the 32- bit register.

Primary (Filters) Head for Group 0 Register

The `SINC_PHEAD0` register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 0 assignments).

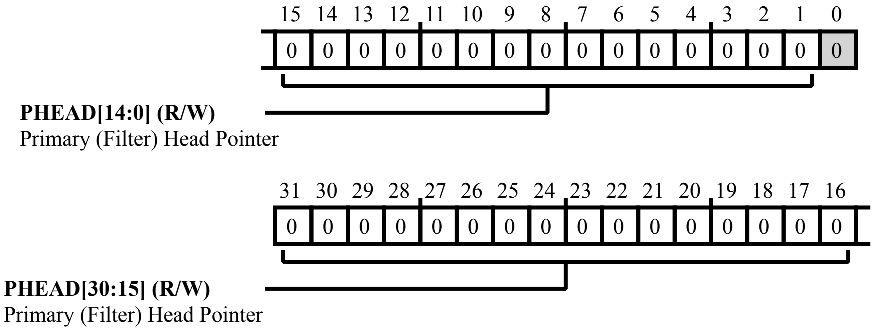

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHEAD[14:0] (R/W)
Primary (Filter) Head Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHEAD[30:15] (R/W)
Primary (Filter) Head Pointer

Figure 32-21: SINC_PHEAD0 Register Diagram

Table 32-22: SINC_PHEAD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PHEAD	Primary (Filter) Head Pointer. The `SINC_PHEAD0.PHEAD` bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to `SINC_PHEAD0.PHEAD` after `SINC_PTAIL0.PTAIL` is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 0). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Head for Group 1 Register

The `SINC_PHEAD1` register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

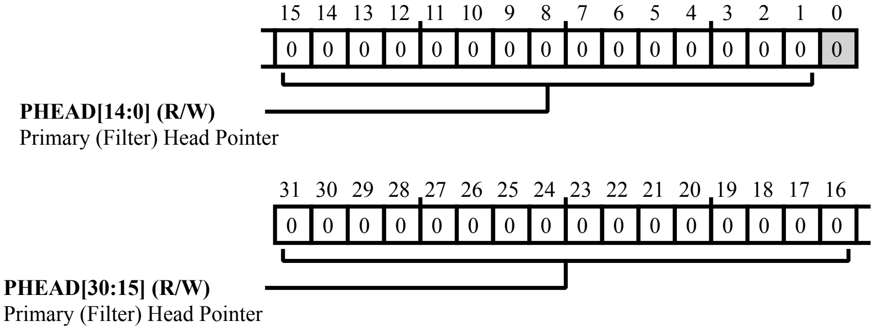

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHEAD[14:0] (R/W)
Primary (Filter) Head Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PHEAD[30:15] (R/W)
Primary (Filter) Head Pointer

Figure 32-22: SINC_PHEAD1 Register Diagram

Table 32-23: SINC_PHEAD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PHEAD	Primary (Filter) Head Pointer. The `SINC_PHEAD1.PHEAD` bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to `SINC_PHEAD1.PHEAD` after `SINC_PTAIL1.PTAIL` is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Pointer for Group 0 Register

The `SINC_PPTR0` read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 0 assignments.

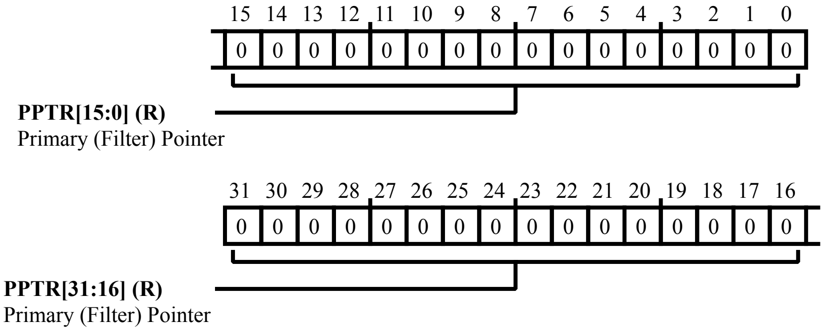

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPTR[15:0] (R)
Primary (Filter) Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPTR[31:16] (R)
Primary (Filter) Pointer

Figure 32-23: SINC_PPTR0 Register Diagram

Table 32-24: SINC_PPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The `SINC_PPTR0.PPTR` bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 0). The address is incremented once all of the primary SINC filter data (assigned to group 0 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Pointer for Group 1 Register

The `SINC_PPTR1` read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 1 assignments.

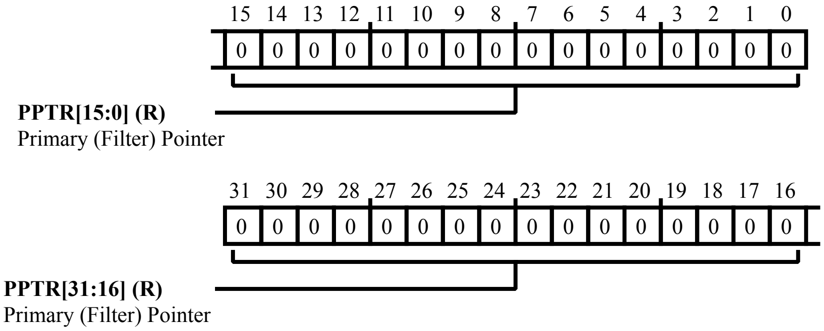

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPTR[15:0] (R)
Primary (Filter) Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PPTR[31:16] (R)
Primary (Filter) Pointer

Figure 32-24: SINC_PPTR1 Register Diagram

Table 32-25: SINC_PPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The `SINC_PPTR1.PPTR` bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 1). The address is incremented once all of the primary SINC filter data (assigned to group 1 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Tail for Group 0 Register

The `SINC_PTAIL0` register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

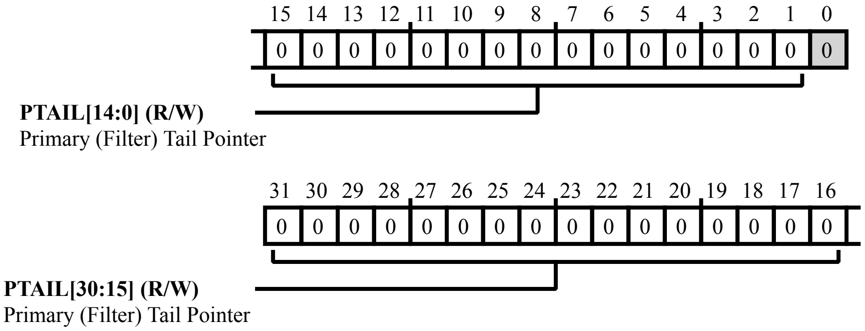

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PTAIL[14:0] (R/W)
Primary (Filter) Tail Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PTAIL[30:15] (R/W)
Primary (Filter) Tail Pointer

Figure 32-25: SINC_PTAIL0 Register Diagram

Table 32-26: SINC_PTAIL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The `SINC_PTAIL0.PTAIL` bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to `SINC_PHEAD0.PHEAD` after `SINC_PTAIL0.PTAIL` is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Tail for Group 1 Register

The `SINC_PTAIL1` register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

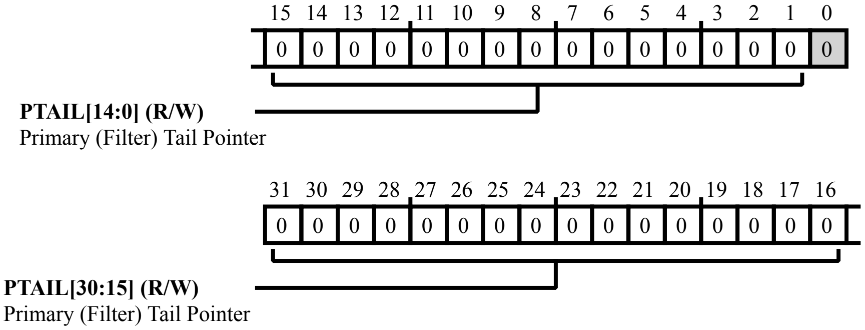

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PTAIL[14:0] (R/W)
Primary (Filter) Tail Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PTAIL[30:15] (R/W)
Primary (Filter) Tail Pointer

Figure 32-26: SINC_PTAIL1 Register Diagram

Table 32-27: SINC_PTAIL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The `SINC_PTAIL1.PTAIL` bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to `SINC_PHEAD1.PHEAD` after `SINC_PTAIL1.PTAIL` is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Rate Control for Group 0 Register

The `SINC_RATE0` register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 0.

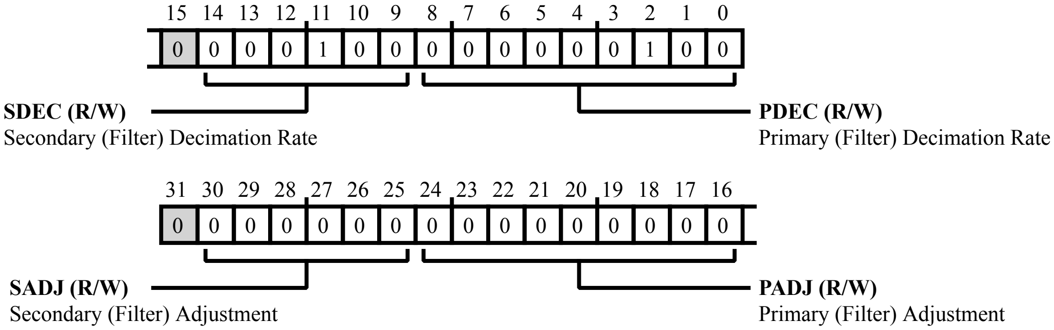

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
SDEC (R/W)
Secondary (Filter) Decimation Rate
PDEC (R/W)
Primary (Filter) Decimation Rate
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SADJ (R/W)
Secondary (Filter) Adjustment
PADJ (R/W)
Primary (Filter) Adjustment

Figure 32-27: SINC_RATE0 Register Diagram

Table 32-28: SINC_RATE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The `SINC_RATE0.SADJ` bits provide the phase adjustment for the decimated output of group 0 secondary filters. The valid adjustment is between 0 and (`SINC_RATE0.SDEC` - 1), in modulator clock cycles, relative to the time the filter is enabled in the `SINC_CTL` register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((`SINC_RATE0.SDEC` * n) - `SINC_RATE0.SADJ`), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The `SINC_RATE0.PADJ` bits provide the phase adjustment for the decimated output of group 0 primary filters. The valid adjustment is between 0 and (`SINC_RATE0.PDEC` - 1), in modulator clock cycles, relative to the time the filter is enabled in the `SINC_CTL` register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((`SINC_RATE0.PDEC` * n) - `SINC_RATE0.PADJ`), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.

Table 32-28: SINC_RATE0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The `SINC_RATE0.SDEC` bits provide the decimation rate for group 0 secondary filters. The valid range depends on the SINC order selected. If the third order (`SINC_LEVEL0.SORD` = 0), the valid range is 4 to 40. If the forth order (`SINC_LEVEL0.SORD` = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The `SINC_RATE0.PDEC` bits provide the decimation rate for group 0 primary filters. The valid rate is 256 to 4.

Rate Control for Group 1 Register

The `SINC_RATE1` register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 1.

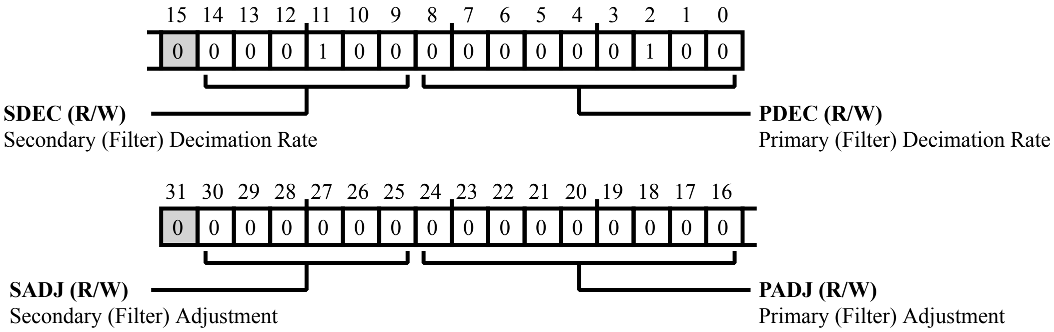

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
SDEC (R/W)
Secondary (Filter) Decimation Rate
PDEC (R/W)
Primary (Filter) Decimation Rate
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SADJ (R/W)
Secondary (Filter) Adjustment
PADJ (R/W)
Primary (Filter) Adjustment

Figure 32-28: SINC_RATE1 Register Diagram

Table 32-29: SINC_RATE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The `SINC_RATE1.SADJ` bits provide the phase adjustment for the decimated output of group 1 secondary filters. The valid adjustment is between 0 and (`SINC_RATE1.SDEC` - 1), in modulator clock cycles, relative to the time the filter is enabled in the `SINC_CTL` register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((`SINC_RATE1.SDEC` * n) - `SINC_RATE1.SADJ`), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The `SINC_RATE1.PADJ` bits provide the phase adjustment for the decimated output of group 1 primary filters. The valid adjustment is between 0 and (`SINC_RATE1.PDEC` - 1), in modulator clock cycles, relative to the time the filter is enabled in the `SINC_CTL` register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((`SINC_RATE1.PDEC` * n) - `SINC_RATE1.PADJ`), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.

Table 32-29: SINC_RATE1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The `SINC_RATE1.SDEC` bits provide the decimation rate for group 1 secondary filters. The valid range depends on the SINC order selected. If the third order (`SINC_LEVEL1.SORD` = 0), the valid range is 4 to 40. If the forth order (`SINC_LEVEL1.SORD` = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The `SINC_RATE1.PDEC` bits provide the decimation rate for group 1 primary filters. The valid rate is 256 to 4.

Status Register

The `SINC_STAT` register indicates status for SINC output saturation, amplitude and duration limits, overload conditions, and data transfer errors.

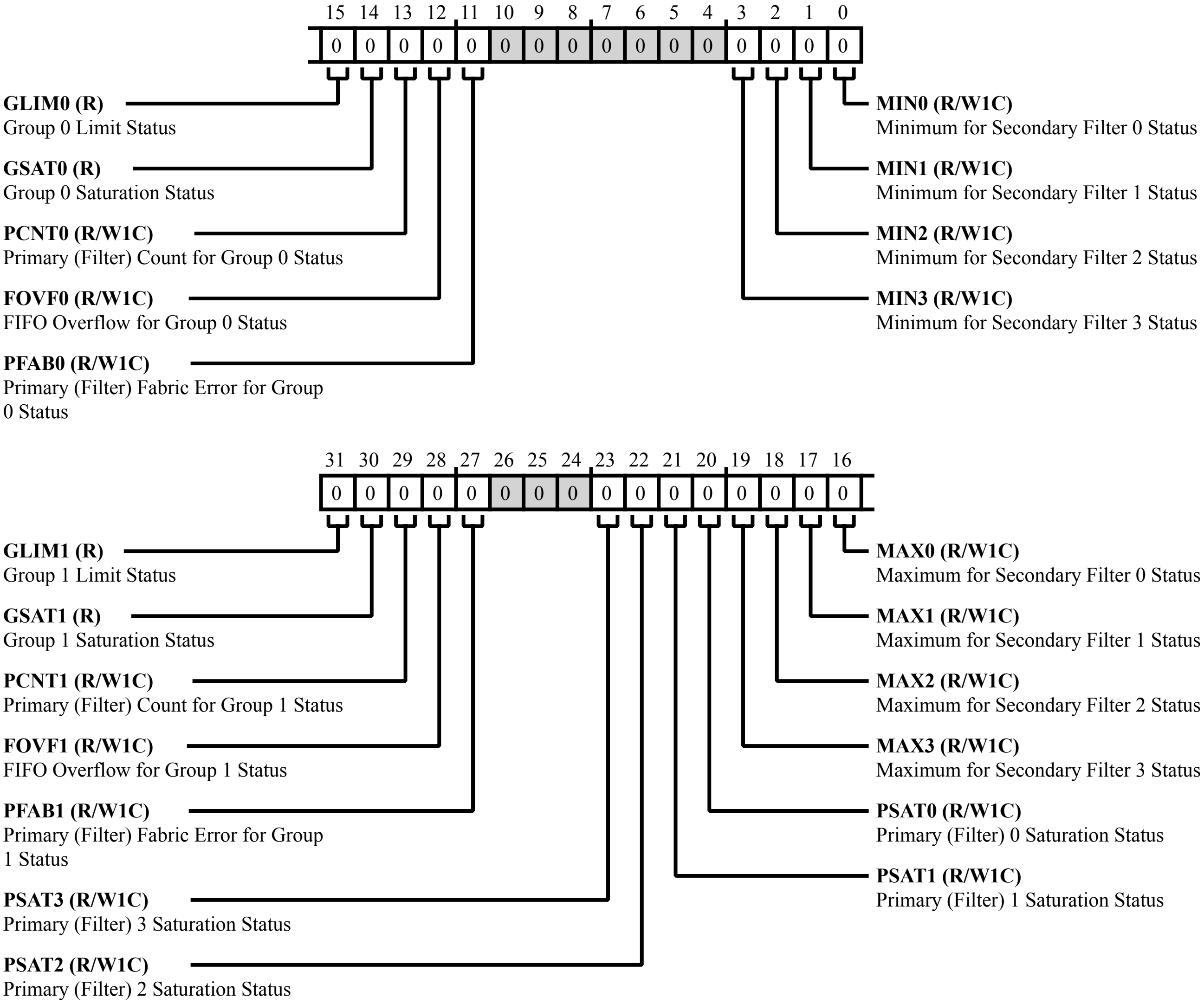

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GLIM0 (R)
Group 0 Limit Status
GSAT0 (R)
Group 0 Saturation Status
PCNT0 (R/W1C)
Primary (Filter) Count for Group 0 Status
FOVF0 (R/W1C)
FIFO Overflow for Group 0 Status
PFAB0 (R/W1C)
Primary (Filter) Fabric Error for Group 0 Status
MIN0 (R/W1C)
Minimum for Secondary Filter 0 Status
MIN1 (R/W1C)
Minimum for Secondary Filter 1 Status
MIN2 (R/W1C)
Minimum for Secondary Filter 2 Status
MIN3 (R/W1C)
Minimum for Secondary Filter 3 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
GLIM1 (R)
Group 1 Limit Status
GSAT1 (R)
Group 1 Saturation Status
PCNT1 (R/W1C)
Primary (Filter) Count for Group 1 Status
FOVF1 (R/W1C)
FIFO Overflow for Group 1 Status
PFAB1 (R/W1C)
Primary (Filter) Fabric Error for Group 1 Status
PSAT3 (R/W1C)
Primary (Filter) 3 Saturation Status
PSAT2 (R/W1C)
Primary (Filter) 2 Saturation Status
MAX0 (R/W1C)
Maximum for Secondary Filter 0 Status
MAX1 (R/W1C)
Maximum for Secondary Filter 1 Status
MAX2 (R/W1C)
Maximum for Secondary Filter 2 Status
MAX3 (R/W1C)
Maximum for Secondary Filter 3 Status
PSAT0 (R/W1C)
Primary (Filter) 0 Saturation Status
PSAT1 (R/W1C)
Primary (Filter) 1 Saturation Status

Figure 32-29: SINC_STAT Register Diagram

Table 32-30: SINC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	GLIM1	Group 1 Limit Status. The `SINC_STAT.GLIM1` indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 1. This bit is set (=1) if any limit specified by registers `SINC_LIMIT3`, `SINC_LIMIT2`, `SINC_LIMIT1`, or `SINC_LIMIT0`, within the duration count and window specified by bits `SINC_LEVEL0.LCNT` and `SINC_LEVEL0.LWIN` are exceeded. To identify the offending secondary SINC filter, examine the filters status bits `SINC_STAT.MAX3`, `SINC_STAT.MAX2`, `SINC_STAT.MAX1`, `SINC_STAT.MAX0`, `SINC_STAT.MIN3`, `SINC_STAT.MIN2`, `SINC_STAT.MIN1` and `SINC_STAT.MAX0` according to the group 1 assignments in the `SINC_CTL` register.	
		0	Not Exceeded
		1	Exceeded
30 (R/NW)	GSAT1	Group 1 Saturation Status. The `SINC_STAT.GSAT1` indicates status for the output saturation bit of primary SINC filters assigned to group 1. The bit is set (=1) if any filter of group 1 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits `SINC_STAT.PSAT3`, `SINC_STAT.PSAT2`, `SINC_STAT.PSAT1`, and `SINC_STAT.PSAT0` according to the group 1 assignments specified by the `SINC_CTL.EN3`, `SINC_CTL.EN2`, `SINC_CTL.EN1`, and `SINC_CTL.EN0` bits.	
		0	Not Set
		1	Set
29 (R/W1C)	PCNT1	Primary (Filter) Count for Group 1 Status. The `SINC_STAT.PCNT1` indicates status for the output count of primary SINC filters assigned to group 1. The bit is set (=1) each time the modulo number of outputs (specified by the `SINC_LEVEL1.PCNT` bits) has been transferred for each primary SINC filter assigned to group 1. Each count in `SINC_LEVEL1.PCNT` corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 1 is assigned three SINC filters pairs 0, 1, and 3, and `SINC_LEVEL1.PCNT` is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit `SINC_CTL.EPCNT1` are set (=1), the SINC_DATA1 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W1C)	FOVF1	FIFO Overflow for Group 1 Status. The `SINC_STAT.FOVF1` indicates status for the data output FIFO bit of primary SINC filters assigned to group 1. This bit is set (= 1) if the output FIFO for any filter in group 1 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit `SINC_CTL.EFOVF1` are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
27 (R/W1C)	PFAB1	Primary (Filter) Fabric Error for Group 1 Status. The `SINC_STAT.PFAB1` indicates error status for the output of any primary SINC filter assigned to group 1. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 1, or if an overrun occurs for a filter in group 1. An interrupt is requested whenever this bit =1 (not maskable).	
		0	Disabled
		1	Enabled
23 (R/W1C)	PSAT3	Primary (Filter) 3 Saturation Status. The `SINC_STAT.PSAT3` bit indicates whether the primary SINC filter 3 requires saturation.	
		0	Not Saturated
		1	Saturated
22 (R/W1C)	PSAT2	Primary (Filter) 2 Saturation Status. The `SINC_STAT.PSAT2` bit indicates whether the primary SINC filter 2 requires saturation.	
		0	Not Saturated
		1	Saturated

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PSAT1	Primary (Filter) 1 Saturation Status. The `SINC_STAT.PSAT1` bit indicates whether the primary SINC filter 1 requires saturation.	
		0	Not Saturated
		1	Saturated
20 (R/W1C)	PSAT0	Primary (Filter) 0 Saturation Status. The `SINC_STAT.PSAT0` bit indicates whether the primary SINC filter 0 requires saturation.	
		0	Not Saturated
		1	Saturated
19 (R/W1C)	MAX3	Maximum for Secondary Filter 3 Status. The `SINC_STAT.MAX3` bit indicates whether the output of the secondary SINC filter 3 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT3.LMAX` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN3` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded
18 (R/W1C)	MAX2	Maximum for Secondary Filter 2 Status. The `SINC_STAT.MAX2` bit indicates whether the output of the secondary SINC filter 2 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT2.LMAX` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN2` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	MAX1	Maximum for Secondary Filter 1 Status. The `SINC_STAT.MAX1` bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT1.LMAX` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN1` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded
16 (R/W1C)	MAX0	Maximum for Secondary Filter 0 Status. The `SINC_STAT.MAX0` bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT0.LMAX` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN0` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	GLIM0	Group 0 Limit Status. The `SINC_STAT.GLIM0` indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 0. This bit is set (=1) if any limit specified by registers `SINC_LIMIT3`, `SINC_LIMIT2`, `SINC_LIMIT1`, or `SINC_LIMIT0`, within the duration count and window specified by bits `SINC_LEVEL1.LCNT` and `SINC_LEVEL1.LWIN` are exceeded. To identify the offending secondary SINC filter, examine the filters status bits `SINC_STAT.MAX3`, `SINC_STAT.MAX2`, `SINC_STAT.MAX1`, `SINC_STAT.MAX0`, `SINC_STAT.MIN3`, `SINC_STAT.MIN2`, `SINC_STAT.MIN1` and `SINC_STAT.MAX0` according to the group 0 assignments in the `SINC_CTL` register.	
		0	Not Exceeded
		1	Exceeded
14 (R/NW)	GSAT0	Group 0 Saturation Status. The `SINC_STAT.GSAT0` indicates status for the output saturation bit of primary SINC filters assigned to group 0. The bit is set (=1) if any filter of group 0 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits `SINC_STAT.PSAT3`, `SINC_STAT.PSAT2`, `SINC_STAT.PSAT1`, and `SINC_STAT.PSAT0` according to the group 0 assignments specified by the `SINC_CTL.EN3`, `SINC_CTL.EN2`, `SINC_CTL.EN1`, and `SINC_CTL.EN0` bits.	
		0	Not Set
		1	Set
13 (R/W1C)	PCNT0	Primary (Filter) Count for Group 0 Status. The `SINC_STAT.PCNT0` indicates status for the output count of primary SINC filters assigned to group 0. The bit is set (=1) each time the modulo number of outputs (specified by the `SINC_LEVEL0.PCNT` bits) has been transferred for each primary SINC filter assigned to group 0. Each count in `SINC_LEVEL0.PCNT` corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 0 is assigned three SINC filters pairs 0, 1, and 3, and `SINC_LEVEL0.PCNT` is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit `SINC_CTL.EPCNT0` are set (=1), the SINC_DATA0 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	FOVF0	FIFO Overflow for Group 0 Status. The `SINC_STAT.FOVF0` indicates status for the data output FIFO bit of primary SINC filters assigned to group 0. This bit is set (= 1) if the output FIFO for any filter in group 0 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit `SINC_CTL.EFOVF0` are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
11 (R/W1C)	PFAB0	Primary (Filter) Fabric Error for Group 0 Status. The `SINC_STAT.PFAB0` indicates error status for the output of any primary SINC filter assigned to group 0. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 0, or if an overrun occurs for a filter in group 0. An interrupt is requested whenever this bit is =1 (not maskable).	
		0	Disabled
		1	Enabled
3 (R/W1C)	MIN3	Minimum for Secondary Filter 3 Status. The `SINC_STAT.MIN3` bit indicates whether the output of the secondary SINC filter 3 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT3.LMIN` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN3` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	MIN2	Minimum for Secondary Filter 2 Status. The `SINC_STAT.MIN2` bit indicates whether the output of the secondary SINC filter 2 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT2.LMIN` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN2` bits. For group 0, the duration limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the duration limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples. 0 Not Exceeded 1 Exceeded
1 (R/W1C)	MIN1	Minimum for Secondary Filter 1 Status. The `SINC_STAT.MIN1` bit indicates whether the output of the secondary SINC filter 1 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT1.LMIN` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN1` bits. For group 0, the limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples. 0 Not Exceeded 1 Exceeded

Table 32-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	MIN0	Minimum for Secondary Filter 0 Status. The `SINC_STAT.MIN0` bit indicates whether the output of the secondary SINC filter 0 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT0.LMIN` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN0` bits. For group 0, the limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded

33 Digital Audio Interface (DAI)

The Digital Audio Interfaces (DAIn) are comprised of groups of identical peripherals and their respective Signal Routing Units (SRUn). The SRU connects inputs and outputs of the DAI peripherals with each other and to the external pins. This configuration allows peripherals to be interconnected to accomodate a wide variety of systems without making external pin connections.

In a typical processor, static (multiplexed) pins are assigned to specific peripherals. When certain peripherals are not required for an application, these pins are unnecessary and expensive. The pins may need to be defined as high or low to prevent any illegal conditions. The signal routing units on the SHARC processors addresses this situation by controlling a number of general-purpose pins which can be assigned flexibly (a virtual connectivity between peripherals) depending on system requirements. This virtual connectivity includes pin buffers and routing logic (multiplexer). It also allows the SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

SRU Features

The SRU has the following features and capabilities.

- Flexible connections that can be made through software and during run time; no hard-wiring is required.
- At reset, a default routing scheme is already programmed.
- Connectivity can be made internally between peripherals, externally between pin buffers, or a mix of both.
- Status of the pin buffers can be programmed for conditional execution or interrupts.
- Some pin buffers allow control of signal polarity changes.
- No fan-out limitation, a peripheral or pin buffer output that can be routed to multiple peripheral or pin buffer inputs.
- Two independent routing systems are availablethe DAI0 and the DAI1. Signals can't be interconnected between both routing units with the exception of the following modules under some specific scenarios.
    - The precision clock generator (PCG)
    - A pair of DAI pin buffers from each DAI unit
    - Asynchronous Sample Rate Converter (ASRC) data for daisy chaining across the DAI units

NOTE: The 349–ball CSP BGA package has 2 x 14 DAI pins bonded off-chip (DAIx_PIN01 through DAIx_PIN12, DAIx_PIN19 and DAIx_PIN20).

Functional Description

The fundamental timing clock of the DAI modules is SCLK0_0.

The DAI Block Diagram shows how the DAI pin buffers are connected through the SRUn. This configuration allows for flexible signal routing.

The DAI units are comprised of four primary blocks:

- Peripherals (A/B/C) associated with DAIn
- Signal Routing Units (SRUn)
- DAIn I/O pin buffers
- Miscellaneous buffers

The peripherals shown in ***DAI Functional Block Diagram*** can have up to three connections (if master or slave capable); one acts as a signal input, one as a signal output and the third as an output enable. The SRUs are based on a group of multiplexers which are controlled by registers to establish the desired interconnections. The DAI pin buffers have three signals which are used for input and output to or from off-chip and the third for output enable.

The miscellaneous buffers have an input and an output and are used for group interconnection.

The figures are a simplified representation of a DAI system. In a real representation, the SRU and DAI would show several types of data being routed from several sources including the following:

- Serial ports (SPORT)
- Precision clock generators (PCG)
- Asynchronous sample rate converters (SRC)
- S/PDIF transmitter
- S/PDIF receiver
- DAI interrupts (miscellaneous)

ADSP-SC58x DAI Register List

The Digital Audio Interfaces (DAIn) contain groups of identical peripherals which can be connected internally between peripherals, externally between pin buffers, or a mix of both. This module contains the following registers.

Table 33-1: ADSP-SC58x DAI Register List

Name	Description
`DAI_CLK0`	Clock Routing Control Register 0

Table 33-1: ADSP-SC58x DAI Register List (Continued)

Name	Description
DAI_CLK1	Clock Routing Control Register 1
DAI_CLK2	Clock Routing Control Register 2
DAI_CLK3	Clock Routing Control Register 3
DAI_CLK4	Clock Routing Control Register 4
DAI_CLK5	Clock Routing Control Register 5
DAI_DAT0	Serial Data Routing Control Register 0
DAI_DAT1	Serial Data Routing Control Register 1
DAI_DAT2	Serial Data Routing Control Register 2
DAI_DAT3	Serial Data Routing Control Register 3
DAI_DAT4	Serial Data Routing Control Register 4
DAI_DAT5	Serial Data Routing Control Register 5
DAI_DAT6	Serial Data Routing Control Register 6
DAI_FS0	Frame Sync Routing Control Register 0
DAI_FS1	Frame Sync Routing Control Register 1
DAI_FS2	Frame Sync Routing Control Register 2
DAI_FS4	Frame Sync Routing Control Register 4
DAI_IMSK_FE	Falling-Edge Interrupt Mask Register
DAI_IMSK_PRI	Core Interrupt Priority Assignment Register
DAI_IMSK_RE	Rising-Edge Interrupt Mask Register
DAI_IRPTL_H	High Priority Interrupt Latch Register
DAI_IRPTL_HS	Shadow High Priority Interrupt Latch Register
DAI_IRPTL_L	Low Priority Interrupt Latch Register
DAI_IRPTL_LS	Shadow Low Priority Interrupt Latch Register
DAI_MISC0	Miscellaneous Control Register 0
DAI_MISC1	Miscellaneous Control Register 1
DAI_PBEN0	Pin Buffer Enable Register 0
DAI_PBEN1	Pin Buffer Enable Register 1
DAI_PBEN2	Pin Buffer Enable Register 2
DAI_PBEN3	Pin Buffer Enable Register 3
DAI_PIN0	Pin Buffer Assignment Register 0
DAI_PIN1	Pin Buffer Assignment Register 1

Table 33-1: ADSP-SC58x DAI Register List (Continued)

Name	Description
`DAI_PIN2`	Pin Buffer Assignment Register 2
`DAI_PIN3`	Pin Buffer Assignment Register 3
`DAI_PIN4`	Pin Buffer Assignment Register 4
`DAI_PIN_STAT`	Pin Status Register

ADSP-SC58x DAI Interrupt List

Table 33-2: ADSP-SC58x DAI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
24	DAI0_IRQH	DAI0 High Priority Interrupt		
25	DAI1_IRQH	DAI1 High Priority Interrupt		
145	DAI0_IRQL	DAI0 Low Priority Interrupt		
146	DAI1_IRQL	DAI1 Low Priority Interrupt		

DAI Block Diagram

The *DAI Functional Block Diagram* and the *Digital Audio Interconnect Unit* figures show the functional blocks within the DAI and the unit connections to the peripherals.

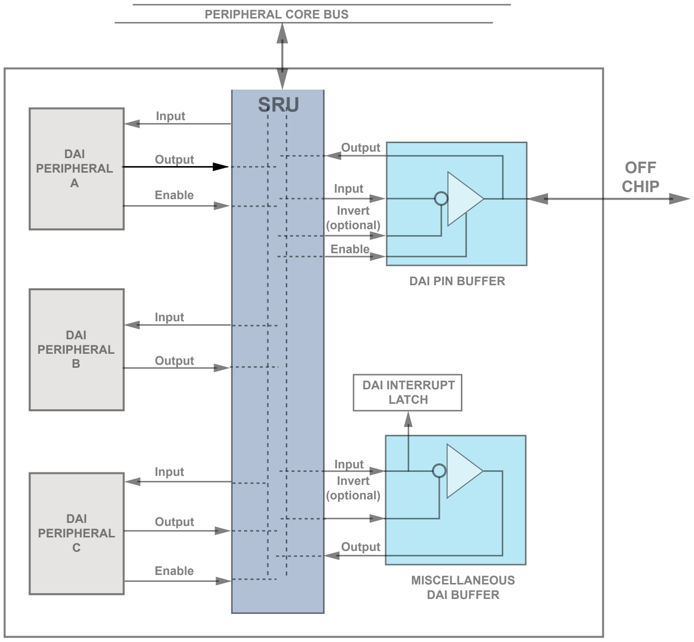

PERIPHERAL CORE BUS
SRU
DAI PERIPHERAL A
Input
Output
Enable
DAI PERIPHERAL B
Input
Output
DAI PERIPHERAL C
Input
Output
Enable
Output
Input
Invert (optional)
Enable
DAI PIN BUFFER
OFF CHIP
DAI INTERRUPT LATCH
Input
Invert (optional)
Output
MISCELLANEOUS DAI BUFFER

Figure 33-1: DAI Functional Block Diagram

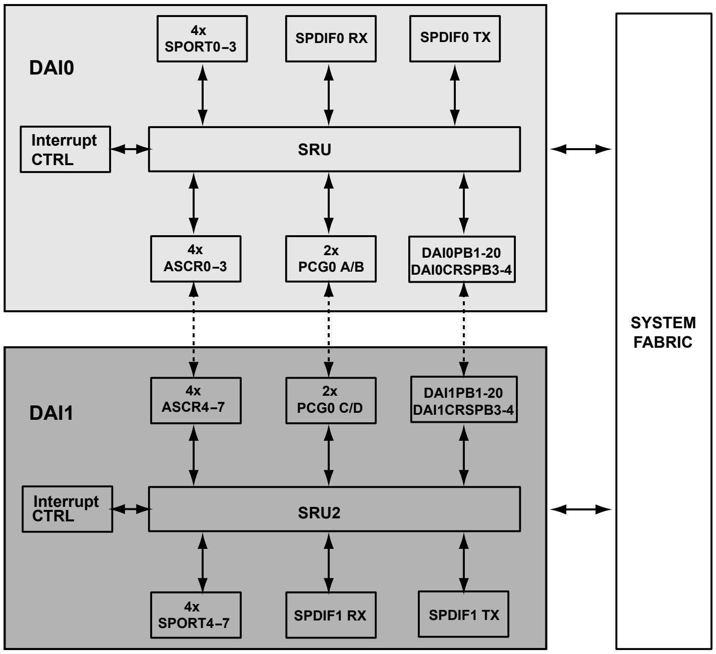

DAI0
4x SPORT0–3
SPDIF0 RX
SPDIF0 TX
Interrupt CTRL
SRU
4x ASCR0–3
2x PCG0 A/B
DAI0PB1-20 DAI0CRSPB3-4
SYSTEM FABRIC
DAI1
4x ASCR4–7
2x PCG0 C/D
DAI1PB1-20 DAI1CRSPB3-4
Interrupt CTRL
SRU2
4x SPORT4–7
SPDIF1 RX
SPDIF1 TX

Figure 33-2: Digital Audio Interconnect Unit

DAI Signal Naming Conventions

The peripherals associated with the DAI do not have any dedicated I/O pins for off-chip communication. Instead, the I/O pin is only accessible in the chip internally and is known as an *internal node*. Every internal node of a DAI peripheral (input or output) is given a unique mnemonic. The convention is to begin the name with an identifier for the peripheral that the signal is coming to or from, followed by the function of the signal.

A number is included if the DAI contains more than one peripheral type (for example, serial ports), or if the peripheral has more than one signal that performs this function (for example, serial ports). The mnemonic always ends with *_I* if the signal is an input, or with *_O* if the signal is an output. An example is shown in the ***Example DAI Mnemonics*** figure.

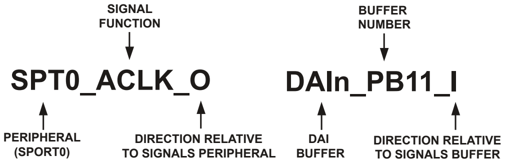

SIGNAL FUNCTION
BUFFER NUMBER
SPT0_ACLK_O
DAIn_PB11_I
PERIPHERAL (SPORT0)
DIRECTION RELATIVE TO SIGNALS PERIPHERAL
DAI BUFFER
DIRECTION RELATIVE TO SIGNALS BUFFER

Figure 33-3: Example DAI Mnemonics

I/O Pin Buffers

Within the context of the SRU, physical connections to the DAI pins are replaced by a logical interface known as a *pin buffer*. This three terminal active device is capable of sourcing or sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has an input, an output, and an enable as shown in the ***Pin Buffer Example*** figure. The inputs and the outputs are defined with respect to the pin, similar to a peripheral device. This naming convention is consistent with the SRU naming convention.

Pin Buffer Signals

The pin buffer is based on three signals shown in the ***Pin Buffer Example*** figure and described in the following sections.

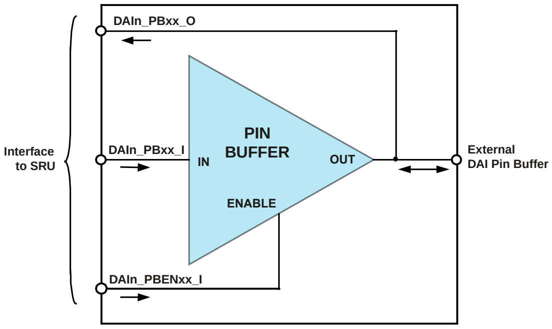


Figure 33-4: Pin Buffer Example

Pin Buffer Input Signal

A pin buffer input (`DAIn_PBxx_I`) is driven as an output from the processor when the pin buffer enable is set (=1). Each physical pin (connected to a bonded pad) can be connected through the SRU to any of the outputs of the DAI peripherals, based on the bit field values. The SRU can also be used to route signals that control the pins in other ways. Many signals can be configured for use as control signals.

Pin Buffer Enable Signal

When a pin buffer enable (`DAIn_PBENxx_I`) is set (=1), the signal present at the corresponding pin buffer input (`DAIn_PBxx_I`) is driven off-chip as an output. When a pin buffer enable is cleared (=0), the signal present at the corresponding pin buffer input is ignored. The pin enable control registers activate the drive buffer for each of the DAI pins. When the pins are not enabled (driven), they can be used as inputs. There are two options to control the pin buffer enable signal; setting the level high for a static solution, or connecting the dedicated peripheral's pin buffer output enable signal to its pin buffer, which automatically enables the pin buffer.

Pin Buffer Input Level State

The DAI pads are provided with an input enable signal (IE). The signal is used to ensure that the input path (`DAI_PBxx_O`) is pushed to a known state (versus a floating state in case of no external drivers on the DAI pad). This state can be considered as an AND gate of the pad signal with the IE as shown in the pin buffer figure. The IE signal is controlled from the PAD logic (`PADS_DAIx_IE` register) and is provided at a granularity at each DAI pin. (Refer to the GP port chapter.) Every DAI pin can be individually controlled depending on whether the pin is used in a system or not. The PADS register controlling the IE signal must have a reset value of 0 and hence all the pin buffer inputs are (`DAI_PBxx_O`) gated to 0.

NOTE: In previous SHARC products, this default value was 1 due to a weak pull-up.

Once the PADS module register is programmed to enable the IE signal on the used DAI pin buffers, the pin-buffer outputs its signal on `DAI_PBxx_O`. The programming model needs to enable all used DAI pin buffers. For unused DAI pin-buffers (no external system connection), keep the corresponding IE-bit at a disabled state.

Pin Buffer Functions

Pin buffers can be configured as inputs or outputs as described in the following sections.

Pin Buffers as Signal Input

When the DAI pin is used only as an input, connect the corresponding pin buffer enable to logic low as shown in the ***Pin Buffer as Input*** figure. This configuration disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI pin and at the pin buffer output. When the pin buffer enable (for example, `DAI_PBEN0.PB01`) is cleared (= 0), the pin buffer output (`DAIn_PBxx_O`) is the signal driven onto the DAI pin by an external source, and the pin buffer input signal (`DAIn_PBxx_I`) is not used.

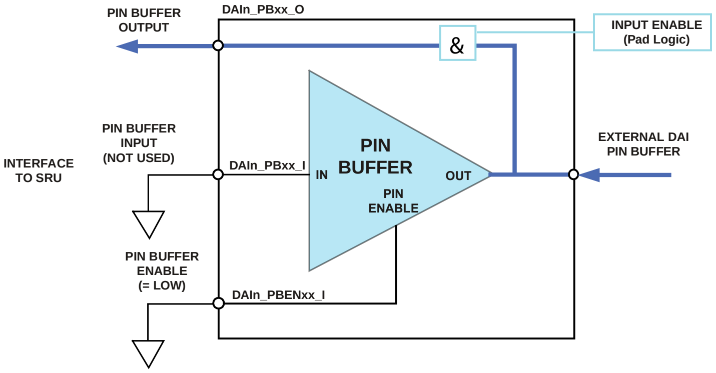


Figure 33-5: Pin Buffer as Input

NOTE: Whether programmed as input or output, a DAI buffer input always routes the same signal to an output internally. The DAI also has an additional programming requirement when it is configured as input. The corresponding pin must be enabled in the `PADS_DAIx_IE` register. The `DAI_PB01_O` pin cannot be routed as an input source for clock (Group A signals) and frame sync (Group C). For more information, see DAI Routing Capabilities.

Pin Buffers As Signal Output

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they can be used in either direction. Each of the DAI pins can be used as either an output or an input. Although the direction of a DAI pin is set simply by writing to a memory-mapped register, most often the direction of the pin is dictated by the designated use of that pin.

When the DAI pin is used only as an output, connect the corresponding pin buffer enable to logic high as shown in the ***Pin Buffer as Output*** figure. This configuration enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer input onto the DAI pin and off-chip. When the pin buffer enable bits are set (in the `DAIn_PBxx_I` registers) (=1), the pin buffer output (`DAIn_PBxx_O`) is the same signal as the pin buffer input (`DAIn_PBxx_I`), and this signal is driven as an output.

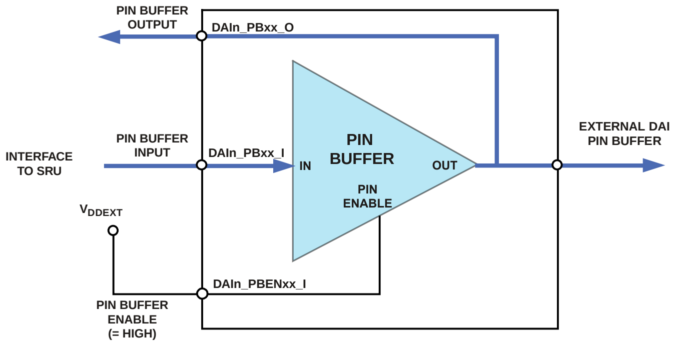


Figure 33-6: Pin Buffer as Output

DAI Pin Buffer Status

The signal levels on the DAI pins can be read with the `DAI_PIN_STAT` registers.

DAIn Peripherals

There are two categories of peripherals associated with the DAI units. These are described in the following sections.

Output Signals With Pin Buffer Enable Control

Many peripherals within the DAIn that have bidirectional pins generate a corresponding pin enable signal. Typically, the settings within the control registers of a peripheral determine if a bidirectional pin is an input or an output. The pin is then driven accordingly.

Though most peripherals are capable of operating bidirectionally, it is not required that all of the `_I` and `_O` signals of a peripheral be connected to the pin buffer. If the system design only uses a signal in one direction, it is simpler to connect the pin buffer accordingly.

NOTE: All available pin buffer output enables must be routed to their pin buffer input enable signals in cases where data streaming connections are used. This arrangement guarantees timing requirements.

NOTE: In some cases, it is necessary to use a peripheral's dedicated pin buffer enable signal instead of static levels. For example, SPORT TDM mode requires the SPORT's dedicated data pin buffer enable signal to be used for the SPORT's data pin to three-state the data pins on inactive channels.

Output Signals Without Pin Buffer Enable Control

Some peripherals have signal outputs without an automated pin buffer control enable signal. The operation of these peripherals is simplified. The routing to a DAIn pin buffer enable input requires a static high from the SRUn. In order to disable the pin buffer output, software must clear the pin buffer enable input accordingly.

Signal Routing Units (SRUs)

The following sections provide details specific to the SRUs.

Cross Mode Connections

The DAI allows a few signals to be interconnected across DAI units (see Figure 33-2 Digital Audio Interconnect Unit). These connections are commonly referred as *cross mode connections*. These cross mode connections are useful for the system designer to share signals across both DAI's so that synchronization can be achieved on the peripheral modules across DAI0 and DAI1.

NOTE: The 349-ball CSP BGA package has 2 x 14 DAI pins bonded off-chip and continues to support 2 x 20 pin buffers for internal routings.

Shared Clock. This cross mode connection (see Figure 33-7 Cross Connect Clock Example) can be used to share a clock signal cross DAIs. `DAI0_PIN03` and `DAI1_PIN03` can be used to share clock signal across DAI. When DAI0_PIN03 is configured as input clock pin then this clock signal is available in Group A of both DAIs. This source signal is `DAI0_PB03_O` in group A of DAI 0 and `DAI1_CRS_PB03_O` in group A of DAI1.

Peripherals routed through DAI0 like PCG A and SPORT 0A can use `DAI0_PB03_O` as the input clock signal. Peripherals routed through DAI1 like PCG C, SPORT 4A can use `DAI1_CRS_PB03` as the input clock signal. Similarly when `DAI1_PIN03` is configured as the input clock pin, then this clock signal is available as `DAI1_PB03_O` in group A of DAI1 and `DAI0_CRS_PB03` in group A of DAI0.

Shared Frame Sync. This cross mode connection can be used to share a frame sync signal cross DAIs. `DAI0_PIN04` and `DAI1_PIN04` can be used to share the frame sync signal across the DAIs. When `DAI0_PIN04` is configured as the input frame sync pin then this frame sync signal is available in Group C of both DAIs. This source signal is `DAI0_PB04_O` in group C of DAI 0 and `DAI1_CRS_PB04_O` in group C of DAI1.

Peripherals on DAI0 like SPORT 0A can use `DAI0_PB04_O` as the input frame sync signal. The peripherals on DAI1 like SPORT 4A can use `DAI1_CRS_PB04` as the input sync signal. Similarly when `DAI1_PIN04` is configured as input frame sync pin, then this frame sync signal is available as `DAI1_PB04_O` in group A of DAI1 and `DAI0_CRS_PB04` in group C of DAI0.

Shared Pin Buffer. This cross mode connection can be used to connect input signal on `DAI0_PIN03`, `DAI0_PIN04`, `DAI1_PIN03` and `DAI1_PIN04` to other DAI pins across DAI. When `DAI0_PIN03` and `DAI0_PIN04` are configured as input signals. These signals can be routed to the pin buffers of DAI1 in Group D using `DAI1_CRS_PIN03` and `DAI1_CRS_PIN04`. When `DAI1_PIN03` and `DAI1_PIN04` are configured as input signals they can be routed to the pin buffers of DAI0 in Group D using `DAI0_CRS_PIN03` and `DAI0_CRS_PIN04` signals.

PCGs and Pin Buffers. This cross mode connection (see Figure 33-8 Cross Connect PCG Example) can be used to connect PCG output signals of one DAI to pin buffers of other DAI. For example, PCG C outputs of DAI 1 can be connected to pin buffers of DAI 0 using `PCG0_CRS_CLKC_O` of Group D.

ASRC TDM and Data Signal. This cross mode connection is available in Group B (Data Signals). This allows programs to connect the Data and TDM signals of SRC3 and SRC 7 across DAIs so that eight SRCs can be daisy chained for TDM 16 operation. For example `SRC3_DAT_OP_O` and `SRC3_TDM_IP_O` of DAI0 are available as `SRC3_CRS_DAT_OP_O` and `SRC3_CRS_IP_O` in DAI1 so that DAI0 SRC3 can be connected to DAI1's SRC for daisy chaining.

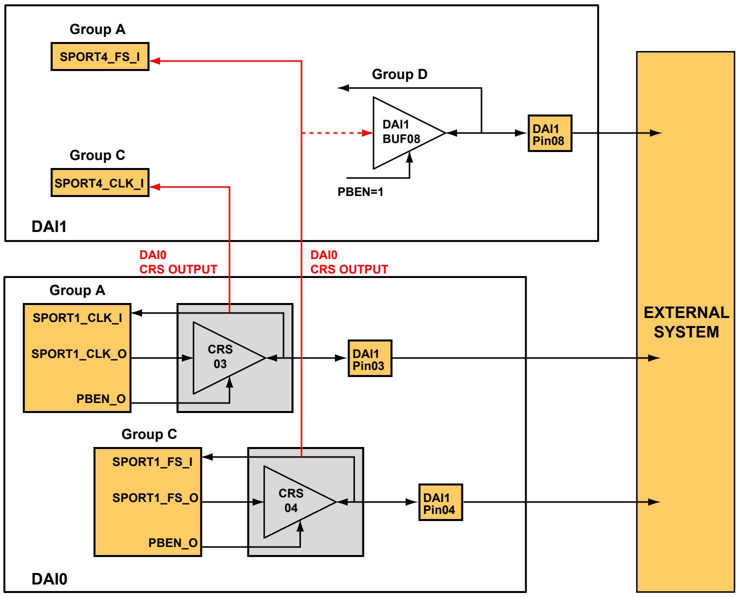

Group A
SPORT4_FS_I
Group D
DAI1 BUF08
DAI1 Pin08
Group C
SPORT4_CLK_I
PBEN=1
DAI1
DAI0 CRS OUTPUT
DAI0 CRS OUTPUT
Group A
SPORT1_CLK_I
SPORT1_CLK_O
PBEN_O
CRS 03
DAI1 Pin03
EXTERNAL SYSTEM
Group C
SPORT1_FS_I
SPORT1_FS_O
PBEN_O
CRS 04
DAI1 Pin04
DAI0

Figure 33-7: Cross Connect Clock Example

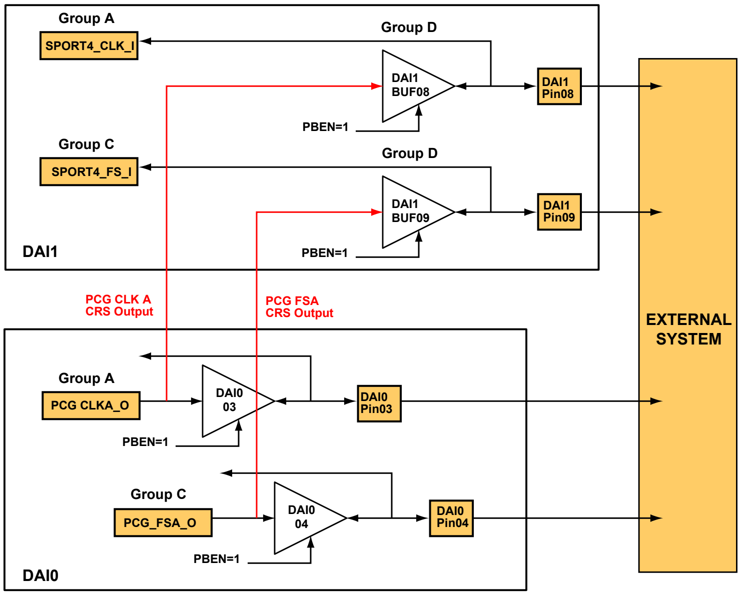

Group A
SPORT4_CLK_I
Group D
DAI1 BUF08
DAI1 Pin08
PBEN=1
Group C
SPORT4_FS_I
Group D
DAI1 BUF09
DAI1 Pin09
DAI1
PBEN=1
PCG CLK A CRS Output
PCG FSA CRS Output
EXTERNAL SYSTEM
Group A
PCG CLKA_O
DAI0 03
DAI0 Pin03
PBEN=1
Group C
PCG_FSA_O
DAI0 04
DAI0 Pin04
PBEN=1
DAI0

Figure 33-8: Cross Connect PCG Example

Refer to the DAI Group Routing section for more details on DAI routing.

Signal Routing Matrix by Groups

The SRU is similar to a set of patch bays, which contain a bank of inputs and a bank of outputs. For each input (destination), there is a set of permissible output (source) options. Outputs can feed to any number of inputs in parallel, but every input must be patched to exactly one valid output source. Together, the set of inputs and outputs are called a group. The signal's inputs and outputs that comprise each group all serve similar purposes. They are compatible such that almost any output-to-input patch makes functional sense. With the grouping, the multiplexing scheme becomes highly efficient since it does not make sense (for example) to route a frame sync signal to a data signal.

The SRU for the DAI contains six groups named A through F. Each group routes a unique set of signals with a specific purpose:

- Group A routes clock signals
- Group B routes serial data signals
- Group C routes frame sync signals
- Group D routes pin signals
- Group E routes miscellaneous signals
- Group F routes pin output enable signals

Together, the six groups of the SRU include all of the inputs and outputs of the DAI peripherals, a number of additional signals from the core, and all of the connections to the DAI pins.

NOTE: It is not possible to connect a signal in one group directly to a signal in a different group (analogous to wiring from one patch bay to another). However, group D (DAI) is largely devoted to routing in this vein.

DAI Group Routing

Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, DAI group A is used to route clock signals. The memory-mapped group A registers, `DAI_CLK0` through `DAI_CLK5`, contain bit fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems). The ***Example DAI Group A Multiplexing (DAI_CLKx)*** diagrams the input signals that are controlled by the group A registers. All bit fields in the SRU configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the SRU.

The SRU is similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRU, a bit pattern that is associated with a signal output (shown in the ***Example DAI Group A Multiplexing (DAI_CLKx)*** figure) is written to a bit field corresponding to a signal input.

The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

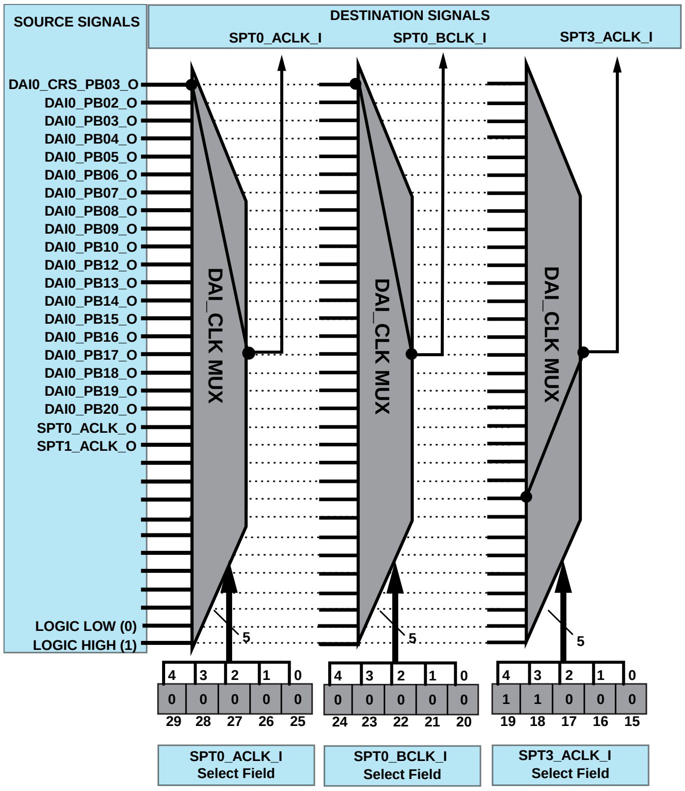

SOURCE SIGNALS
DESTINATION SIGNALS
SPT0_ACLK_I
SPT0_BCLK_I
SPT3_ACLK_I
DAI0_CRS_PB03_O
DAI0_PB02_O
DAI0_PB03_O
DAI0_PB04_O
DAI0_PB05_O
DAI0_PB06_O
DAI0_PB07_O
DAI0_PB08_O
DAI0_PB09_O
DAI0_PB10_O
DAI0_PB12_O
DAI0_PB13_O
DAI0_PB14_O
DAI0_PB15_O
DAI0_PB16_O
DAI0_PB17_O
DAI0_PB18_O
DAI0_PB19_O
DAI0_PB20_O
SPT0_ACLK_O
SPT1_ACLK_O
LOGIC LOW (0)
LOGIC HIGH (1)
DAI_CLK MUX
DAI_CLK MUX
DAI_CLK MUX
5
5
5
4 3 2 1 0
0 0 0 0 0
29 28 27 26 25
4 3 2 1 0
0 0 0 0 0
24 23 22 21 20
4 3 2 1 0
1 1 0 0 0
19 18 17 16 15
SPT0_ACLK_I Select Field
SPT0_BCLK_I Select Field
SPT3_ACLK_I Select Field

Figure 33-9: Example DAI Group A Multiplexing (DAIn_CLKx)

Just as group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. All of the groups have an encoding that allow a signal to flow from a pin output to the input being specified by the bit field.

Group D routes signals to pins so that they may be driven off-chip (required to route a signal to the pin input). Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. The input of one pin can be patched to the output of another pin, allowing board-level routing under software control.

Rules for SRU Connections

There are two rules which apply to all routing:

1. One source (output node) can drive different destinations (input nodes).
2. One destination (input node) can only be assigned to one source (output node).

As an example from the ***Example DAI Group A Multiplexing (DAI_CLKx)*** figure:

- `DAI0_CRS_PB03_O` is routed to `SPT0_ACLK_I`
- `DAI0_CRS_PB03_O` is routed to `SPT0_BCLK_I`
- `SPT4_ACLK_O` is routed to `SPT3_ACLK_I`

NOTE: Inputs may only be connected to outputs.

Miscellaneous Buffers and Functions

The SRU group E provides miscellaneous buffers used for group interconnect.

DAI group E connections are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals and provides a means of connecting signals between groups.

In the ***DAI MISCAx SRU Signal Connections*** table, the `DAIn_MISCAx_I` signals appear as inputs in group E (also connected to the DAI interrupt logic), but do not directly feed any peripheral. Rather, the `MISCAx_O` signals reappear as outputs in group F.

Table 33-3: DAI MISCAx SRU Signal Connections

MISCA Source	DAI Connection	MISCA Destination
	Group E	DAIn_MISCA5-0_I
DAIn_MISCA5-0_O	Group F	

Additional connections among groups provide a great amount of utility. Since the output groups F (DAI) dictate pin direction, these few signal paths enable a number of possible uses and connections for the DAI pins. Other examples include:

- A pin input can be patched to another pin's enable, allowing an off-chip signal to gate an output from the processor.
- Any of the DAI pins can be used as interrupt sources or general-purpose I/O (GPIO) signals.

In summary, the SRU enables many possible functional changes, both within the processor as well as externally. Used creatively, it allows system designers to radically change functionality at run time, and to potentially reuse circuit boards across many products.

DAI Routing Capabilities

This section describes the routing options to aid in designing a system using the DAI units. The Cross Mode Connections section provides information on connecting signals across DAI0 and DAI1. The DAI Default Routing section provides diagrams that show how that DAIs connect at default. The DAI Group tables provide the source signals and selections codes. Finally, the ADSP-SC58x DAI Register Descriptions provides information about configuring destinations.

The ***DAI Routing Capabilities*** tables provide an overview of the different routing capabilities for the DAI unit. For information on an individual peripherals routing, see the "SRU Programming" section of the specific peripheral chapter.

Table 33-4: DAI0 Routing Capabilities

Source Signals - Output (xxxx_O)		DAI0 Group	Destination Signals - Input (xxxx_I)
SPORT2B–0A (clocks) PCG A, B S/PDIF0 Rx (clock, TDM clock)	DAI0 pin buffer 20–2 DAI0 CRS pin buffer (buffer 3 from other DAI) Logic level high Logic level low	A-clocks	SPORT3B–0A (clocks) SRC3–0 (clocks) PCG A–B ext. clock, ext. sync S/PDIF0 Tx (clock, HF clock)
SPORT3B–0A (data) SRC3–0 (data, TDM data) S/PDIF0 Tx/Rx (data) SRC7 from other DAI (data, TDM data)	DAI0 pin buffer 20–1 Logic level high Logic level low	B-data	SPORT3B–0A (data) SRC3–0 (data, TDM data) S/PDIF0 Tx/Rx (data)
SPORT2B–0A (FS) PCG A, B (FS) S/PDIF0 Rx (FS)	DAI0 pin buffer 20–2 DAI0 CRS pin buffer (buffer 4 from other DAI) Logic level high Logic level low	C-frame sync	SPORT3B–0A (FS) SRC3–0 (FS)
SPORT3B–0A (clock, FS, TDV, data) PCG A,B (clock, FS) S/PDIF0 Rx (clock, TDM clock, FS, data) S/PDIF0 Tx (data, block start) PCG C, D from the other DAI (clock, FS)	DAI0 pin buffer 20–1 DAI0 pin buffer 3 and 4 from other DAI Logic level high Logic level low	D-pin buffer inputs	DAI0 pin buffer 20–1 Options: DAI0 pin buffer 20–19 Polarity change
SPORT2B–0A (FS) PCG A (clock) PCG B (clock, FS) S/PDIF0 Tx (block start)	DAI0 pin buffer 20–1 Logic level high Logic level low	E-miscellaneous signals	***DAI Interrupt 31-22*** MISCA5–0 Options: MISCA5– Polarity change
SPORT3B–0A (clock, FS, data, TDV) MISCA5-0	DAI0 pin buffer 20–1 Logic level high Logic level low	F-pin buffer enable	DAI0 pin buffer enable 20–1

Table 33-5: DAI1 Routing Capabilities

Source Signals - Output (xxxx_O)		DAI1 Group	Destination Signals - Input (xxxx_I)
SPORT6B–4A (clocks) PCG C, D (clocks) S/PDIF1 Rx (clock, TDM clock)	DAI1 Pin Buffer 20–2 DAI1 CRS pin buffer (buffer 3 from other DAI) Logic level high Logic level low	A-Clocks	SPORT7B–4A (clocks) SRC4–7 (clocks) PCG C–D Ext. clock, Ext. sync S/PDIF1 Tx (clock, HF clock)
SPORT6B–4A (data) SRC4–7 (data, TDM data) S/PDIF1 Tx/Rx (data) SRC3 from other DAI (data, TDM data)	DAI1 Pin Buffer 20–1 Logic level high Logic level low	B-Data	SPORT7B–4A (data) SRC4–7 (data, TDM data) S/PDIF1 Tx/Rx (data)
SPORT6B–4A (FS) PCG C, D (FS) S/PDIF1 Rx (FS)	DAI1 Pin Buffer 20–2 DAI1 CRS pin buffer (buffer 4 from other DAI) Logic level high Logic level low	C-Frame Sync	SPORT7B–4A (FS) SRC4–7 (FS)
SPORT6B–4A (clock, FS, TDV, data) PCG C, D (clock, FS) S/PDIF1 Rx (clock, TDM clock, FS, data) S/PDIF1 Tx (data, block start) PCG A, B from the other DAI (clock, FS)	DAI1 Pin Buffer 20–1 DAI1 Pin Buffer 3 and 4 from other DAI Logic level high Logic level low	D-Pin Buffer Inputs	DAI1 Pin Buffer 20–1 Options: DAI1 Pin Buffer 20–19 Polarity Change
SPORT6B–4A (FS) PCG C (clock) PCG D (clock, FS) S/PDIF1 Tx (block start)	DAI1 Pin Buffer 20–1 Logic level high Logic level low	E-Miscellaneous Signals	***DAI Interrupt 31-22*** MISCA5–0 Options: MISCA5–4 Polarity Change
SPORT7B–4A (clock, FS, data, TDV) MISCA5-0	Logic level high Logic level low	F-Pin Buffer Enable	DAI1 Pin Buffer Enable 20–1

NOTE: For DAI sources (Group A, clocks) and (Group C, FS) only DAI pin buffer 2 through 20 can be used (DAI pin buffer 1 is no longer available and replaced by DAI CRS buffer for the other DAI).

DAI Default Routing

When the processor comes out of reset, the SPORT junctions are bidirectional to the DAI pin buffers. This configuration allows systems to use the SPORTs as either master or slave (without changing the routing scheme). Therefore, programs only need to use the SPORT control register settings to configure master or slave operations. Note that all DAI inputs which are not routed by default are tied to signal low.

NOTE: All DAI input buffers which are not routed by default are driven low and all DAI pin buffer enable signals are driven low.

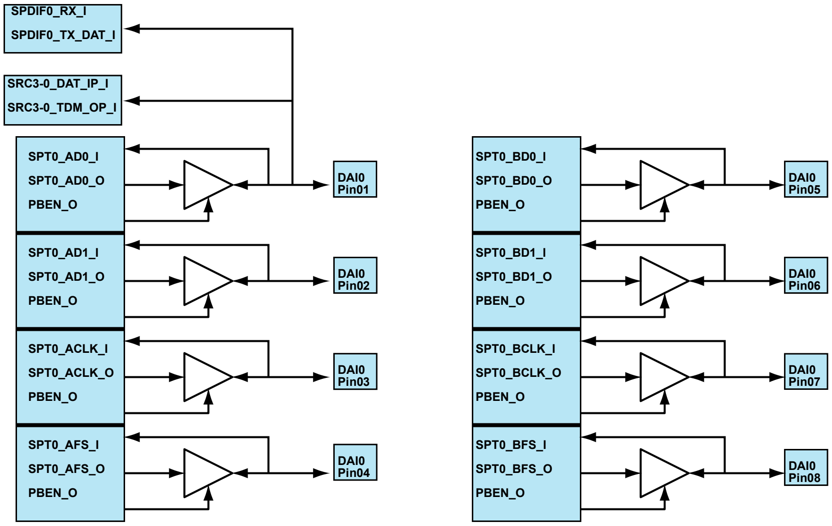

SPDIF0_RX_I
SPDIF0_TX_DAT_I
SRC3-0_DAT_IP_I
SRC3-0_TDM_OP_I
SPT0_AD0_I
SPT0_AD0_O
PBEN_O
DAI0 Pin01
SPT0_AD1_I
SPT0_AD1_O
PBEN_O
DAI0 Pin02
SPT0_ACLK_I
SPT0_ACLK_O
PBEN_O
DAI0 Pin03
SPT0_AFS_I
SPT0_AFS_O
PBEN_O
DAI0 Pin04
SPT0_BD0_I
SPT0_BD0_O
PBEN_O
DAI0 Pin05
SPT0_BD1_I
SPT0_BD1_O
PBEN_O
DAI0 Pin06
SPT0_BCLK_I
SPT0_BCLK_O
PBEN_O
DAI0 Pin07
SPT0_BFS_I
SPT0_BFS_O
PBEN_O
DAI0 Pin08

Figure 33-10: DAI0 Default Routing Pins 01-08

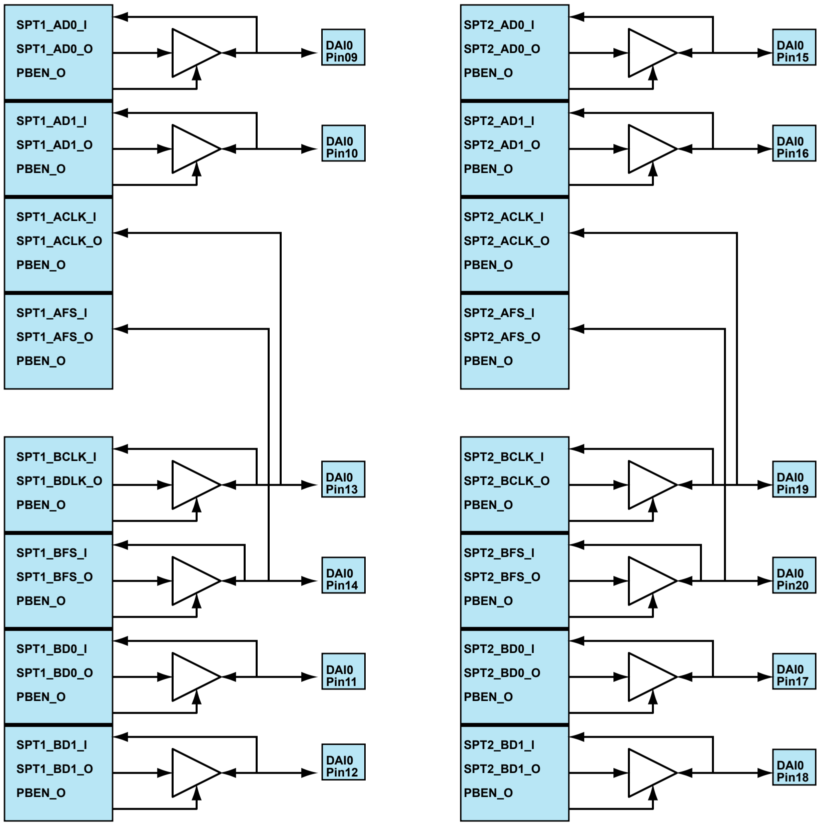

SPT1_AD0_I
SPT1_AD0_O
PBEN_O
DAI0 Pin09
SPT1_AD1_I
SPT1_AD1_O
PBEN_O
DAI0 Pin10
SPT1_ACLK_I
SPT1_ACLK_O
PBEN_O
SPT1_AFS_I
SPT1_AFS_O
PBEN_O
SPT1_BCLK_I
SPT1_BDLK_O
PBEN_O
DAI0 Pin13
SPT1_BFS_I
SPT1_BFS_O
PBEN_O
DAI0 Pin14
SPT1_BD0_I
SPT1_BD0_O
PBEN_O
DAI0 Pin11
SPT1_BD1_I
SPT1_BD1_O
PBEN_O
DAI0 Pin12
SPT2_AD0_I
SPT2_AD0_O
PBEN_O
DAI0 Pin15
SPT2_AD1_I
SPT2_AD1_O
PBEN_O
DAI0 Pin16
SPT2_ACLK_I
SPT2_ACLK_O
PBEN_O
SPT2_AFS_I
SPT2_AFS_O
PBEN_O
SPT2_BCLK_I
SPT2_BCLK_O
PBEN_O
DAI0 Pin19
SPT2_BFS_I
SPT2_BFS_O
PBEN_O
DAI0 Pin20
SPT2_BD0_I
SPT2_BD0_O
PBEN_O
DAI0 Pin17
SPT2_BD1_I
SPT2_BD1_O
PBEN_O
DAI0 Pin18

Figure 33-11: DAI0 Default Routing Pins 09-20

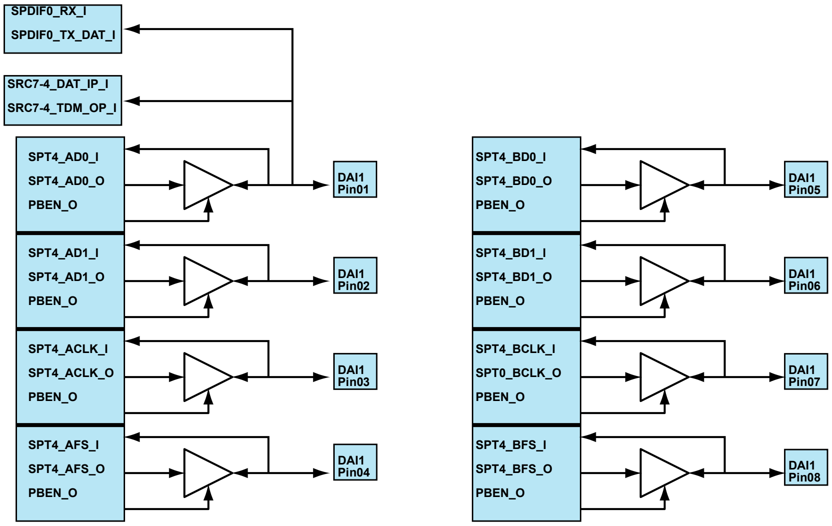

SPDIF0_RX_I
SPDIF0_TX_DAT_I
SRC7-4_DAT_IP_I
SRC7-4_TDM_OP_I
SPT4_AD0_I
SPT4_AD0_O
PBEN_O
DAI1 Pin01
SPT4_AD1_I
SPT4_AD1_O
PBEN_O
DAI1 Pin02
SPT4_ACLK_I
SPT4_ACLK_O
PBEN_O
DAI1 Pin03
SPT4_AFS_I
SPT4_AFS_O
PBEN_O
DAI1 Pin04
SPT4_BD0_I
SPT4_BD0_O
PBEN_O
DAI1 Pin05
SPT4_BD1_I
SPT4_BD1_O
PBEN_O
DAI1 Pin06
SPT4_BCLK_I
SPT0_BCLK_O
PBEN_O
DAI1 Pin07
SPT4_BFS_I
SPT4_BFS_O
PBEN_O
DAI1 Pin08

Figure 33-12: DAI1 Default Routing Pins 01-08

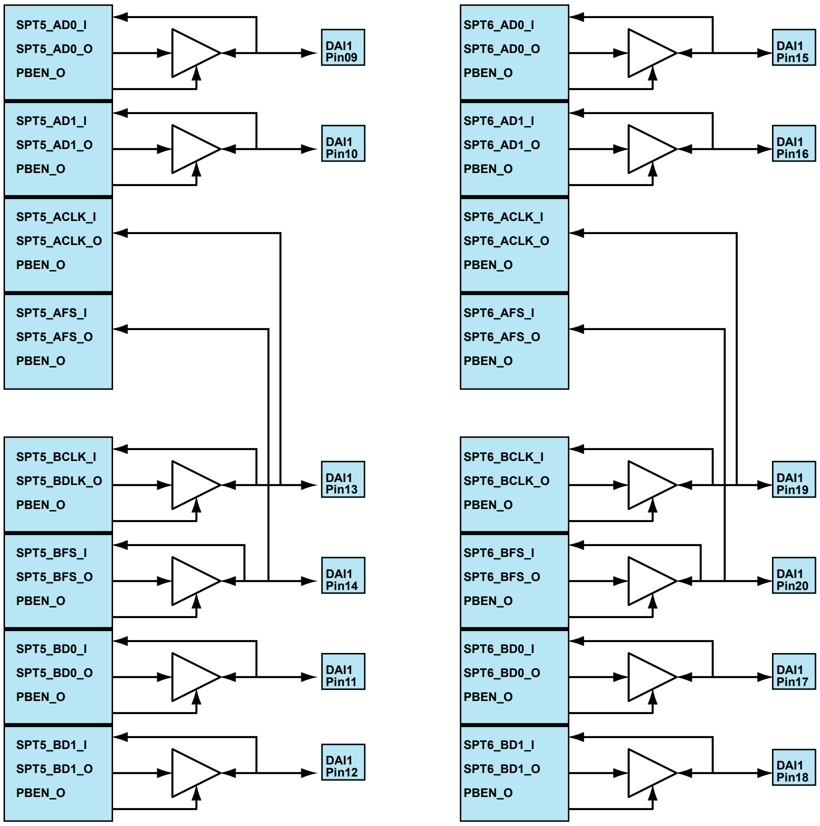

SPT5_AD0_I
SPT5_AD0_O
PBEN_O
DAI1 Pin09
SPT5_AD1_I
SPT5_AD1_O
PBEN_O
DAI1 Pin10
SPT5_ACLK_I
SPT5_ACLK_O
PBEN_O
SPT5_AFS_I
SPT5_AFS_O
PBEN_O
SPT5_BCLK_I
SPT5_BDLK_O
PBEN_O
DAI1 Pin13
SPT5_BFS_I
SPT5_BFS_O
PBEN_O
DAI1 Pin14
SPT5_BD0_I
SPT5_BD0_O
PBEN_O
DAI1 Pin11
SPT5_BD1_I
SPT5_BD1_O
PBEN_O
DAI1 Pin12
SPT6_AD0_I
SPT6_AD0_O
PBEN_O
DAI1 Pin15
SPT6_AD1_I
SPT6_AD1_O
PBEN_O
DAI1 Pin16
SPT6_ACLK_I
SPT6_ACLK_O
PBEN_O
SPT6_AFS_I
SPT6_AFS_O
PBEN_O
SPT6_BCLK_I
SPT6_BCLK_O
PBEN_O
DAI1 Pin19
SPT6_BFS_I
SPT6_BFS_O
PBEN_O
DAI1 Pin20
SPT6_BD0_I
SPT6_BD0_O
PBEN_O
DAI1 Pin17
SPT6_BD1_I
SPT6_BD1_O
PBEN_O
DAI1 Pin18

Figure 33-13: DAI1 Default Routing Pins 09-20

Unused DAI Connections

The SRUs have a default general-purpose routing scheme which can be modified to accomodate any number of different system designs. Regardless of the system design, it is good practice to tie all unused inputs to a high or low level to reduce dynamic power consumption.

Un-Bonded DAI Buffers

Some product variants have some DAI pin buffer not bonded off-chip. Note thee un bonded-DAI buffers can continued to be used for internal signal routing purposes (for example exchange CLK/FS/data between DAI buffers).

DAI Operating Modes

Some buffers allow polarity changes, described as follows.

DAI Pin Buffer Polarity

As shown in the ***Pin Buffer Polarity*** figure, the DAI pin buffer 20-19 can change the polarity of the input signal if the corresponding control bits (`DAI_PIN4.INV20`, `DAI_PIN4.INV19`) are set. These bits can be set during run time and the buffer should not loopback to itself.

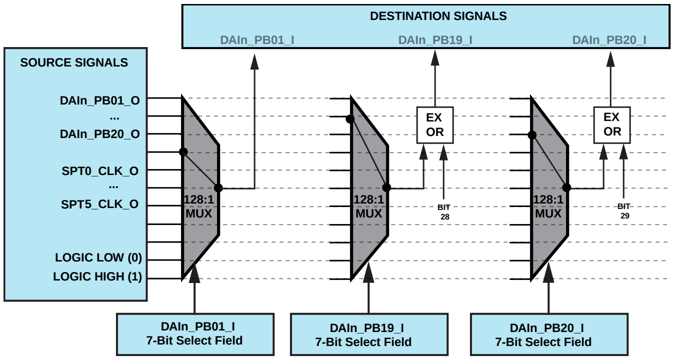

DESTINATION SIGNALS
DAIn_PB01_I
DAIn_PB19_I
DAIn_PB20_I
SOURCE SIGNALS
DAIn_PB01_O
...
DAIn_PB20_O
SPT0_CLK_O
...
SPT5_CLK_O
LOGIC LOW (0)
LOGIC HIGH (1)
128:1 MUX
128:1 MUX
EX OR
BIT 28
128:1 MUX
EX OR
BIT 29
DAIn_PB01_I 7-Bit Select Field
DAIn_PB19_I 7-Bit Select Field
DAIn_PB20_I 7-Bit Select Field

Figure 33-14: Pin Buffer Polarity

DAI Miscellaneous Buffer Polarity

As shown in the ***Pin Buffer Polarity*** figure, the A5-4 miscellaneous buffers can change the polarity of the input signal if the corresponding control bits (`DAI_MISC1.IN5`, `DAI_MISC1.IN4`) are set. Both buffers are not connected to the DAI interrupt latch register. These bits can be set during run time.

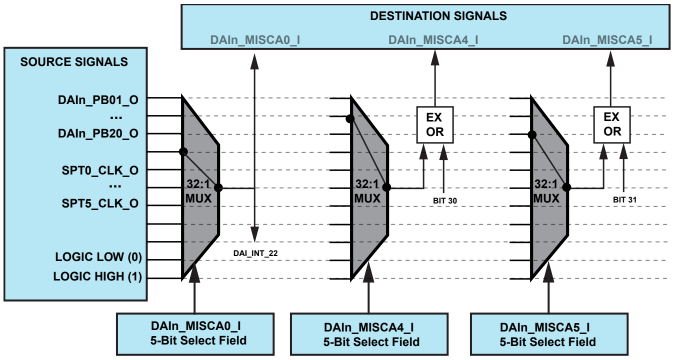

DESTINATION SIGNALS
DAIn_MISCA0_I
DAIn_MISCA4_I
DAIn_MISCA5_I
SOURCE SIGNALS
DAIn_PB01_O
...
DAIn_PB20_O
SPT0_CLK_O
...
SPT5_CLK_O
LOGIC LOW (0)
LOGIC HIGH (1)
32:1
MUX
DAI_INT_22
32:1
MUX
EX
OR
BIT 30
32:1
MUX
EX
OR
BIT 31
DAIn_MISCA0_I
5-Bit Select Field
DAIn_MISCA4_I
5-Bit Select Field
DAIn_MISCA5_I
5-Bit Select Field

Figure 33-15: Miscellaneous Buffer Polarity

DAI System Interrupt Controller (SIC)

The DAI module incorporates a system interrupt controller (SIC) which is connected to the SEC and GIC as seen in the ***DAI System Interrupt Controller*** figure.

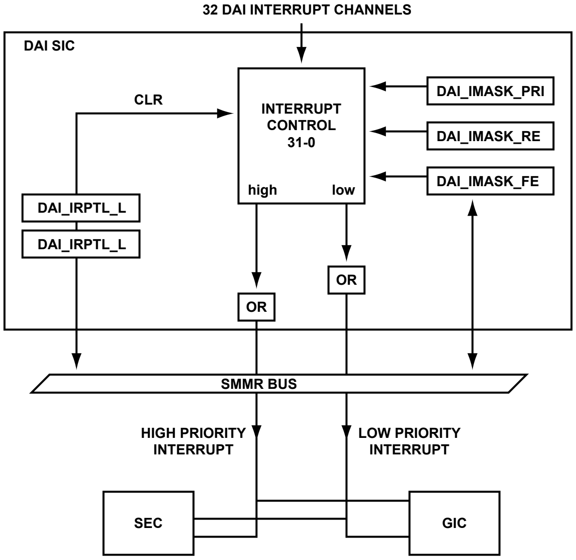

32 DAI INTERRUPT CHANNELS
DAI SIC
CLR
INTERRUPT
CONTROL
31-0
high
low
DAI_IMASK_PRI
DAI_IMASK_RE
DAI_IMASK_FE
DAI_IRPTL_L
DAI_IRPTL_L
OR
OR
SMMR BUS
HIGH PRIORITY
INTERRUPT
LOW PRIORITY
INTERRUPT
SEC
GIC

Figure 33-16: DAI System Interrupt Controller

The DAI has its own system interrupt controllers that indicate to the core when DAI audio peripheral-related events have occurred. Since audio events generally occur infrequently relative to the SHARC core, the DAI interrupt controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems. One interrupt is mapped with DAI low priority. The second interrupt is mapped with DAI high priority. This configuration allows programs to broadly indicate priority. In this way, the DAI SIC provides 32 independently configurable sources or channels. The output bus interrupt signals are logically OR'ed into one interrupt line and fed to the interrupt controller logic of the core.

Three registers are used to configure the DAI interrupt controller. Each of the 32 interrupt sources can be independently configured to trigger on a rising edge, falling edge, both edges, or neither edge of an incoming signal. All DAI interrupt control registers are memory-mapped registers and are accessed through the peripheral bus.

Interrupt Sources

The DAI's five peripheral sources are multiplexed into 32 interrupt sources and are labeled DAI_INT31-0 (*DAI Interrupt Sources* table).

NOTE: There are two naming conventions. The DAI interrupt controller register bits are labeled DAI_31-0_INT. Their corresponding SRU routing signals are labeled DAI_INT_31-0_I.

Table 33-6: DAI Interrupt Sources

Interrupt Source	Description	Signal Response
DAI_INT2–0, DAI_INT4	S/PDIF RX, 4 channels	Event
DAI_INT2–0, DAI_INT4	S/PDIF RX, 4 channels	Waveform
DAI_INT21–18	ASRC, 4 channels	
DAI_INT31–22	Miscellaneous, S/PDIF TX, 9 channels	

Interrupt Latch Priority Option

The `DAI_IMSK_PRI` register specifies the priority for the DAI interrupt channels. DAI system interrupt controller has a pair of interrupt latch registers, `DAI_IRPTL_H` and `DAI_IRPTL_L`. The configuration of the `DAI_IMSK_PRI` register also determines the interrupt latch mapping for a particular DAI interrupt. When a DAI interrupt is configured as low priority (`DAI_IMSK_PRI` bit cleared, default setting), interrupts are mapped to the DAI_INTR_IRQL signal and when an interrupt occurs, the corresponding bit of the `DAI_IRPTL_L` register is set. When a DAI interrupt is configured as high priority (`DAI_IMSK_PRI` bit set), interrupts are mapped to the DAI_INTR_IRQH signal and the interrupt is latched to the `DAI_IRPTL_H` register. The low priority DAI interrupt (INTR_DAI_IRQL) and high priority DAI interrupt (INTR_DAI_IRQH) are connected to the SEC and GIC.

Interrupt Mask for Waveforms

The `DAI_IMSK_RE` and `DAI_IMSK_FE` registers allow programs to mask or unmask interrupts for specific edges of a signal mapped to the channel. It can be configured for rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges by masking them separately. Signals from the SRU can be used to generate interrupts. For example, when the `DAI_IMSK_FE.MISCINT9` bit is set to one, any falling edge signals from the external channel generate an interrupt and the interrupt latch is set.

Interrupt Mask for Events

The system interrupt controller needs information about a peripheral's interrupt sources that correspond to event signals (see the ***DAI Interrupt Sources*** table). As a result, the rising edge is used as an interrupt source only. For DAI peripherals marked as events, programs may unmask an interrupt source on the rising edge only.

Shadow Interrupt Register

The DAI interrupt controller has shadow registers to simplify debug activities since these registers do not manipulate status control. Any read of the shadow registers (`DAI_IRPTL_HS`, `DAI_IRPTL_LS`), provides the same data as a read of the `DAI_IRPTL_H` and `DAI_IRPTL_L` registers. However, reads of the DAI shadow registers do not change the interrupt acknowledge status to the core interrupt controller.

Interrupt Service

The interrupt acknowledge operates differently when multiple channels are multiplexed into one interrupt output signal. When an interrupt from the DAI must be serviced, any of the two interrupt service routines (INTR_DAI_IRQL, INTR_DAI_IRQH) must query the SIC to determine the source(s). Sources can be any one or more of the DAI channels (DAI_INT31-0).

- When the `DAI_IRPTL_H` register is read, the high priority latched interrupts are cleared.
- When the `DAI_IRPTL_L` register is read, the low priority latched interrupts are cleared.

If an interrupt occurs in the same cycle as a latch register is cleared, the clear mechanism has lower priority and the new interrupt is registered.

Signal Routing Unit Effect Latency

After the DAIy registers are configured the effect latency is 2 *SCLK0* cycles minimum and 3 *SCLK0* cycles maximum.

DAI Programming Model

As discussed in the previous sections, the signal routing unit is controlled by writing values that provide a plug-in tool in CCES so that configuring the SRU is done graphically. Analog Devices offers macros that are included with the CrossCore or VisualDSP++ tools, greatly easing code development in the SRU.

There is a macro that has been created to connect peripherals used in a DAI configuration. This code can be used in both assembly and C code. See the INCLUDE file SRU.H. In practice the macro is provided and forms the style `SRU(source_O, destination_I)` for DAI0 and `SRU2(source_O, destination_I)` for DAI1. Example appears as: `SRU(DAI0_PB12_O, SPT0_ACLK_I); or SRU2(DAI1_PB14_O, SPT4_ACLK_I);`

Debug Features

The following section describes the feature that can be used to help in debugging the DAI.

Loopback Routing

The SPORT serial peripheral supports an internal loopback mode. If the loopback bit for each peripheral is enabled, it connects the transmitter with the receiver block internally (does not signal off-chip). The SRU can be used for this purpose. The ***Loopback Routing*** table describes the different possible routings based on the peripheral.

NOTE: The peripheral's loopback mode for debug is independent from both of the signal routing units.

Table 33-7: Loopback Routing

Peripheral	Loopback Mode	SRU1–0 Internal Routing for Loopback	SRU1–0 External Routing for Loopback
SPORT	Yes	SPTx_xx_O → SPTx_xx_I	SPTx_xx_O → DAIn_PBxx_I DAIn_PBxx_O → SPTx_xx_I
S/PDIF Tx/Rx	No	SPDIFn_TX_O → SPDIFn_RX_I	SPDIFn_TX_O → DAI_PBxx_I DAI_PBxx_O → SPDIFn_RX_I
SRC	No	SRCx_DAT_OP_O → SRCx_DAT_IP_I	SRCx_DAT_OP_O → DAI_PBxx_I DAI_PBxx_O → SRCx_DAT_IP_I

DAI Sources Overview

The following tables provide information of the various DAI sources sorted by group. The shaded cells in the tables denote connection to the other DAI.

DAI0 Group A – Clock Routing Source Signals

The group A clock sources are listed in the following table. The Group A clock destinations are configured using Clock Routing Control Register 0 through Clock Routing Control Register 5 .

Table 33-8: DAI0 Group A – Clock Routing

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_CRS_PB03_O	Pin Buffer 3 (DAI1)
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9

Table 33-8: DAI0 Group A – Clock Routing (Continued)

Selection Code	Source Signal	Description (Source Selection)
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12
01100 (0xC)	DAI0_PB13_O	Pin Buffer 13
01101 (0xD)	DAI0_PB14_O	Pin Buffer 14
01110 (0xE)	DAI0_PB15_O	Pin Buffer 15
01111 (0xF)	DAI0_PB16_O	Pin Buffer 16
10000 (0x10)	DAI0_PB17_O	Pin Buffer 17
10001 (0x11)	DAI0_PB18_O	Pin Buffer 18
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20
10100 (0x14)	SPT0_ACLK_O	SPORT 0 Clock A
10101 (0x15)	SPT0_BCLK_O	SPORT 0 Clock B
10110 (0x16)	SPT1_ACLK_O	SPORT 1 Clock A
10111 (0x17)	SPT1_BCLK_O	SPORT 1 Clock B
11000 (0x18)	SPT2_ACLK_O	SPORT 2 Clock A
11001 (0x19)	SPT2_BCLK_O	SPORT 2 Clock B
11010 (0x1A)	SPDIF0_RX_CLK_O	SPDIF 0 Receive Clock Output
11011 (0x1B)	SPDIF0_RX_TDMCLK_O	SPDIF 0 Receive TDM Clock Output
11100 (0x1C)	PCG0_CLKA_O	Precision Clock A Output
11101 (0x1D)	PCG0_CLKB_O	Precision Clock B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI0 Group B – Serial Data Source Signals

The group B data sources are listed in the following table. The group B data destinations are configured using Serial Data Routing Control Register 0 through Serial Data Routing Control Register 6 .

Table 33-9: DAI0 Group B – Serial Data Signals

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI0_PB01_O	Pin Buffer 1
000001 (0x1)	DAI0_PB02_O	Pin Buffer 2

Table 33-9: DAI0 Group B – Serial Data Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
000010 (0x2)	DAI0_PB03_O	Pin Buffer 3
000011 (0x3)	DAI0_PB04_O	Pin Buffer 4
000100 (0x4)	DAI0_PB05_O	Pin Buffer 5
000101 (0x5)	DAI0_PB06_O	Pin Buffer 6
000110 (0x6)	DAI0_PB07_O	Pin Buffer 7
000111 (0x7)	DAI0_PB08_O	Pin Buffer 8
001000 (0x8)	DAI0_PB09_O	Pin Buffer 9
001001 (0x9)	DAI0_PB10_O	Pin Buffer 10
001010 (0xA)	DAI0_PB11_O	Pin Buffer 11
001011 (0xB)	DAI0_PB12_O	Pin Buffer 12
001100 (0xC)	DAI0_PB13_O	Pin Buffer 13
001101 (0xD)	DAI0_PB14_O	Pin Buffer 14
001110 (0xE)	DAI0_PB15_O	Pin Buffer 15
001111 (0xF)	DAI0_PB16_O	Pin Buffer 16
010000 (0x10)	DAI0_PB17_O	Pin Buffer 17
010001 (0x11)	DAI0_PB18_O	Pin Buffer 18
010010 (0x12)	DAI0_PB19_O	Pin Buffer 19
010011 (0x13)	DAI0_PB20_O	Pin Buffer 20
010100 (0x14)	SPT0_AD0_O	SPORT 0 Data AD0
010101 (0x15)	SPT0_AD1_O	SPORT 0 Data AD1
010110 (0x16)	SPT0_BD0_O	SPORT 0 Data BD0
010111 (0x17)	SPT0_BD1_O	SPORT 0 Data BD1
011000 (0x18)	SPT1_AD0_O	SPORT 1 Data AD0
011001 (0x19)	SPT1_AD1_O	SPORT 1 Data AD1
011010 (0x1A)	SPT1_BD0_O	SPORT 1 Data BD0
011011 (0x1B)	SPT1_BD1_O	SPORT 1 Data BD1
011100 (0x1C)	SPT2_AD0_O	SPORT 2 Data AD0
011101 (0x1D)	SPT2_AD1_O	SPORT 2 Data AD1
011110 (0x1E)	SPT2_BD0_O	SPORT 2 Data BD0
011111 (0x1F)	SPT2_BD1_O	SPORT 2 Data BD1
100000 (0x20)	SRC0_DAT_OP_O	SRC0 Data Out

Table 33-9: DAI0 Group B – Serial Data Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
100001 (0x21)	SRC1_DAT_OP_O	SRC1 Data Out
100010 (0x22)	SRC2_DAT_OP_O	SRC2 Data Out
100011 (0x23)	SRC3_DAT_OP_O	SRC3 Data Out
100100 (0x24)	SRC0_TDM_IP_O	SRC0 Data Out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 Data Out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 Data Out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 Data Out
101000 (0x28)	SPDIF0_RX_DAT_O	SPDIF 0 RX Serial Data Out
101100(0x2C)	SPT3_AD0_O	SPORT 3 Data AD0
101101(0x2D)	SPT3_AD1_O	SPORT 3 Data AD1
101110(0x2E)	SPT3_BD0_O	SPORT 3 Data BD0
101111(0x2F)	SPT3_BD1_O	SPORT 3 Data BD1
110000(0x30)	SPDIF0_TX_O	SPDIF 0 TX Biphase Stream
110001(0x31)	SRC7_CRS_DAT_OP_O	SRC7 data out (DAI1)
110010(0x32)	SRC7_CRS_TDM_IP_O	SRC7 data out (DAI1)
110001(0x33)–111101(0x3D)	Reserved	Reserved
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

DAI0 Group C – Frame Sync Source Signals

The group C frame sync signal sources are listed in the following table. The frame sync destinations are configured using Frame Sync Routing Control Register 0 through Frame Sync Routing Control Register 4 .

Table 33-10: DAI0 Group C – Frame Sync Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_CRS_PB04_O	Pin Buffer 4 (DAI1)
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7

Table 33-10: DAI0 Group C – Frame Sync Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12
01100 (0xC)	DAI0_PB13_O	Pin Buffer 13
01101 (0xD)	DAI0_PB14_O	Pin Buffer 14
01110 (0xE)	DAI0_PB15_O	Pin Buffer 15
01111 (0xF)	DAI0_PB16_O	Pin Buffer 16
10000 (0x10)	DAI0_PB17_O	Pin Buffer 17
10001 (0x11)	DAI0_PB18_O	Pin Buffer 18
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20
10100 (0x14)	SPT0_AFS_O	SPORT 0 Frame Sync A
10101 (0x15)	SPT0_BFS_O	SPORT 0 Frame Sync B
10110 (0x16)	SPT1_AFS_O	SPORT 1 Frame Sync A
10111 (0x17)	SPT1_BFS_O	SPORT 1 Frame Sync B
11000 (0x18)	SPT2_AFS_O	SPORT 2 Frame Sync A
11001 (0x19)	SPT2_BFS_O	SPORT 2 Frame Sync B
11010 (0x1A)	SPDIF0_FS_O	SPDIF 0 RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG0_FSA_O	Precision Frame Sync A Output
11101 (0x1D)	PCG0_FSB_O	Precision Frame Sync B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI0 Group D – Pin Signal Assignment Source Signals

The group D pin signal assignment sources are listed in the following table. The group D destinations are configured using Pin Buffer Assignment Register 0 through Pin Buffer Assignment Register 4 .

Table 33-11: DAI0 Group D – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI0_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI0_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI0_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI0_PB04_O	Pin Buffer 4
0000100 (0x4)	DAI0_PB05_O	Pin Buffer 5
0000101 (0x5)	DAI0_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI0_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI0_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI0_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI0_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI0_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI0_PB12_O	Pin Buffer 12
0001100 (0xC)	DAI0_PB13_O	Pin Buffer 13
0001101 (0xD)	DAI0_PB14_O	Pin Buffer 14
0001110 (0xE)	DAI0_PB15_O	Pin Buffer 15
0001111 (0xF)	DAI0_PB16_O	Pin Buffer 16
0010000 (0x10)	DAI0_PB17_O	Pin Buffer 17
0010001 (0x11)	DAI0_PB18_O	Pin Buffer 18
0010010 (0x12)	DAI0_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI0_PB20_O	Pin Buffer 20
0010100 (0x14)	SPT0_AD0_O	SPORT 0 Data AD0
0010101 (0x15)	SPT0_AD1_O	SPORT 0 Data AD1
0010110 (0x16)	SPT0_BD0_O	SPORT 0 Data BD0
0010111 (0x17)	SPT0_BD1_O	SPORT 0 Data BD1
0011000 (0x18)	SPT1_AD0_O	SPORT 1 Data AD0
0011001 (0x19)	SPT1_AD1_O	SPORT 1 Data AD1
0011010 (0x1A)	SPT1_BD0_O	SPORT 1 Data BD0
0011011 (0x1B)	SPT1_BD1_O	SPORT 1 Data BD1
0011100 (0x1C)	SPT2_AD0_O	SPORT 2 Data AD0
0011101 (0x1D)	SPT2_AD1_O	SPORT 2 Data AD1
0011110 (0x1E)	SPT2_BD0_O	SPORT 2 Data BD0

Table 33-11: DAI0 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0011111 (0x1F)	SPT2_BD1_O	SPORT 2 Data BD1
0100000 (0x20)	SPT0_ACLK_O	SPORT 0 Clock A
0100001 (0x21)	SPT0_BCLK_O	SPORT 0 Clock B
0100010 (0x22)	SPT1_ACLK_O	SPORT 1 Clock A
0100011 (0x23)	SPT1_BCLK_O	SPORT 1 Clock B
0100100 (0x24)	SPT2_ACLK_O	SPORT 2 Clock A
0100101 (0x25)	SPT2_BCLK_O	SPORT 2 Clock B
0100110 (0x26)	SPT0_AFS_O	SPORT 0 Frame Sync A
0100111 (0x27)	SPT0_BFS_O	SPORT 0 Frame Sync B
0101000 (0x28)	SPT1_AFS_O	SPORT 1 Frame Sync A
0101001 (0x29)	SPT1_BFS_O	SPORT 1 Frame Sync B
0101010 (0x2A)	SPT2_AFS_O	SPORT 2 Frame Sync A
0101011 (0x2B)	SPT2_BFS_O	SPORT 2 Frame Sync B
0101100 (0x2C)	SPT3_AD0_O	SPORT 3 Data AD0
0101101 (0x2D)	SPT3_AD1_O	SPORT 3 Data AD1
0101110 (0x2E)	SPT3_BD0_O	SPORT 3 Data BD0
0101111 (0x2F)	SPT3_BD1_O	SPORT 3 Data BD1
0110000 (0x30)	MLB0_CLKOUT	MLB PLL clock output
0110001 (0x31)	SPDIF0_TX_BLKSTART_O	SPDIF 0 TX Block Start Output
0110100 (0x34)	SPT3_ACLK_O	SPORT 3 Clock A
0110101 (0x35)	SPT3_BCLK_O	SPORT 3 Clock B
0110110 (0x36)	SPT3_AFS_O	SPORT 3 Frame Sync A
0110111 (0x37)	SPT3_BFS_O	SPORT 3 Frame Sync B
0111000 (0x38)	PCG0_CLKA_O	Precision Clock A
0111001 (0x39)	PCG0_CLKB_O	Precision Clock B
0111010 (0x3A)	PCG0_FSA_O	Precision Frame Sync A
0111011 (0x3B)	PCG0_FSB_O	Precision Frame Sync B
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 Data Output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 Data Output
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 Data Output

Table 33-11: DAI0 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 Data Output
1000001 (0x41)	SPDIF0_RX_DAT_O	SPDIF 0 RX Data Output
1000010 (0x42)	SPDIF0_FS_O	SPDIF 0 RX Frame Sync Output
1000011 (0x43)	SPDIF0_RXCLK_O	SPDIF 0 RX Clock Output
1000100 (0x44)	SPDIF0_RX_TDMCLK_O	SPDIF 0 RX TDM Clock Output
1000101 (0x45)	SPDIF0_TX_O	SPDIF 0 TX Biphase Encoded Data Output
1000110 (0x46)	SPT0_ATDV_O	SPORT0 Transmit A Data Valid Output
1000111 (0x47)	SPT0_BTDV_O	SPORT0 Transmit B Data Valid Output
1001000 (0x48)	SPT1_ATDV_O	SPORT1 Transmit A Data Valid Output
1001001 (0x49)	SPT1_BTDV_O	SPORT1 Transmit B Data Valid Output
1001010 (0x4A)	SPT2_ATDV_O	SPORT2 Transmit A Data Valid Output
1001011 (0x4B)	SPT2_BTDV_O	SPORT2 Transmit B Data Valid Output
1001100 (0x4C)	SPT3_ATDV_O	SPORT3 Transmit A Data Valid Output
1001101 (0x4D)	SPT3_BTDV_O	SPORT3 Transmit B Data Valid Output
1001110 (0x4E)	Reserved	
1001111 (0x4F)	Reserved	
1010000 (0x50)	PCG0_CRS_CLKC_O	Precision Clock C (DAI1)
1010001 (0x51)	PCG0_CRS_CLKD_O	Precision Clock D (DAI1)
1010010 (0x52)	PCG0_CRS_FSC_O	Precision Frame Sync C (DAI1)
1010011 (0x53)	PCG0_CRS_FSD_O	Precision Frame Sync D (DAI1)
1010100 (0x54)	DAI0_CRS_PB03_O	Pin Buffer 3 (DAI1)
1010101 (0x55)	DAI0_CRS_PB04_O	Pin Buffer 4 (DAI1)
1010110 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)
1111111 (0x7F)	HIGH	Logic Level High (1)

DAI0 Group E – Miscellaneous Source Signals

The group E miscellaneous signal sources are listed in the following table. The group E destinations are configured using Miscellaneous Control Register 0 through Miscellaneous Control Register 1 .

Table 33-12: DAI0 Group E – Miscellaneous Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI0_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI0_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI0_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI0_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI0_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI0_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI0_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI0_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI0_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI0_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI0_PB11_O	Pin Buffer 11 Output
01011 (0xB)	DAI0_PB12_O	Pin Buffer 12 Output
01100 (0xC)	DAI0_PB13_O	Pin Buffer 13 Output
01101 (0xD)	DAI0_PB14_O	Pin Buffer 14 Output
01110 (0xE)	DAI0_PB15_O	Pin Buffer 15 Output
01111 (0xF)	DAI0_PB16_O	Pin Buffer 16 Output
10000 (0x10)	DAI0_PB17_O	Pin Buffer 17 Output
10001 (0x11)	DAI0_PB18_O	Pin Buffer 18 Output
10010 (0x12)	DAI0_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI0_PB20_O	Pin Buffer 20 Output
10100 (0x14)	SPT0_AFS_O	SPORT 0 Frame Sync A
10101 (0x15)	SPT0_BFS_O	SPORT 0 Frame Sync B
10110 (0x16)	SPT1_AFS_O	SPORT 1 Frame Sync A
10111 (0x17)	SPT1_BFS_O	SPORT 1 Frame Sync B
11000 (0x18)	SPT2_AFS_O	SPORT 2 Frame Sync A
11001 (0x19)	SPT2_BFS_O	SPORT 2 Frame Sync B
11010 (0x1A)	SPDIF0_TX_BLKSTART_O	SPDIF 0 TX Block Start Output
11011 (0x1B)	PCG0_FSA_O	Precision Frame Sync A
11100 (0x1C)0	PCG0_CLKB_O	Precision Clock B
11101 (0x1D)	PCG0_FSB_O	Precision Frame Sync B
11110 (0x1E)	LOW	Logic Level Low (0) as a Source

Table 33-12: DAI0 Group E – Miscellaneous Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

DAI0 Group F – Pin Output Enable Source Signals

The group F pin output enable signal sources are listed in the following table. The group F destinations are configured using Pin Buffer Enable Register 0 through Pin Buffer Enable Register 3 .

Table 33-13: DAI0 Group F – Pin Output Enable

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DAI0_MISCA0_O	DAI0 Miscellaneous Control A0 Output
000011 (0x3)	DAI0_MISCA1_O	DAI0 Miscellaneous Control A1 Output
000100 (0x4)	DAI0_MISCA2_O	DAI0 Miscellaneous Control A2 Output
000101 (0x5)	DAI0_MISCA3_O	DAI0 Miscellaneous Control A3 Output
000110 (0x6)	DAI0_MISCA4_O	DAI0 Miscellaneous Control A4 Output
000111 (0x7)	DAI0_MISCA5_O	DAI0 Miscellaneous Control A5 Output
001000 (0x8)	SPT0_ACLK_PBEN_O	SPORT 0 Clock A Output Enable
001001 (0x9)	SPT0_AFS_PBEN_O	SPORT 0 Frame Sync A Output Enable
001010 (0xA)	SPT0_AD0_PBEN_O	SPORT 0 Data AD0 Output Enable
001011 (0xB)	SPT0_AD1_PBEN_O	SPORT 0 Data AD1 Output Enable
001100 (0xC)	SPT0_BCLK_PBEN_O	SPORT 0 Clock B Output Enable
001101 (0xD)	SPT0_BFS_PBEN_O	SPORT 0 Frame Sync B Output Enable
001110 (0xE)	SPT0_BD0_PBEN_O	SPORT 0 Data BD0 Output Enable
001111 (0xF)	SPT0_BD1_PBEN_O	SPORT 0 Data BD1 Output Enable
010000 (0x10)	SPT1_ACLK_PBEN_O	SPORT 1 Clock A Output Enable
010001 (0x11)	SPT1_AFS_PBEN_O	SPORT 1 Frame Sync A Output Enable
010010 (0x12)	SPT1_AD0_PBEN_O	SPORT 1 Data AD0 Output Enable
010011 (0x13)	SPT1_AD1_PBEN_O	SPORT 1 Data AD1 Output Enable
010100 (0x14)	SPT1_BCLK_PBEN_O	SPORT 1 Clock B Output Enable
010101 (0x15)	SPT1_BFS_PBEN_O	SPORT 1 Frame Sync B Output Enable
010110 (0x16)	SPT1_BD0_PBEN_O	SPORT 1 Data BD0 Output Enable
010111 (0x17)	SPT1_BD1_PBEN_O	SPORT 1 Data BD1 Output Enable

Table 33-13: DAI0 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
011000 (0x18)	SPT2_ACLK_PBEN_O	SPORT 2 Clock A Output Enable
011001 (0x19)	SPT2_AFS_PBEN_O	SPORT 2 Frame Sync A Output Enable
011010 (0x1A)	SPT2_AD0_PBEN_O	SPORT 2 Data AD0 Output Enable
011011 (0x1B)	SPT2_AD1_PBEN_O	SPORT 2 Data AD1 Output Enable
011100 (0x1C)	SPT2_BCLK_PBEN_O	SPORT 2 Clock B Output Enable
011101 (0x1D)	SPT2_BFS_PBEN_O	SPORT 2 Frame Sync B Output Enable
011110 (0x1E)	SPT2_BD0_PBEN_O	SPORT 2 Data BD0 Output Enable
011111 (0x1F)	SPT2_BD1_PBEN_O	SPORT 2 Data BD1 Output Enable
100000 (0x20)	SPT3_ACLK_PBEN_O	SPORT 3 Clock A Output Enable
100001 (0x21)	SPT3_AFS_PBEN_O	SPORT 3 Frame Sync A Output Enable
100010 (0x22)	SPT3_AD0_PBEN_O	SPORT 3 Data AD0 Output Enable
100011 (0x23)	SPT3_AD1_PBEN_O	SPORT 3 Data AD1 Output Enable
100100 (0x24)	SPT3_BCLK_PBEN_O	SPORT 2 Clock B Output Enable
100101 (0x25)	SPT3_BFS_PBEN_O	SPORT 3 Frame Sync B Output Enable
100110 (0x26)	SPT3_BD0_PBEN_O	SPORT 3 Data BD0 Output Enable
100111 (0x27)	SPT3_BD1_PBEN_O	SPORT 3 Data BD1 Output Enable
101000 (0x28)	SPT0_ATDV_PBEN_O	SPORT 0 A Transmit Data Valid Output
101001 (0x29)	SPT0_BTDV_PBEN_O	SPORT 0 B Transmit Data Valid Output
101010 (0x2A)	SPT1_ATDV_PBEN_O	SPORT 1 A Transmit Data Valid Output
101011 (0x2B)	SPT1_BTDV_PBEN_O	SPORT 1 B Transmit Data Valid Output
101100 (0x2C)	SPT2_ATDV_PBEN_O	SPORT 2 A Transmit Data Valid Output
101101 (0x2D)	SPT2_BTDV_PBEN_O	SPORT 2 B Transmit Data Valid Output
100111 (0x2E)	SPT3_ATDV_PBEN_O	SPORT 3 A Transmit Data Valid Output
101110 (0x2F)	SPT3_BTDV_PBEN_O	SPORT 3 B Transmit Data Valid Output
110000 (0x30)–1111111 (0x3F)	Reserved	

DAI1 Group A – Clock Routing Source Signals

The group A clock routing signal sources are listed in the following table. The Group A clock destinations are configured using Clock Routing Control Register 0 through Clock Routing Control Register 5 .

Table 33-14: DAI1 Group A – Clock Routing Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI1_CRS_PB03_O	Pin Buffer 3 (DAI0)
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12
01100 (0xC)	DAI1_PB13_O	Pin Buffer 13
01101 (0xD)	DAI1_PB14_O	Pin Buffer 14
01110 (0xE)	DAI1_PB15_O	Pin Buffer 15
01111 (0xF)	DAI1_PB16_O	Pin Buffer 16
10000 (0x10)	DAI1_PB17_O	Pin Buffer 17
10001 (0x11)	DAI1_PB18_O	Pin Buffer 18
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20
10100 (0x14)	SPT4_ACLK_O	SPORT 4 Clock A
10101 (0x15)	SPT4_BCLK_O	SPORT 4 Clock B
10110 (0x16)	SPT5_ACLK_O	SPORT 5 Clock A
10111 (0x17)	SPT5_BCLK_O	SPORT 5 Clock B
11000 (0x18)	SPT6_ACLK_O	SPORT 6 Clock A
11001 (0x19)	SPT6_BCLK_O	SPORT 6 Clock B
11010 (0x1A)	SPDIF1_RX_CLK_O	SPDIF 1 Receive Clock Output
11011 (0x1B)	SPDIF1_RX_TDMCLK_O	SPDIF 1 Receive TDM Clock Output
11100 (0x1C)	PCG0_CLKC_O	Precision Clock C Output
11101 (0x1D)	PCG0_CLKD_O	Precision Clock D Output
11110 (0x1E)	LOW	Logic Level Low (0)

Table 33-14: DAI1 Group A – Clock Routing Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI1 Group B – Serial Data Source Signals

The group B data sources are listed in the following table. The group B data destinations are configured using Serial Data Routing Control Register 0 through Serial Data Routing Control Register 6 .

Table 33-15: DAI1 Group B – Serial Data Signals

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI1_PB01_O	Pin Buffer 1
000001 (0x1)	DAI1_PB02_O	Pin Buffer 2
000010 (0x2)	DAI1_PB03_O	Pin Buffer 3
000011 (0x3)	DAI1_PB04_O	Pin Buffer 4
000100 (0x4)	DAI1_PB05_O	Pin Buffer 5
000101 (0x5)	DAI1_PB06_O	Pin Buffer 6
000110 (0x6)	DAI1_PB07_O	Pin Buffer 7
000111 (0x7)	DAI1_PB08_O	Pin Buffer 8
001000 (0x8)	DAI1_PB09_O	Pin Buffer 9
001001 (0x9)	DAI1_PB10_O	Pin Buffer 10
001010 (0xA)	DAI1_PB11_O	Pin Buffer 11
001011 (0xB)	DAI1_PB12_O	Pin Buffer 12
001100 (0xC)	DAI1_PB13_O	Pin Buffer 13
001101 (0xD)	DAI1_PB14_O	Pin Buffer 14
001110 (0xE)	DAI1_PB15_O	Pin Buffer 15
001111 (0xF)	DAI1_PB16_O	Pin Buffer 16
010000 (0x10)	DAI1_PB17_O	Pin Buffer 17
010001 (0x11)	DAI1_PB18_O	Pin Buffer 18
010010 (0x12)	DAI1_PB19_O	Pin Buffer 19
010011 (0x13)	DAI1_PB20_O	Pin Buffer 20
010100 (0x14)	SPT4_AD0_O	SPORT 4 Data AD0
010101 (0x15)	SPT4_AD1_O	SPORT 4 Data AD1
010110 (0x16)	SPT4_BD0_O	SPORT 4 Data BD0
010111 (0x17)	SPT4_BD1_O	SPORT 4 Data BD1

Table 33-15: DAI1 Group B – Serial Data Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
011000 (0x18)	SPT5_AD0_O	SPORT 5 Data AD0
011001 (0x19)	SPT5_AD1_O	SPORT 5 Data AD1
011010 (0x1A)	SPT5_BD0_O	SPORT 5 Data BD0
011011 (0x1B)	SPT5_BD1_O	SPORT 5 Data BD1
011100 (0x1C)	SPT6_AD0_O	SPORT 6 Data AD0
011101 (0x1D)	SPT6_AD1_O	SPORT 6 Data AD1
011110 (0x1E)	SPT6_BD0_O	SPORT 6 Data BD0
011111 (0x1F)	SPT6_BD1_O	SPORT 6 Data BD1
100000 (0x20)	SRC4_DAT_OP_O	SRC4 Data Out
100001 (0x21)	SRC5_DAT_OP_O	SRC5 Data Out
100010 (0x22)	SRC6_DAT_OP_O	SRC6 Data Out
100011 (0x23)	SRC7_DAT_OP_O	SRC7 Data Out
100100 (0x24)	SRC4_TDM_IP_O	SRC4 Data Out
100101 (0x25)	SRC5_TDM_IP_O	SRC5 Data Out
100110 (0x26)	SRC6_TDM_IP_O	SRC6 Data Out
100111 (0x27)	SRC7_TDM_IP_O	SRC7 Data Out
101000 (0x28)	SPDIF1_RX_DAT_O	SPDIF 1 RX Serial Data Out
101100(0x2C)	SPT7_AD0_O	SPORT 7 Data AD0
101101(0x2D)	SPT7_AD1_O	SPORT 7 Data AD1
101110(0x2E)	SPT7_BD0_O	SPORT 7 Data BD0
101111(0x2F)	SPT7_BD1_O	SPORT 7 Data BD1
110000(0x30)	SPDIF1_TX_O	SPDIF 1 TX Biphase Stream
110001(0x31)	SRC3_CRS_DAT_OP_O	SRC3 data out (DAI0)
110010(0x32)	SRC3_CRS_TDM_IP_O	SRC3 data out (DAI0)
110001(0x33)–111101(0x3D)	Reserved	
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

DAI1 Group C – Frame Sync Source Signals

The group C frame sync signals are listed in the following table. The frame sync destinations are configured using Frame Sync Routing Control Register 0 through Frame Sync Routing Control Register 4 .

Table 33-16: DAI1 Group C – Frame Sync Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI1_CRS_PB04_O	Pin Buffer 4 (DAI0)
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12
01100 (0xC)	DAI1_PB13_O	Pin Buffer 13
01101 (0xD)	DAI1_PB14_O	Pin Buffer 14
01110 (0xE)	DAI1_PB15_O	Pin Buffer 15
01111 (0xF)	DAI1_PB16_O	Pin Buffer 16
10000 (0x10)	DAI1_PB17_O	Pin Buffer 17
10001 (0x11)	DAI1_PB18_O	Pin Buffer 18
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20
10100 (0x14)	SPT4_AFS_O	SPORT 4 Frame Sync A
10101 (0x15)	SPT4_BFS_O	SPORT 4 Frame Sync B
10110 (0x16)	SPT5_AFS_O	SPORT 5 Frame Sync A
10111 (0x17)	SPT5_BFS_O	SPORT 5 Frame Sync B
11000 (0x18)	SPT6_AFS_O	SPORT 6 Frame Sync A
11001 (0x19)	SPT6_BFS_O	SPORT 6 Frame Sync B
11010 (0x1A)	SPDIF1_FS_O	SPDIF 1 RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG0_FSC_O	Precision Frame Sync C Output
11101 (0x1D)	PCG0_FSD_O	Precision Frame Sync D Output
11110 (0x1E)	LOW	Logic Level Low (0)

Table 33-16: DAI1 Group C – Frame Sync Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
11111 (0x1F)	HIGH	Logic Level High (1)

DAI1 Group D – Pin Signal Assignment Source Signals

The group D pin signal assignments are listed in the following table. The group D destinations are configured using Pin Buffer Assignment Register 0 through Pin Buffer Assignment Register 4 .

Table 33-17: DAI1 Group D – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI1_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI1_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI1_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI1_PB04_O	Pin Buffer 4
0000100 (0x4)	DAI1_PB05_O	Pin Buffer 5
0000101 (0x5)	DAI1_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI1_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI1_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI1_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI1_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI1_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI1_PB12_O	Pin Buffer 12
0001100 (0xC)	DAI1_PB13_O	Pin Buffer 13
0001101 (0xD)	DAI1_PB14_O	Pin Buffer 14
0001110 (0xE)	DAI1_PB15_O	Pin Buffer 15
0001111 (0xF)	DAI1_PB16_O	Pin Buffer 16
0010000 (0x10)	DAI1_PB17_O	Pin Buffer 17
0010001 (0x11)	DAI1_PB18_O	Pin Buffer 18
0010010 (0x12)	DAI1_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI1_PB20_O	Pin Buffer 20
0010100 (0x14)	SPT4_AD0_O	SPORT 4 Data AD0
0010101 (0x15)	SPT4_AD1_O	SPORT 4 Data AD1
0010110 (0x16)	SPT4_BD0_O	SPORT 4 Data BD0
0010111 (0x17)	SPT4_BD1_O	SPORT 4 Data BD1

Table 33-17: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0011000 (0x18)	SPT5_AD0_O	SPORT 5 Data AD0
0011001 (0x19)	SPT5_AD1_O	SPORT 5 Data AD1
0011010 (0x1A)	SPT5_BD0_O	SPORT 5 Data BD0
0011011 (0x1B)	SPT5_BD1_O	SPORT 5 Data BD1
0011100 (0x1C)	SPT6_AD0_O	SPORT 6 Data AD0
0011101 (0x1D)	SPT6_AD1_O	SPORT 6 Data AD1
0011110 (0x1E)	SPT6_BD0_O	SPORT 6 Data BD0
0011111 (0x1F)	SPT6_BD1_O	SPORT 6 Data BD1
0100000 (0x20)	SPT4_ACLK_O	SPORT 4 Clock A
0100001 (0x21)	SPT4_BCLK_O	SPORT4 Clock B
0100010 (0x22)	SPT5_ACLK_O	SPORT 5 Clock A
0100011 (0x23)	SPT5_BCLK_O	SPORT 5 Clock B
0100100 (0x24)	SPT6_ACLK_O	SPORT 6 Clock A
0100101 (0x25)	SPT6_BCLK_O	SPORT 6 Clock B
0100110 (0x26)	SPT4_AFS_O	SPORT 4 Frame Sync A
0100111 (0x27)	SPT4_BFS_O	SPORT 4 Frame Sync B
0101000 (0x28)	SPT5_AFS_O	SPORT 5 Frame Sync A
0101001 (0x29)	SPT5_BFS_O	SPORT 5 Frame Sync B
0101010 (0x2A)	SPT6_AFS_O	SPORT 5 Frame Sync A
0101011 (0x2B)	SPT6_BFS_O	SPORT 5 Frame Sync B
0101100 (0x2C)	SPT7_AD0_O	SPORT 7 Data AD0
0101101 (0x2D)	SPT7_AD1_O	SPORT 7 Data AD1
0101110 (0x2E)	SPT7_BD0_O	SPORT 7 Data BD0
0101111 (0x2F)	SPT7_BD1_O	SPORT 7 Data BD1
0110000 (0x30)	MLB0_CLKOUT	MLB PLL clock output
0110001 (0x31)	SPDIF1_TX_BLKSTART_O	SPDIF 1 TX Block Start Output
0110100 (0x34)	SPT7_ACLK_O	SPORT 7 Clock A
0110101 (0x35)	SPT7_BCLK_O	SPORT 7 Clock B
0110110 (0x36)	SPT7_AFS_O	SPORT 7 Frame Sync A
0110111 (0x37)	SPT7_BFS_O	SPORT 7 Frame Sync B
0111000 (0x38)	PCG0_CLKC_O	Precision Clock C

Table 33-17: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
0111001 (0x39)	PCG0_CLKD_O	Precision Clock D
0111010 (0x3A)	PCG0_FSC_O	Precision Frame Sync C
0111011 (0x3B)	PCG0_FSD_O	Precision Frame Sync D
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC4_DAT_OP_O	SRC4 Data Output
0111110 (0x3E)	SRC5_DAT_OP_O	SRC5 Data Output
0111111 (0x3F)	SRC6_DAT_OP_O	SRC6 Data Output
1000000 (0x40)	SRC7_DAT_OP_O	SRC7 Data Output
1000001 (0x41)	SPDIF1_RX_DAT_O	SPDIF 1 RX Data Output
1000010 (0x42)	SPDIF1_FS_O	SPDIF 1 RX Frame Sync Output
1000011 (0x43)	SPDIF1_RXCLK_O	SPDIF 1 RX Clock Output
1000100 (0x44)	SPDIF1_RX_TDMCLK_O	SPDIF 1 RX TDM Clock Output
1000101 (0x45)	SPDIF1_TX_O	SPDIF 1 TX Biphase Encoded Data Output
1000110 (0x46)	SPT4_ATDV_O	SPORT4 Transmit A Data Valid Output
1000111 (0x47)	SPT4_BTDV_O	SPORT4 Transmit B Data Valid Output
1001000 (0x48)	SPT5_ATDV_O	SPORT5 Transmit A Data Valid Output
1001001 (0x49)	SPT5_BTDV_O	SPORT5 Transmit B Data Valid Output
1001010 (0x4A)	SPT6_ATDV_O	SPORT6 Transmit A Data Valid Output
1001011 (0x4B)	SPT6_BTDV_O	SPORT6 Transmit B Data Valid Output
1001100 (0x4C)	SPT7_ATDV_O	SPORT7 Transmit A Data Valid Output
1001101 (0x4D)	SPT7_BTDV_O	SPORT7 Transmit B Data Valid Output
1001110 (0x4E)	Reserved	
1001111 (0x4F)	Reserved	
1010000 (0x50)	PCG0_CRS_CLKA_O	Precision Clock A (DAI0)
1010001 (0x51)	PCG0_CRS_CLKB_O	Precision Clock B (DAI0)
1010010 (0x52)	PCG0_CRS_FSA_O	Precision Frame Sync A (DAI0)
1010011 (0x53)	PCG0_CRS_FSB_O	Precision Frame Sync B (DAI0)
1010100 (0x54)	DAI1_CRS_PB03_O	Pin Buffer 3 (DAI0)
1010101 (0x55)	DAI1_CRS_PB04_O	Pin Buffer 4 (DAI0)
1010110 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)

Table 33-17: DAI1 Group D – Pin Signal Assignments (Continued)

Selection Code	Source Signal	Description (Source Selection)
1111111 (0x7F)	HIGH	Logic Level High (1)

DAI1 Group E – Miscellaneous Source Signals

The group E miscellaneous signals are listed in the following table. The group E destinations are configured using Miscellaneous Control Register 0 and Miscellaneous Control Register 1 .

Table 33-18: DAI1 Group E – Miscellaneous Signals

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI1_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI1_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI1_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI1_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI1_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI1_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI1_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI1_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI1_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI1_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI1_PB11_O	Pin Buffer 11 Output
01011 (0xB)	DAI1_PB12_O	Pin Buffer 12 Output
01100 (0xC)	DAI1_PB13_O	Pin Buffer 13 Output
01101 (0xD)	DAI1_PB14_O	Pin Buffer 14 Output
01110 (0xE)	DAI1_PB15_O	Pin Buffer 15 Output
01111 (0xF)	DAI1_PB16_O	Pin Buffer 16 Output
10000 (0x10)	DAI1_PB17_O	Pin Buffer 17 Output
10001 (0x11)	DAI1_PB18_O	Pin Buffer 18 Output
10010 (0x12)	DAI1_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI1_PB20_O	Pin Buffer 20 Output
10100 (0x14)	SPT4_AFS_O	SPORT 4 Frame Sync A
10101 (0x15)	SPT4_BFS_O	SPORT 4 Frame Sync B
10110 (0x16)	SPT5_AFS_O	SPORT 5 Frame Sync A
10111 (0x17)	SPT5_BFS_O	SPORT 5 Frame Sync B

Table 33-18: DAI1 Group E – Miscellaneous Signals (Continued)

Selection Code	Source Signal	Description (Source Selection)
11000 (0x18)	SPT6_AFS_O	SPORT 6 Frame Sync A
11001 (0x19)	SPT6_BFS_O	SPORT 6 Frame Sync B
11010 (0x1A)	SPDIF1_TX_BLKSTART_O	SPDIF 1 TX Block Start Output
11011 (0x1B)	PCG0_FSC_O	Precision Frame Sync C
11100 (0x1C)	PCG0_CLKD_O	Precision Clock D
11101 (0x1D)	PCG0_FSD_O	Precision Frame Sync D
11110 (0x1E)	LOW	Logic Level Low (0) as a Source
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

DAI1 Group F – Pin Output Enable Source Signals

The group F pin output enable signals are listed in the following table. The group F destinations are configured using Pin Buffer Enable Register 0 through Pin Buffer Enable Register 3 .

Table 33-19: DAI1 Group F – Pin Output Enable

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DAI1_MISCA0_O	DAI1 Miscellaneous Control A0 Output
000011 (0x3)	DAI1_MISCA1_O	DAI1 Miscellaneous Control A1 Output
000100 (0x4)	DAI1_MISCA2_O	DAI1 Miscellaneous Control A2 Output
000101 (0x5)	DAI1_MISCA3_O	DAI1 Miscellaneous Control A3 Output
000110 (0x6)	DAI1_MISCA4_O	DAI1 Miscellaneous Control A4 Output
000111 (0x7)	DAI1_MISCA5_O	DAI1 Miscellaneous Control A5 Output
001000 (0x8)	SPT4_ACLK_PBEN_O	SPORT 4 Clock A Output Enable
001001 (0x9)	SPT4_AFS_PBEN_O	SPORT 4 Frame Sync A Output Enable
001010 (0xA)	SPT4_AD0_PBEN_O	SPORT 4 Data AD0 Output Enable
001011 (0xB)	SPT4_AD1_PBEN_O	SPORT 4 Data AD1 Output Enable
001100 (0xC)	SPT4_BCLK_PBEN_O	SPORT 4 Clock B Output Enable
001101 (0xD)	SPT4_BFS_PBEN_O	SPORT 4 Frame Sync B Output Enable
001110 (0xE)	SPT4_BD0_PBEN_O	SPORT 4 Data BD0 Output Enable
001111 (0xF)	SPT4_BD1_PBEN_O	SPORT 4 Data BD1 Output Enable
010000 (0x10)	SPT5_ACLK_PBEN_O	SPORT 5 Clock A Output Enable

Table 33-19: DAI1 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
010001 (0x11)	SPT5_AFS_PBEN_O	SPORT 5 Frame Sync A Output Enable
010010 (0x12)	SPT5_AD0_PBEN_O	SPORT 5 Data AD0 Output Enable
010011 (0x13)	SPT5_AD1_PBEN_O	SPORT 5 Data AD1 Output Enable
010100 (0x14)	SPT5_BCLK_PBEN_O	SPORT 5 Clock B Output Enable
010101 (0x15)	SPT5_BFS_PBEN_O	SPORT 5 Frame Sync B Output Enable
010110 (0x16)	SPT5_BD0_PBEN_O	SPORT 5 Data BD0 Output Enable
010111 (0x17)	SPT5_BD1_PBEN_O	SPORT 5 Data BD1 Output Enable
011000 (0x18)	SPT6_ACLK_PBEN_O	SPORT 6 Clock A Output Enable
011001 (0x19)	SPT6_AFS_PBEN_O	SPORT 6 Frame Sync A Output Enable
011010 (0x1A)	SPT6_AD0_PBEN_O	SPORT 6 Data AD0 Output Enable
011011 (0x1B)	SPT6_AD1_PBEN_O	SPORT 6 Data AD1 Output Enable
011100 (0x1C)	SPT6_BCLK_PBEN_O	SPORT 6 Clock B Output Enable
011101 (0x1D)	SPT6_BFS_PBEN_O	SPORT 6 Frame Sync B Output Enable
011110 (0x1E)	SPT6_BD0_PBEN_O	SPORT 6 Data BD0 Output Enable
011111 (0x1F)	SPT6_BD1_PBEN_O	SPORT 6 Data BD1 Output Enable
100000 (0x20)	SPT7_ACLK_PBEN_O	SPORT 7 Clock A Output Enable
100001 (0x21)	SPT7_AFS_PBEN_O	SPORT 7 Frame Sync A Output Enable
100010 (0x22)	SPT7_AD0_PBEN_O	SPORT 7 Data AD0 Output Enable
100011 (0x23)	SPT7_AD1_PBEN_O	SPORT 7 Data AD1 Output Enable
100100 (0x24)	SPT7_BCLK_PBEN_O	SPORT 6 Clock B Output Enable
100101 (0x25)	SPT7_BFS_PBEN_O	SPORT 7 Frame Sync B Output Enable
100110 (0x26)	SPT7_BD0_PBEN_O	SPORT 7 Data BD0 Output Enable
100111 (0x27)	SPT7_BD1_PBEN_O	SPORT 7 Data BD1 Output Enable
101000 (0x28)	SPT4_ATDV_PBEN_O	SPORT 4 A Transmit Data Valid Output
101001 (0x29)	SPT4_BTDV_PBEN_O	SPORT 4 B Transmit Data Valid Output
101010 (0x2A)	SPT5_ATDV_PBEN_O	SPORT 5 A Transmit Data Valid Output
101011 (0x2B)	SPT5_BTDV_PBEN_O	SPORT 5 B Transmit Data Valid Output
101100 (0x2C)	SPT6_ATDV_PBEN_O	SPORT 6 A Transmit Data Valid Output
101101 (0x2D)	SPT6_BTDV_PBEN_O	SPORT 6 B Transmit Data Valid Output
100111 (0x2E)	SPT7_ATDV_PBEN_O	SPORT 7 A Transmit Data Valid Output
101110 (0x2F)	SPT7_BTDV_PBEN_O	SPORT 7 B Transmit Data Valid Output

Table 33-19: DAI1 Group F – Pin Output Enable (Continued)

Selection Code	Source Signal	Description (Source Selection)
110000 (0x30)–1111111 (0x3F)	Reserved	

DAI Destination Registers Overview

The tables in DAI Routing Capabilities provide high level descriptions that illustrate source (output) connections to destinations (inputs) depending on the routing groups A through F. This sections lists the various input fields (INx) which are described in detail in the ADSP-SC58x DAI Register Descriptions section.

Table 33-20: Clock Destination Registers (Group A)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_CLK0	IN0	SPT0_ACLK_I	SPT4_ACLK_I	SPORT0A or SPORT4A Clock
	IN1	SPT0_BCLK_I	SPT4_BCLK_I	SPORT0B or SPORT4B Clock
	IN2	SPT1_ACLK_I	SPT5_ACLK_I	SPORT1A or SPORT5A Clock
	IN3	SPT1_BCLK_I	SPT5_BCLK_I	SPORT1B or SPORT5B Clock
	IN4	SPT2_ACLK_I	SPT6_ACLK_I	SPORT2A or SPORT6A Clock
	IN5	SPT2_BCLK_I	SPT6_BCLK_I	SPORT2B or SPORT6B Clock
DAI_CLK1	IN0	SRC0_CLK_IP_I	SRC4_CLK_IP_I	SRC0 or SRC4 Clock Input
	IN1	SRC0_CLK_OP_I	SRC4_CLK_OP_I	SRC0 or SRC4 Clock Output
	IN2	SRC1_CLK_IP_I	SRC5_CLK_IP_I	SRC1 or SRC5 Clock Input
	IN3	SRC1_CLK_OP_I	SRC5_CLK_OP_I	SRC1 or SRC5 Clock Output
	IN4	SRC2_CLK_IP_I	SRC6_CLK_IP_I	SRC2 or SRC6 Clock Input
	IN5	SRC2_CLK_OP_I	SRC6_CLK_OP_I	SRC2 or SRC6 Clock Output
DAI_CLK2	IN0	SRC3_CLK_IP_I	SRC7_CLK_IP_I	SRC3 or SRC7 Clock Input
	IN1	SRC3_CLK_OP_I	SRC7_CLK_OP_I	SRC3 or SRC7 Clock Output
	IN2	SPDIF0_TX_CLK_I	SPDIF1_TX_CLK_I	SPDIF0 or SPDIF1 TX Clock
DAI_CLK3	IN5	SPDIF0_TX_HFCLK_I	SPDIF1_TX_HFCLK_I	SPDIF0 or SPDIF1 TX HF Clock
DAI_CLK4	IN0	PCG0_EXTCLKA_I	PCG0_EXTCLKC_I	PCG0 External Clock A or C
	IN1	PCG0_EXTCLKB_I	PCG0_EXTCLKD_I	PCG0 External Clock B or D
	IN3	SPDIF0_TX_EXT_SYNC_I	SPDIF1_TX_EXT_SYNC_I	SPDIF0 or SPDIF1 TX External Sync
	IN4	PCG0_SYNC_CLKA_I	PCG0_SYNC_CLKC_I	PCG0 Sync Clock A or C
	IN5	PCG0_SYNC_CLKB_I	PCG0_SYNC_CLKD_I	PCG0 Sync Clock B or D

Table 33-20: Clock Destination Registers (Group A) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_CLK5	IN0	SPT3_ACLK_I	SPT7_ACLK_I	SPORT3A or SPORT 7A Clock
	IN1	SPT3_BCLK_I	SPT7_BCLK_I	SPORT3B or SPORT 7B Clock

Table 33-21: Data Destination Registers (Group B)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_DAT0	IN0	SPT0_AD0_I	SPT4_AD0_I	SPORT0A or SPORT4A Primary Data
	IN1	SPT0_AD1_I	SPT4_AD1_I	SPORT0A or SPORT4A Secondary Data
	IN2	SPT0_BD0_I	SPT4_BD0_I	SPORT0B or SPORT4B Primary Data
	IN3	SPT0_BD1_I	SPT4_BD1_I	SPORT0B or SPORT4B Secondary Data
	IN4	SPT1_AD0_I	SPT5_AD0_I	SPORT1A or SPORT5A Primary Data
DAI_DAT1	IN0	SPT1_AD1_I	SPT5_AD1_I	SPORT1A or SPORT5A Secondary Data
	IN1	SPT1_BD0_I	SPT5_BD0_I	SPORT1B or SPORT5B Primary Data
	IN2	SPT1_BD1_I	SPT5_BD1_I	SPORT1B or SPORT5B Secondary Data
	IN3	SPT2_AD0_I	SPT6_AD0_I	SPORT2A or SPORT6A Primary Data
	IN4	SPT2_AD1_I	SPT6_AD1_I	SPORT2A or SPORT6A Secondary Data
DAI_DAT2	IN0	SPT2_BD0_I	SPT6_BD0_I	SPORT2B or SPORT6B Primary Data
	IN1	SPT2_BD1_I	SPT6_BD1_I	SPORT2B or SPORT6B Secondary Data
	IN2	SRC0_DAT_IP_I	SRC4_DAT_IP_I	SRC0 or SRC4 Data
	IN3	SRC1_DAT_IP_I	SRC5_DAT_IP_I	SRC1 or SRC5 Data
	IN4	SRC2_DAT_IP_I	SRC6_DAT_IP_I	SRC2 or SRC6 Data

Table 33-21: Data Destination Registers (Group B) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_DAT3	IN0	SRC3_DAT_IP_I	SRC7_DAT_IP_I	SRC3 or SRC7 Data
	IN1	SRC0_DAT_TDM_OP_I	SRC4_DAT_TDM_OP_I	SRC0 or SRC4 TDM output port
	IN2	SRC1_DAT_TDM_OP_I	SRC5_DAT_TDM_OP_I	SRC1 or SRC5 TDM Output port Data
	IN3	SRC2_DAT_TDM_OP_I	SRC6_DAT_TDM_OP_I	SRC2 or SRC6 TDM Output port Data
	IN4	SRC3_DAT_TDM_OP_I	SRC7_DAT_TDM_OP_I	SRC3 or SRC7 TDM Output port Data
DAI_DAT4	IN0	SPDIF0_TX_DAT_I	SPDIF1_TX_DAT_I	SPDIF0 or SPDIF1 serial transmitter Data
DAI_DAT5	IN4	SPDIF0_RX_I	SPDIF1_RX_I	SPDIF0 or SPDIF1 receiver bi-phase Data
DAI_DAT6	IN0	SPT3_AD0_I	SPT7_AD0_I	SPORT3A or SPORT7A Primary Data
	IN1	SPT3_AD1_I	SPT7_AD1_I	SPORT3A or SPORT7A Secondary Data
	IN2	SPT3_BD0_I	SPT7_BD0_I	SPORT3B or SPORT7B Primary Data
	IN3	SPT3_BD1_I	SPT7_BD1_I	SPORT3B or SPORT7B Secondary Data
	IN4	Reserved	Reserved	

Table 33-22: Frame Sync Destination Registers (Group C)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_FS0	IN0	SPT0_AFS_I	SPT4_AFS_I	SPORT0A or SPORT4A Frame Sync
	IN1	SPT0_BFS_I	SPT4_BFS_I	SPORT0B or SPORT4B Frame Sync
	IN2	SPT1_AFS_I	SPT5_AFS_I	SPORT1A or SPORT5A Frame Sync
	IN3	SPT1_BFS_I	SPT5_BFS_I	SPORT1B or SPORT5B Frame Sync
	IN4	SPT2_AFS_I	SPT6_AFS_I	SPORT2A or SPORT6A Frame Sync
	IN5	SPT2_BFS_I	SPT6_BFS_I	SPORT2B or SPORT6B Frame Sync
DAI_FS1	IN0	SRC0_FS_IP_I	SRC4_FS_IP_I	SRC0 or SRC4 Frame Sync Input
	IN1	SRC0_FS_OP_I	SRC4_FS_OP_I	SRC0 or SRC4 Frame Sync Output
	IN2	SRC1_FS_IP_I	SRC5_FS_IP_I	SRC1 or SRC5 Frame Sync Input
	IN3	SRC1_FS_OP_I	SRC5_FS_OP_I	SRC1 or SRC5 Frame Sync Output
	IN4	SRC2_FS_IP_I	SRC6_FS_IP_I	SRC2 or SRC6 Frame Sync Input

Table 33-22: Frame Sync Destination Registers (Group C) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN5	SRC2_FS_OP_I	SRC6_FS_OP_I	SRC2 or SRC6 Frame Sync Output
DAI_FS2	IN0	SRC3_FS_IP_I	SRC7_FS_IP_I	SRC3 or SRC7 Frame Sync Input
	IN1	SRC3_FS_OP_I	SRC7_FS_OP_I	SRC3 or SRC7 Frame Sync Output
	IN2	SPDIF0_TX_FS_I	SPDIF1_TX_FS_I	SPDIF0 or SPDIF1 Transmit Frame Sync
DAI_FS4	IN0	SPT3_AFS_I	SPT7_AFS_I	SPORT3A or SPORT7A Frame Sync
	IN1	SPT3_BFS_I	SPT7_BFS_I	SPORT3B or SPORT7B Frame Sync

Table 33-23: Pin Buffer Assignment Destination Registers (Group D)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_PIN0	IN0	DAI0_PB01_I	DAI1_PB01_I	Pin Buffer 1
	IN1	DAI0_PB02_I	DAI1_PB02_I	Pin Buffer 2
	IN2	DAI0_PB03_I	DAI1_PB03_I	Pin Buffer 3
	IN3	DAI0_PB04_I	DAI1_PB04_I	Pin Buffer 4
DAI_PIN1	IN0	DAI0_PB05_I	DAI1_PB05_I	Pin Buffer 5
	IN1	DAI0_PB06_I	DAI1_PB06_I	Pin Buffer 6
	IN2	DAI0_PB07_I	DAI1_PB07_I	Pin Buffer 7
	IN3	DAI0_PB08_I	DAI1_PB08_I	Pin Buffer 8
DAI_PIN2	IN0	DAI0_PB09_I	DAI1_PB09_I	Pin Buffer 9
	IN1	DAI0_PB10_I	DAI1_PB10_I	Pin Buffer 10
	IN2	DAI0_PB11_I	DAI1_PB11_I	Pin Buffer 11
	IN3	DAI0_PB12_I	DAI1_PB12_I	Pin Buffer 12
DAI_PIN3	IN0	DAI0_PB13_I	DAI1_PB13_I	Pin Buffer 13
	IN1	DAI0_PB14_I	DAI1_PB14_I	Pin Buffer 14
	IN2	DAI0_PB15_I	DAI1_PB15_I	Pin Buffer 15
	IN3	DAI0_PB16_I	DAI1_PB16_I	Pin Buffer 16
DAI_PIN4	IN0	DAI0_PB17_I	DAI1_PB17_I	Pin Buffer 17
	IN1	DAI0_PB18_I	DAI1_PB18_I	Pin Buffer 18
	IN2	DAI0_PB19_I	DAI1_PB19_I	Pin Buffer 19
	IN3	DAI0_PB20_I	DAI1_PB20_I	Pin Buffer 20
	IN4	INV_DAI0_PB19_I	INV_DAI1_PB19_I	Inverted Pin Buffer 19

Table 33-23: Pin Buffer Assignment Destination Registers (Group D) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN5	INV_DAI0_PB20_I	INV_DAI1_PB20_I	Inverted Pin Buffer 20

Table 33-24: Pin Buffer Enable Destination Registers (Group E)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_PBEN0	IN0	DAI0_PBEN01_I	DAI1_PBEN01_I	Pin Buffer 1 Enable
	IN1	DAI0_PBEN02_I	DAI1_PBEN02_I	Pin Buffer 2 Enable
	IN2	DAI0_PBEN03_I	DAI1_PBEN03_I	Pin Buffer 3 Enable
	IN3	DAI0_PBEN04_I	DAI1_PBEN04_I	Pin Buffer 4 Enable
	IN4	DAI0_PBEN05_I	DAI1_PBEN05_I	Pin Buffer 5 Enable
DAI_PBEN1	IN0	DAI0_PBEN06_I	DAI1_PBEN06_I	Pin Buffer 6 Enable
	IN1	DAI0_PBEN07_I	DAI1_PBEN07_I	Pin Buffer 7 Enable
	IN2	DAI0_PBEN08_I	DAI1_PBEN08_I	Pin Buffer 8 Enable
	IN3	DAI0_PBEN09_I	DAI1_PBEN09_I	Pin Buffer 9 Enable
	IN4	DAI0_PBEN10_I	DAI1_PBEN10_I	Pin Buffer 10 Enable
DAI_PBEN2	IN0	DAI0_PBEN11_I	DAI1_PBEN11_I	Pin Buffer 11 Enable
	IN1	DAI0_PBEN12_I	DAI1_PBEN12_I	Pin Buffer 12 Enable
	IN2	DAI0_PBEN13_I	DAI1_PBEN13_I	Pin Buffer 13 Enable
	IN3	DAI0_PBEN14_I	DAI1_PBEN14_I	Pin Buffer 14 Enable
	IN4	DAI0_PBEN15_I	DAI1_PBEN15_I	Pin Buffer 15 Enable
DAI_PBEN3	IN0	DAI0_PBEN16_I	DAI1_PBEN16_I	Pin Buffer 16 Enable
	IN1	DAI0_PBEN17_I	DAI1_PBEN17_I	Pin Buffer 17 Enable
	IN2	DAI0_PBEN18_I	DAI1_PBEN18_I	Pin Buffer 18 Enable
	IN3	DAI0_PBEN19_I	DAI1_PBEN19_I	Pin Buffer 19 Enable
	IN4	DAI0_PBEN20_I	DAI1_PBEN20_I	Pin Buffer 20 Enable

The the ***Miscellaneous Control Registers*** table, DAI_MISCAx_I is an alias name for DAI_INT_x_I.

Table 33-25: Miscellaneous Control Destination Registers (Group F)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
DAI_MISC0	IN6	DAI0_MISCA0_I/ DAI0_INT_6_I	DAI1_MISCA0_I/ DAI1_INT_6_I	DAIx miscellaneous A0 input, or DAIx interrupt 28

Table 33-25: Miscellaneous Control Destination Registers (Group F) (Continued)

DAI Register	Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
	IN7	DAI0_MISCA1_I/ DAI0_INT_7_I	DAI1_MISCA1_I/ DAI1_INT_7_I	DAIx miscellaneous A1 input, or DAIx interrupt 29
	IN8	DAI0_MISCA2_I/ DAI0_INT_8_I	DAI1_MISCA2_I/ DAI1_INT_8_I	DAIx miscellaneous A2 input, or DAIx interrupt 30
	IN9	DAI0_MISCA3_I/ DAI0_INT_9_I	DAI1_MISCA3_I/ DAI1_INT_9_I	DAIx miscellaneous A3 input, or DAIx interrupt 31
	IN10	DAI0_MISCA4_I	DAI1_MISCA4_I	DAIx miscellaneous A4 input
	IN11	DAI0_MISCA5_I	DAI1_MISCA5_I	DAIx miscellaneous A5 input
	INV10	DAI0_INV_MISCA4_I	DAI1_INV_MISCA4_I	DAIx inverted miscellaneous A4 input
	INV11	DAI0_INV_MISCA5_I	DAI1_INV_MISCA5_I	DAIx inverted miscellaneous A5 input
DAI_MISC1	IN0	DAI0_INT_0	DAI1_INT_0	DAIx interrupt 22
	IN1	DAI0_INT_1	DAI1_INT_1	DAIx interrupt 23
	IN2	DAI0_INT_2	DAI1_INT_2	DAIx interrupt 24
	IN3	DAI0_INT_3	DAI1_INT_3	DAIx interrupt 25
	IN4	DAI0_INT_4	DAI1_INT_4	DAIx interrupt 26
	IN5	DAI0_INT_5	DAI1_INT_5	DAIx interrupt 27

The ***Interrupt Events*** table presents the information for the following registers.

- DAI_IMSK_PRI (Core Interrupt Priority Assignment) register
- DAI_IMSK_FE (Falling-Edge Interrupt Mask) and DAI_IMSK_RE (Rising-Edge Interrupt Mask) registers
- DAI_IRPTL_H (High-Priority Interrupt Latch) and DAI_IRPTL_L (Low-Priority Interrupt Latch) registers
- DAI_IRPTL_HS (Shadow High Interrupt Latch) and DAI_IRPTL_LS (Shadow Low Interrupt Latch) registers

Table 33-26: Interrupt Events

Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
RXVALID	SPDIF0_RXVALID_INT	SPDIF1_RXVALID_INT	SPDIF0 or SPDIF1 RX Valid interrupt
RXLOCK	SPDIF0_RXLOCK_INT	SPDIF1_RXLOCK_INT	SPDIF0 or SPDIF1 RX Lock interrupt
RXLOSSOFLOCK	SPDIF0_RXLOSSOFLOCK_INT	SPDIF1_RXLOSSOFLOCK_INT	SPDIF0 or SPDIF1 RX Loss of Lock interrupt

Table 33-26: Interrupt Events (Continued)

Bit Field Name	DAI0 Mapping	DAI1 Mapping	Description
RXNONAUDIO	SPDIF0_RXNONAUDIO_INT	SPDIF1_RXNONAUDIO_INT	SPDIF0 or SPDIF1 RX Non-audio interrupt
SRC0MUTE	SRC0_MUTE_INT	SRC4_MUTE_INT	DAI0 or DAI1 ASRC0 Mute interrupt
SRC1MUTE	SRC1_MUTE_INT	SRC5_MUTE_INT	DAI0 or DAI1 ASRC1 Mute interrupt
SRC2MUTE	SRC2_MUTE_INT	SRC6_MUTE_INT	DAI0 or DAI1 ASRC2 Mute interrupt
SRC3MUTE	SRC3_MUTE_INT	SRC7_MUTE_INT	DAI0 or DAI1 ASRC3 Mute interrupt
MISCINT0	DAI0_INT_00	DAI1_INT_00	DAI0 or DAI1 Miscellaneous Interrupt 0
MISCINT1	DAI0_INT_01	DAI1_INT_01	DAI0 or DAI1 Miscellaneous Interrupt 1
MISCINT2	DAI0_INT_02	DAI1_INT_02	DAI0 or DAI1 Miscellaneous Interrupt 2
MISCINT3	DAI0_INT_03	DAI1_INT_03	DAI0 or DAI1 Miscellaneous Interrupt 3
MISCINT4	DAI0_INT_04	DAI1_INT_04	DAI0 or DAI1 Miscellaneous Interrupt 4
MISCINT5	DAI0_INT_05	DAI1_INT_05	DAI0 or DAI1 Miscellaneous Interrupt 5
MISCINT6	DAI0_EXTMISCA0_INT/ DAI0_INT_06	DAI1_EXTMISCA0_INT/ DAI1_INT_06	DAI0 or DAI1 Miscellaneous Interrupt 6 / External Miscellaneous A0 Interrupt
MISCINT7	DAI0_EXTMISCA1_INT/ DAI0_INT_07	DAI1_EXTMISCA1_INT/ DAI1_INT_07	DAI0 or DAI1 Miscellaneous Interrupt 7 / External Miscellaneous A1 Interrupt
MISCINT8	DAI0_EXTMISCA2_INT/ DAI0_INT_08	DAI1_EXTMISCA2_INT/ DAI1_INT_08	DAI0 or DAI1 Miscellaneous Interrupt 8 / External Miscellaneous A2 Interrupt
MISCINT9	DAI0_EXTMISCA3_INT/ DAI0_INT_09	DAI1_EXTMISCA3_INT/ DAI1_INT_09	DAI0 or DAI1 Miscellaneous Interrupt 9 / External Miscellaneous A3 Interrupt

ADSP-SC58x DAI Register Descriptions

The Digital Audio Interface (DAIn) registers are used to configure DAI destinations for the DAI sources shown in the Group A - F Routing tables. (DAI) contains the following registers.

Table 33-27: ADSP-SC58x DAI Register List

Name	Description
DAI_CLK0	Clock Routing Control Register 0
DAI_CLK1	Clock Routing Control Register 1
DAI_CLK2	Clock Routing Control Register 2
DAI_CLK3	Clock Routing Control Register 3
DAI_CLK4	Clock Routing Control Register 4
DAI_CLK5	Clock Routing Control Register 5
DAI_DAT0	Serial Data Routing Control Register 0
DAI_DAT1	Serial Data Routing Control Register 1
DAI_DAT2	Serial Data Routing Control Register 2
DAI_DAT3	Serial Data Routing Control Register 3
DAI_DAT4	Serial Data Routing Control Register 4
DAI_DAT5	Serial Data Routing Control Register 5
DAI_DAT6	Serial Data Routing Control Register 6
DAI_FS0	Frame Sync Routing Control Register 0
DAI_FS1	Frame Sync Routing Control Register 1
DAI_FS2	Frame Sync Routing Control Register 2
DAI_FS4	Frame Sync Routing Control Register 4
DAI_IMSK_FE	Falling-Edge Interrupt Mask Register
DAI_IMSK_PRI	Core Interrupt Priority Assignment Register
DAI_IMSK_RE	Rising-Edge Interrupt Mask Register
DAI_IRPTL_H	High Priority Interrupt Latch Register
DAI_IRPTL_HS	Shadow High Priority Interrupt Latch Register
DAI_IRPTL_L	Low Priority Interrupt Latch Register
DAI_IRPTL_LS	Shadow Low Priority Interrupt Latch Register
DAI_MISC0	Miscellaneous Control Register 0
DAI_MISC1	Miscellaneous Control Register 1
DAI_PBEN0	Pin Buffer Enable Register 0
DAI_PBEN1	Pin Buffer Enable Register 1
DAI_PBEN2	Pin Buffer Enable Register 2
DAI_PBEN3	Pin Buffer Enable Register 3
DAI_PIN0	Pin Buffer Assignment Register 0

Table 33-27: ADSP-SC58x DAI Register List (Continued)

Name	Description
`DAI_PIN1`	Pin Buffer Assignment Register 1
`DAI_PIN2`	Pin Buffer Assignment Register 2
`DAI_PIN3`	Pin Buffer Assignment Register 3
`DAI_PIN4`	Pin Buffer Assignment Register 4
`DAI_PIN_STAT`	Pin Status Register

Clock Routing Control Register 0

The `DAI_CLK0` register provides clock routing connections for the serial ports (SPORTs).

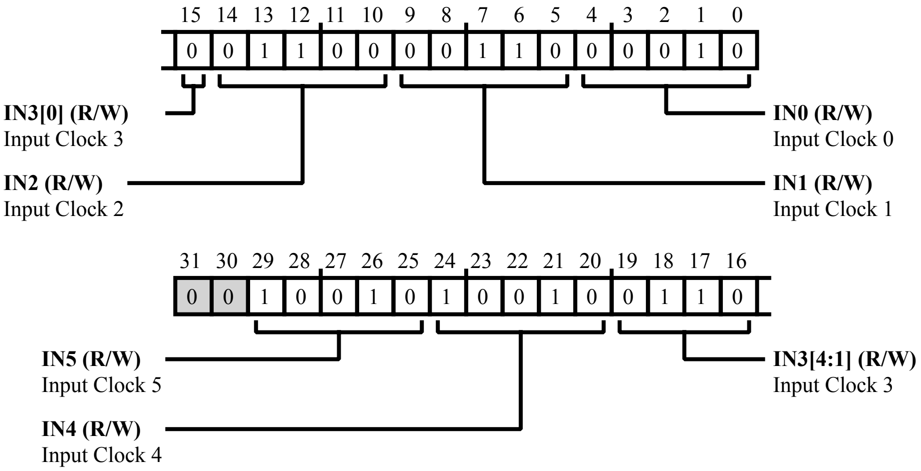

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0
IN3[0] (R/W)
Input Clock 3
IN2 (R/W)
Input Clock 2
IN0 (R/W)
Input Clock 0
IN1 (R/W)
Input Clock 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0
IN5 (R/W)
Input Clock 5
IN4 (R/W)
Input Clock 4
IN3[4:1] (R/W)
Input Clock 3

Figure 33-17: DAI_CLK0 Register Diagram

Table 33-28: DAI_CLK0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. `DAI_CLK0.IN5` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN5` Destination. Refer to the Group A Signals table for Source and Destination mappings
24:20 (R/W)	IN4	Input Clock 4. `DAI_CLK0.IN4` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN4` Destination. Refer to the Group A Signals table for Source and Destination mappings
19:15 (R/W)	IN3	Input Clock 3. `DAI_CLK0.IN3` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN3` Destination. Refer to the Group A Signals table for Source and Destination mappings
14:10 (R/W)	IN2	Input Clock 2. `DAI_CLK0.IN2` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN2` Destination. Refer to the Group A Signals table for Source and Destination mappings
9:5 (R/W)	IN1	Input Clock 1. `DAI_CLK0.IN1` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN1` Destination. Refer to the Group A Signals table for Source and Destination mappings

Table 33-28: DAI_CLK0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Clock 0. `DAI_CLK0.IN0` holds the Source signal assignment that will be routed to the `DAI_CLK0.IN0` Destination. Refer to the Group A Signals table for Source and Destination mappings

Clock Routing Control Register 1

The `DAI_CLK1` register provides clock routing connections for the asynchronous sample rate converters (ASRC).

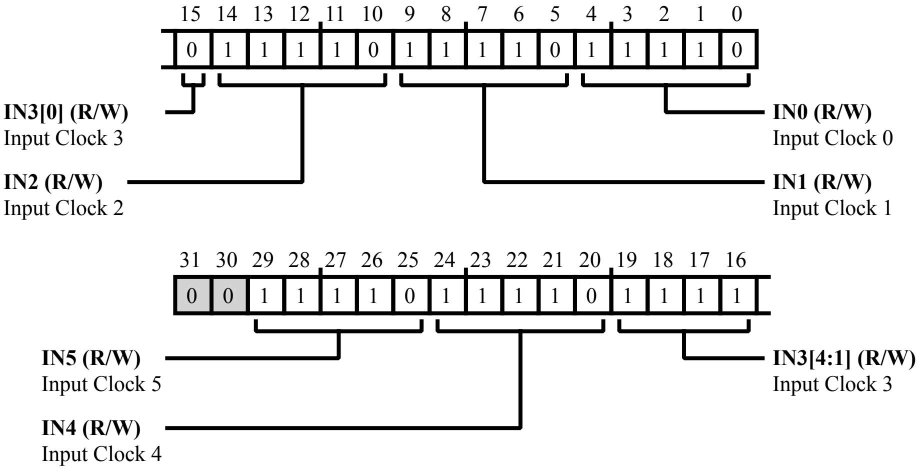

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN3[0] (R/W)
Input Clock 3
IN2 (R/W)
Input Clock 2
IN0 (R/W)
Input Clock 0
IN1 (R/W)
Input Clock 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
IN5 (R/W)
Input Clock 5
IN4 (R/W)
Input Clock 4
IN3[4:1] (R/W)
Input Clock 3

Figure 33-18: DAI_CLK1 Register Diagram

Table 33-29: DAI_CLK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. `DAI_CLK1.IN5` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN5` Destination. Refer to the Group A Signals table for Source and Destination mappings
24:20 (R/W)	IN4	Input Clock 4. `DAI_CLK1.IN4` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN4` Destination. Refer to the Group A Signals table for Source and Destination mappings
19:15 (R/W)	IN3	Input Clock 3. `DAI_CLK1.IN3` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN3` Destination. Refer to the Group A Signals table for Source and Destination mappings
14:10 (R/W)	IN2	Input Clock 2. `DAI_CLK1.IN2` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN2` Destination. Refer to the Group A Signals table for Source and Destination mappings
9:5 (R/W)	IN1	Input Clock 1. `DAI_CLK1.IN1` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN1` Destination. Refer to the Group A Signals table for Source and Destination mappings.

Table 33-29: DAI_CLK1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Clock 0. `DAI_CLK1.IN0` holds the Source signal assignment that will be routed to the `DAI_CLK1.IN0` Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 2

The `DAI_CLK2` register provides clock routing connections for the S/PDIF and ASRC.

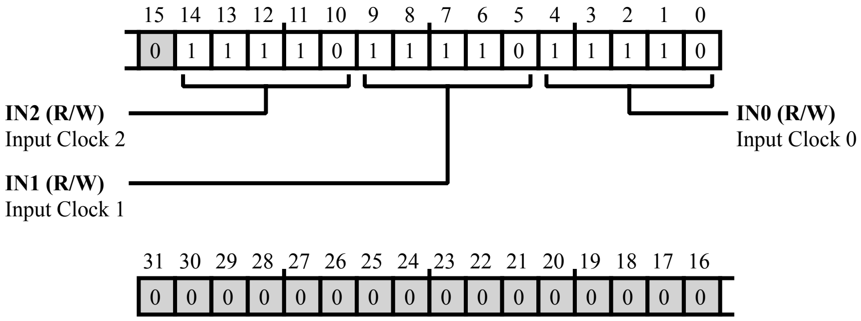

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN2 (R/W)
Input Clock 2
IN1 (R/W)
Input Clock 1
IN0 (R/W)
Input Clock 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 33-19: DAI_CLK2 Register Diagram

Table 33-30: DAI_CLK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:10 (R/W)	IN2	Input Clock 2. `DAI_CLK2.IN2` holds the Source signal assignment that will be routed to the `DAI_CLK2.IN2` Destination. Refer to the Group A Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Clock 1. `DAI_CLK2.IN1` holds the Source signal assignment that will be routed to the `DAI_CLK2.IN1` Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. `DAI_CLK2.IN0` holds the Source signal assignment that will be routed to the `DAI_CLK2.IN0` Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 3

The `DAI_CLK3` register provides clock routing connections for the S/PDIF.

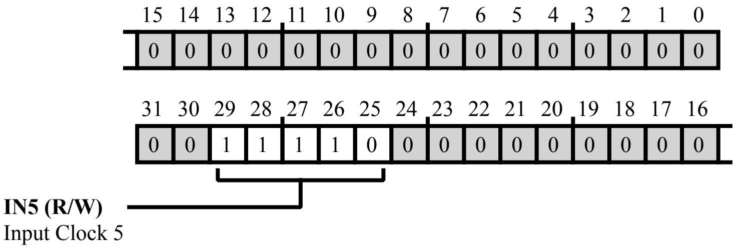

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
IN5 (R/W)
Input Clock 5

Figure 33-20: DAI_CLK3 Register Diagram

Table 33-31: DAI_CLK3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. The `DAI_CLK3.IN5` bit field provides the S/PDIF oversampling clock.

Clock Routing Control Register 4

The DAI_CLK4 register provides clock routing connections for the precision clock generators (PCGs).

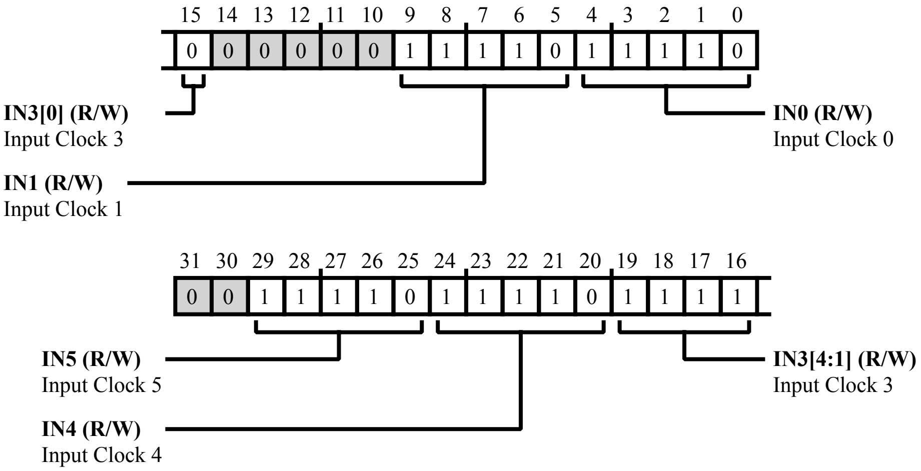

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0
IN3[0] (R/W)
Input Clock 3
IN1 (R/W)
Input Clock 1
IN0 (R/W)
Input Clock 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
IN5 (R/W)
Input Clock 5
IN4 (R/W)
Input Clock 4
IN3[4:1] (R/W)
Input Clock 3

Figure 33-21: DAI_CLK4 Register Diagram

Table 33-32: DAI_CLK4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Clock 5. `DAI_CLK4.IN5` holds the Source signal assignment that will be routed to the `DAI_CLK4.IN5` Destination. Refer to the Group A Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Clock 4. `DAI_CLK4.IN4` holds the Source signal assignment that will be routed to the `DAI_CLK4.IN4` Destination. Refer to the Group A Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Clock 3. `DAI_CLK4.IN3` holds the Source signal assignment that will be routed to the `DAI_CLK4.IN3` Destination. Refer to the Group A Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Clock 1. `DAI_CLK4.IN1` holds the Source signal assignment that will be routed to the `DAI_CLK4.IN1` Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. `DAI_CLK4.IN0` holds the Source signal assignment that will be routed to the `DAI_CLK4.IN0` Destination. Refer to the Group A Signals table for Source and Destination mappings.

Clock Routing Control Register 5

The `DAI_CLK5` register provides clock routing connections for the serial ports.

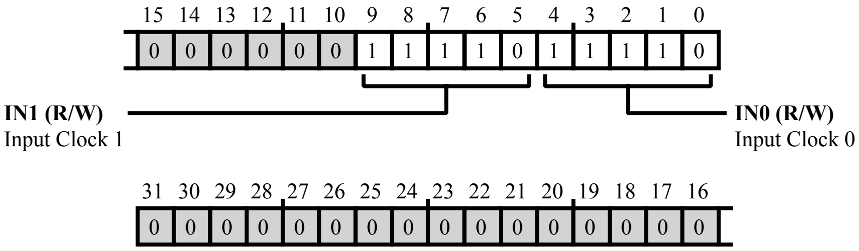

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0
IN1 (R/W)
Input Clock 1
IN0 (R/W)
Input Clock 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 33-22: DAI_CLK5 Register Diagram

Table 33-33: DAI_CLK5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN1	Input Clock 1. `DAI_CLK5.IN1` holds the Source signal assignment that will be routed to the `DAI_CLK5.IN1` Destination. Refer to the Group A Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Clock 0. `DAI_CLK5.IN0` holds the Source signal assignment that will be routed to the `DAI_CLK5.IN0` Destination. Refer to the Group A Signals table for Source and Destination mappings.

Serial Data Routing Control Register 0

The `DAI_DAT0` register routes serial data to the serial ports.

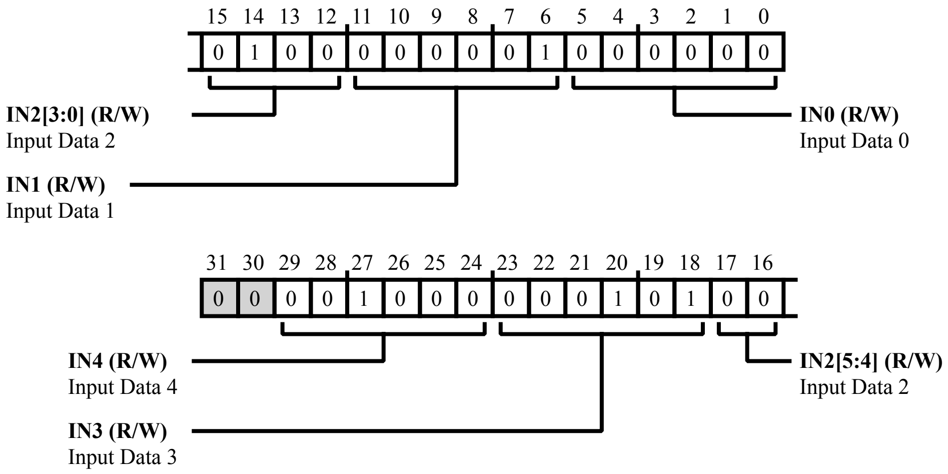

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
IN2[3:0] (R/W)
Input Data 2
IN1 (R/W)
Input Data 1
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0
IN4 (R/W)
Input Data 4
IN3 (R/W)
Input Data 3
IN2[5:4] (R/W)
Input Data 2

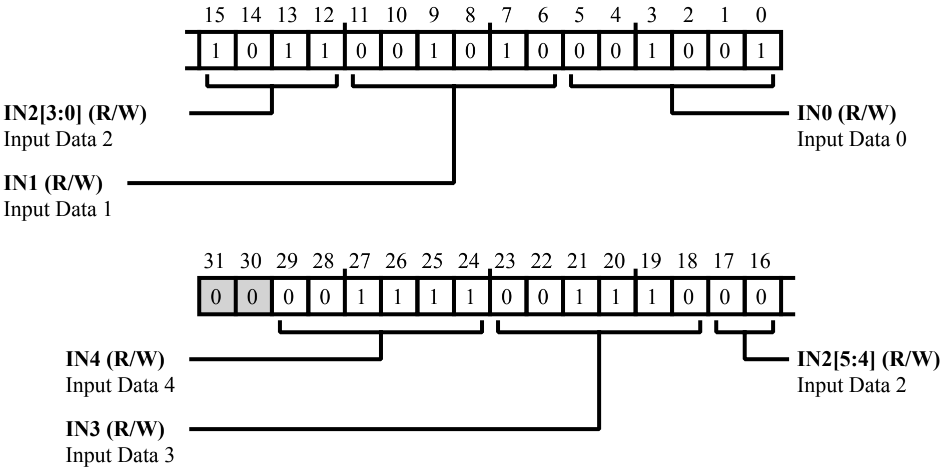
Figure 33-23: DAI_DAT0 Register Diagram

Table 33-34: DAI_DAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. `DAI_DAT0.IN4` holds the Source signal assignment that will be routed to the `DAI_DAT0.IN4` Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. `DAI_DAT0.IN3` holds the Source signal assignment that will be routed to the `DAI_DAT0.IN3` Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. `DAI_DAT0.IN2` holds the Source signal assignment that will be routed to the `DAI_DAT0.IN2` Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. `DAI_DAT0.IN1` holds the Source signal assignment that will be routed to the `DAI_DAT0.IN1` Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT0.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT0.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 1

The `DAI_DAT1` register routes serial data to the serial ports.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 0 0 1 0 1 0 0 0 1 0 0 1
IN2[3:0] (R/W)
Input Data 2
IN1 (R/W)
Input Data 1
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0
IN4 (R/W)
Input Data 4
IN3 (R/W)
Input Data 3
IN2[5:4] (R/W)
Input Data 2

Figure 33-24: DAI_DAT1 Register Diagram

Table 33-35: DAI_DAT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. `DAI_DAT1.IN4` holds the Source signal assignment that will be routed to the `DAI_DAT1.IN4` Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. `DAI_DAT1.IN3` holds the Source signal assignment that will be routed to the `DAI_DAT1.IN3` Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. `DAI_DAT1.IN2` holds the Source signal assignment that will be routed to the `DAI_DAT1.IN2` Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. `DAI_DAT1.IN1` holds the Source signal assignment that will be routed to the `DAI_DAT1.IN1` Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT1.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT1.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 2

The DAI_DAT2 register routes serial data to the serial ports and the asynchronous sample rate converter (ASRC).

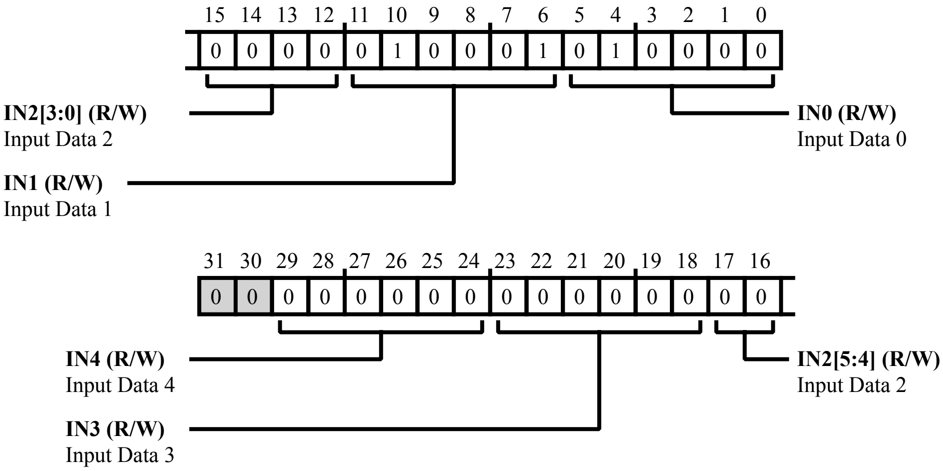

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0
IN2[3:0] (R/W)
Input Data 2
IN1 (R/W)
Input Data 1
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IN4 (R/W)
Input Data 4
IN3 (R/W)
Input Data 3
IN2[5:4] (R/W)
Input Data 2

Figure 33-25: DAI_DAT2 Register Diagram

Table 33-36: DAI_DAT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. `DAI_DAT2.IN4` holds the Source signal assignment that will be routed to the `DAI_DAT2.IN4` Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. `DAI_DAT2.IN3` holds the Source signal assignment that will be routed to the `DAI_DAT2.IN3` Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. `DAI_DAT2.IN2` holds the Source signal assignment that will be routed to the `DAI_DAT2.IN2` Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. `DAI_DAT2.IN1` holds the Source signal assignment that will be routed to the `DAI_DAT2.IN1` Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT2.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT2.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 3

The DAI_DAT3 register routes serial data to the asynchronous sample rate converter (ASRC).

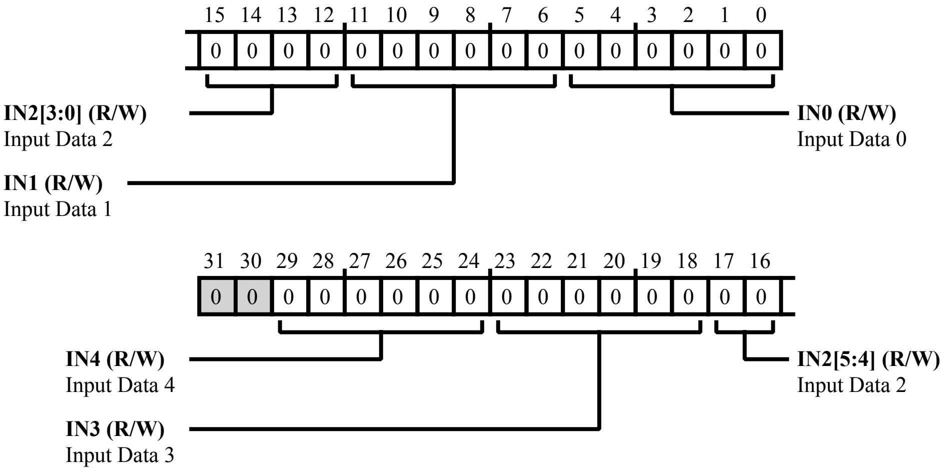

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IN2[3:0] (R/W)
Input Data 2
IN1 (R/W)
Input Data 1
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IN4 (R/W)
Input Data 4
IN3 (R/W)
Input Data 3
IN2[5:4] (R/W)
Input Data 2

Figure 33-26: DAI_DAT3 Register Diagram

Table 33-37: DAI_DAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. `DAI_DAT3.IN4` holds the Source signal assignment that will be routed to the `DAI_DAT3.IN4` Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. `DAI_DAT3.IN3` holds the Source signal assignment that will be routed to the `DAI_DAT3.IN3` Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. `DAI_DAT3.IN2` holds the Source signal assignment that will be routed to the `DAI_DAT3.IN2` Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. `DAI_DAT3.IN1` holds the Source signal assignment that will be routed to the `DAI_DAT3.IN1` Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT3.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT3.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 4

The DAI_DAT4 register routes serial data to the S/PDIF.

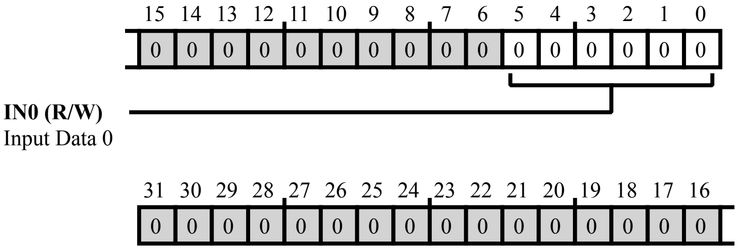

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 33-27: DAI_DAT4 Register Diagram

Table 33-38: DAI_DAT4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT4.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT4.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Serial Data Routing Control Register 5

The DAI_DAT5 register routes serial data to the S/PDIF.

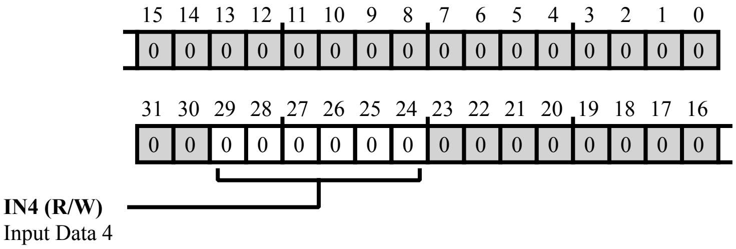

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IN4 (R/W)
Input Data 4

Figure 33-28: DAI_DAT5 Register Diagram

Table 33-39: DAI_DAT5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. The `DAI_DAT5.IN4` bit field routes input data to the S/PDIF Biphase receiver stream.

Serial Data Routing Control Register 6

The `DAI_DAT6` register routes serial data to the serial ports.

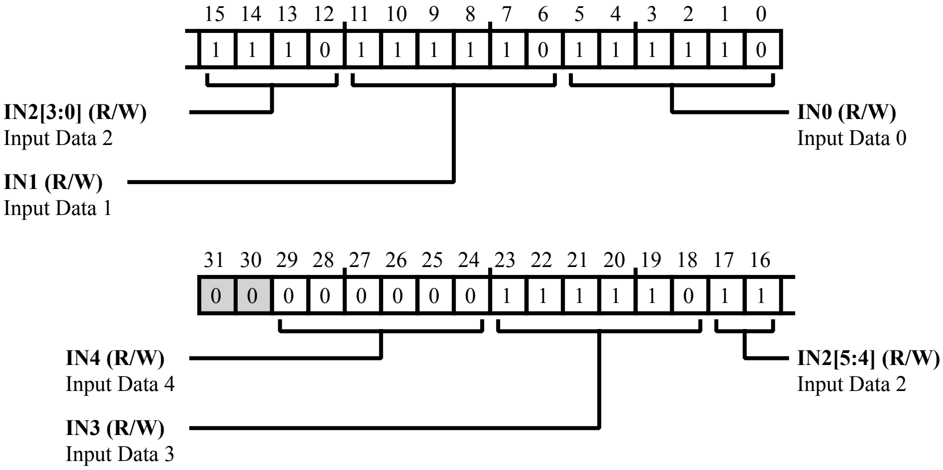

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0
IN2[3:0] (R/W)
Input Data 2
IN1 (R/W)
Input Data 1
IN0 (R/W)
Input Data 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1
IN4 (R/W)
Input Data 4
IN3 (R/W)
Input Data 3
IN2[5:4] (R/W)
Input Data 2

Figure 33-29: DAI_DAT6 Register Diagram

Table 33-40: DAI_DAT6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	IN4	Input Data 4. `DAI_DAT6.IN4` holds the Source signal assignment that will be routed to the `DAI_DAT6.IN4` Destination. Refer to the Group B Signals table for Source and Destination mappings.
23:18 (R/W)	IN3	Input Data 3. `DAI_DAT6.IN3` holds the Source signal assignment that will be routed to the `DAI_DAT6.IN3` Destination. Refer to the Group B Signals table for Source and Destination mappings.
17:12 (R/W)	IN2	Input Data 2. `DAI_DAT6.IN2` holds the Source signal assignment that will be routed to the `DAI_DAT6.IN2` Destination. Refer to the Group B Signals table for Source and Destination mappings.
11:6 (R/W)	IN1	Input Data 1. `DAI_DAT6.IN1` holds the Source signal assignment that will be routed to the `DAI_DAT6.IN1` Destination. Refer to the Group B Signals table for Source and Destination mappings.
5:0 (R/W)	IN0	Input Data 0. `DAI_DAT6.IN0` holds the Source signal assignment that will be routed to the `DAI_DAT6.IN0` Destination. Refer to the Group B Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 0

The DAI_FS0 register routes frame syncs to the serial ports.

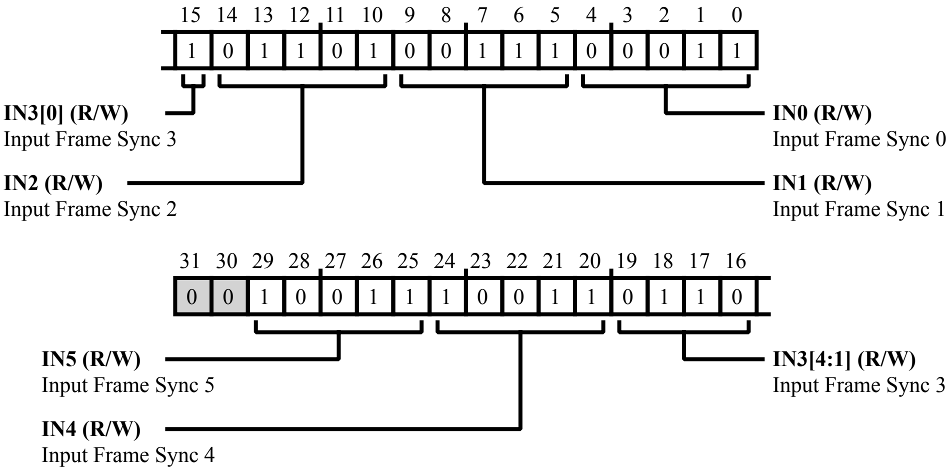

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 1 1 0 1 0 0 1 1 1 0 0 0 1 1
IN3[0] (R/W)
Input Frame Sync 3
IN2 (R/W)
Input Frame Sync 2
IN0 (R/W)
Input Frame Sync 0
IN1 (R/W)
Input Frame Sync 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0
IN5 (R/W)
Input Frame Sync 5
IN4 (R/W)
Input Frame Sync 4
IN3[4:1] (R/W)
Input Frame Sync 3

Figure 33-30: DAI_FS0 Register Diagram

Table 33-41: DAI_FS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Frame Sync 5. `DAI_FS0.IN5` holds the Source signal assignment that will be routed to the `DAI_FS0.IN5` Destination. Refer to the Group C Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Frame Sync 4. `DAI_FS0.IN4` holds the Source signal assignment that will be routed to the `DAI_FS0.IN4` Destination. Refer to the Group C Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Frame Sync 3. `DAI_FS0.IN3` holds the Source signal assignment that will be routed to the `DAI_FS0.IN3` Destination. Refer to the Group C Signals table for Source and Destination mappings.
14:10 (R/W)	IN2	Input Frame Sync 2. `DAI_FS0.IN2` holds the Source signal assignment that will be routed to the `DAI_FS0.IN2` Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. `DAI_FS0.IN1` holds the Source signal assignment that will be routed to the `DAI_FS0.IN1` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Table 33-41: DAI_FS0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Frame Sync 0. `DAI_FS0.IN0` holds the Source signal assignment that will be routed to the `DAI_FS0.IN0` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 1

The `DAI_FS1` register routes frame syncs to the asynchronous sample rate converter (ASRC).

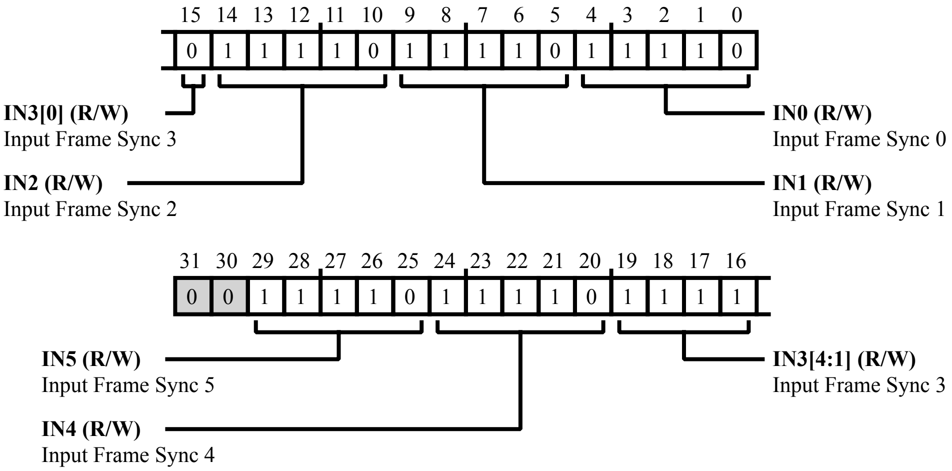

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN3[0] (R/W)
Input Frame Sync 3
IN2 (R/W)
Input Frame Sync 2
IN0 (R/W)
Input Frame Sync 0
IN1 (R/W)
Input Frame Sync 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
IN5 (R/W)
Input Frame Sync 5
IN4 (R/W)
Input Frame Sync 4
IN3[4:1] (R/W)
Input Frame Sync 3

Figure 33-31: DAI_FS1 Register Diagram

Table 33-42: DAI_FS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input Frame Sync 5. `DAI_FS1.IN5` holds the Source signal assignment that will be routed to the `DAI_FS1.IN5` Destination. Refer to the Group C Signals table for Source and Destination mappings.
24:20 (R/W)	IN4	Input Frame Sync 4. `DAI_FS1.IN4` holds the Source signal assignment that will be routed to the `DAI_FS1.IN4` Destination. Refer to the Group C Signals table for Source and Destination mappings.
19:15 (R/W)	IN3	Input Frame Sync 3. `DAI_FS1.IN3` holds the Source signal assignment that will be routed to the `DAI_FS1.IN3` Destination. Refer to the Group C Signals table for Source and Destination mappings.
14:10 (R/W)	IN2	Input Frame Sync 2. `DAI_FS1.IN2` holds the Source signal assignment that will be routed to the `DAI_FS1.IN2` Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. `DAI_FS1.IN1` holds the Source signal assignment that will be routed to the `DAI_FS1.IN1` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Table 33-42: DAI_FS1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input Frame Sync 0. `DAI_FS1.IN0` holds the Source signal assignment that will be routed to the `DAI_FS1.IN0` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 2

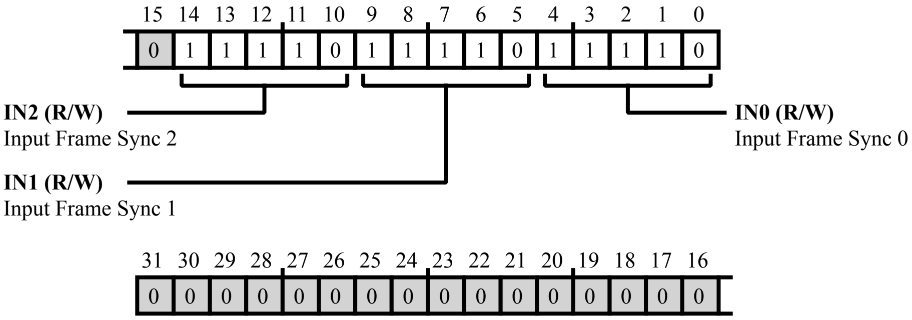

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN2 (R/W)
Input Frame Sync 2
IN1 (R/W)
Input Frame Sync 1
IN0 (R/W)
Input Frame Sync 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 33-32: DAI_FS2 Register Diagram

Table 33-43: DAI_FS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:10 (R/W)	IN2	Input Frame Sync 2. `DAI_FS2.IN2` holds the Source signal assignment that will be routed to the `DAI_FS2.IN2` Destination. Refer to the Group C Signals table for Source and Destination mappings.
9:5 (R/W)	IN1	Input Frame Sync 1. The `DAI_FS2.IN1` bit field routes the frame sync output to the ASRC3 frame sync input port.
4:0 (R/W)	IN0	Input Frame Sync 0. `DAI_FS2.IN0` holds the Source signal assignment that will be routed to the `DAI_FS2.IN0` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Frame Sync Routing Control Register 4

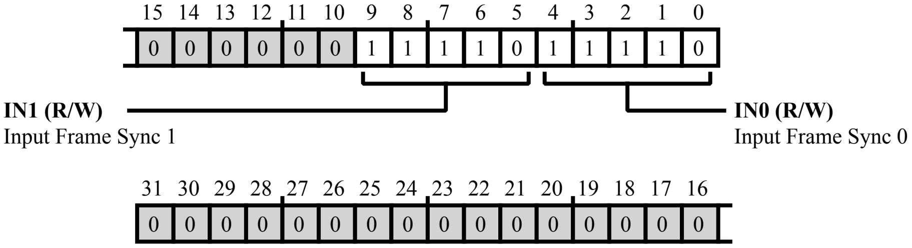

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0
IN1 (R/W)
Input Frame Sync 1
IN0 (R/W)
Input Frame Sync 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 33-33: DAI_FS4 Register Diagram

Table 33-44: DAI_FS4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN1	Input Frame Sync 1. `DAI_FS4.IN1` holds the Source signal assignment that will be routed to the `DAI_FS4.IN1` Destination. Refer to the Group C Signals table for Source and Destination mappings.
4:0 (R/W)	IN0	Input Frame Sync 0. `DAI_FS4.IN0` holds the Source signal assignment that will be routed to the `DAI_FS4.IN0` Destination. Refer to the Group C Signals table for Source and Destination mappings.

Falling-Edge Interrupt Mask Register

The `DAI_IMSK_FE` register masks and unmasks interrupts generated on the falling edge of a waveform. Note that any of the Group E signals can be mapped to any of the Miscellaneous interrupts (9-0).

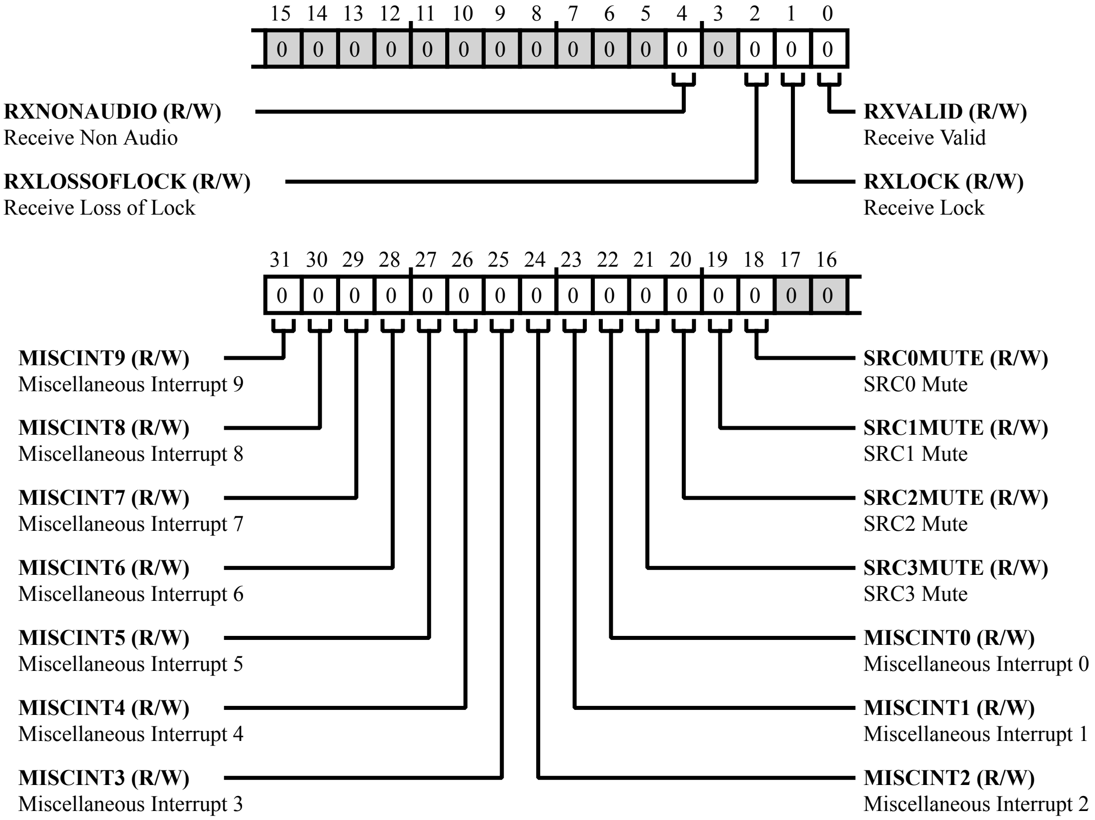

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (R/W)
Receive Non Audio
RXLOSSOFLOCK (R/W)
Receive Loss of Lock
RXVALID (R/W)
Receive Valid
RXLOCK (R/W)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (R/W)
Miscellaneous Interrupt 9
MISCINT8 (R/W)
Miscellaneous Interrupt 8
MISCINT7 (R/W)
Miscellaneous Interrupt 7
MISCINT6 (R/W)
Miscellaneous Interrupt 6
MISCINT5 (R/W)
Miscellaneous Interrupt 5
MISCINT4 (R/W)
Miscellaneous Interrupt 4
MISCINT3 (R/W)
Miscellaneous Interrupt 3
SRC0MUTE (R/W)
SRC0 Mute
SRC1MUTE (R/W)
SRC1 Mute
SRC2MUTE (R/W)
SRC2 Mute
SRC3MUTE (R/W)
SRC3 Mute
MISCINT0 (R/W)
Miscellaneous Interrupt 0
MISCINT1 (R/W)
Miscellaneous Interrupt 1
MISCINT2 (R/W)
Miscellaneous Interrupt 2

Figure 33-34: DAI_IMSK_FE Register Diagram

Table 33-45: DAI_IMSK_FE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. Setting the `DAI_IMSK_FE.MISCINT9` bit unmasks the interrupt for the falling edge of the routed signal.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. Setting the `DAI_IMSK_FE.MISCINT8` bit unmasks the interrupt for the falling edge of the routed signal.
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. Setting the `DAI_IMSK_FE.MISCINT7` bit unmasks the interrupt for the falling edge of the routed signal.

Table 33-45: DAI_IMSK_FE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. Setting the `DAI_IMSK_FE.MISCINT6` bit unmasks the interrupt for the falling edge of the routed signal.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. Setting the `DAI_IMSK_FE.MISCINT5` bit unmasks the interrupt for the falling edge of the routed signal.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. Setting the `DAI_IMSK_FE.MISCINT4` bit unmasks the interrupt for the falling edge of the routed signal.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. Setting the `DAI_IMSK_FE.MISCINT3` bit unmasks the interrupt for the falling edge of the routed signal.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. Setting the `DAI_IMSK_FE.MISCINT2` bit unmasks the interrupt for the falling edge of the routed signal.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. Setting the `DAI_IMSK_FE.MISCINT1` bit unmasks the interrupt for the falling edge of the routed signal.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. Setting the `DAI_IMSK_FE.MISCINT0` bit unmasks the interrupt for the falling edge of the routed signal.
21 (R/W)	SRC3MUTE	SRC3 Mute. The `DAI_IMSK_FE.SRC3MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
20 (R/W)	SRC2MUTE	SRC2 Mute. The `DAI_IMSK_FE.SRC2MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
19 (R/W)	SRC1MUTE	SRC1 Mute. The `DAI_IMSK_FE.SRC1MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.

Table 33-45: DAI_IMSK_FE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SRC0MUTE	SRC0 Mute. The `DAI_IMSK_FE.SRC0MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the falling edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
4 (R/W)	RXNONAUDIO	Receive Non Audio. The `DAI_IMSK_FE.RXNONAUDIO` bit masks or unmasks the non audio frame mode interrupt for the falling edge of this interrupt. If the channel status indicates non-PCM audio, the NONAUDIO bit flag is set.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IMSK_FE.RXLOSSOFLOCK` bit masks or unmasks the emphasis loss of lock interrupt for the falling edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver looses the lock of biphase stream.
1 (R/W)	RXLOCK	Receive Lock. The `DAI_IMSK_FE.RXLOCK` bit masks or unmasks the S/PDIF receiver lock for the falling edge of this interrupt. This interrupt occurs when the S/PDIF receiver locks to the S/PDIF stream.
0 (R/W)	RXVALID	Receive Valid. Setting the `DAI_IMSK_FE.RXVALID` bit unmasks the interrupt for the falling edge of the routed signal. This interrupt is set based on whether data received by the S/PDIF is linear PCM or non-linear audio data.

Core Interrupt Priority Assignment Register

The `DAI_IMSK_PRI` register masks interrupts for DAI high or DAI low interrupt priority.

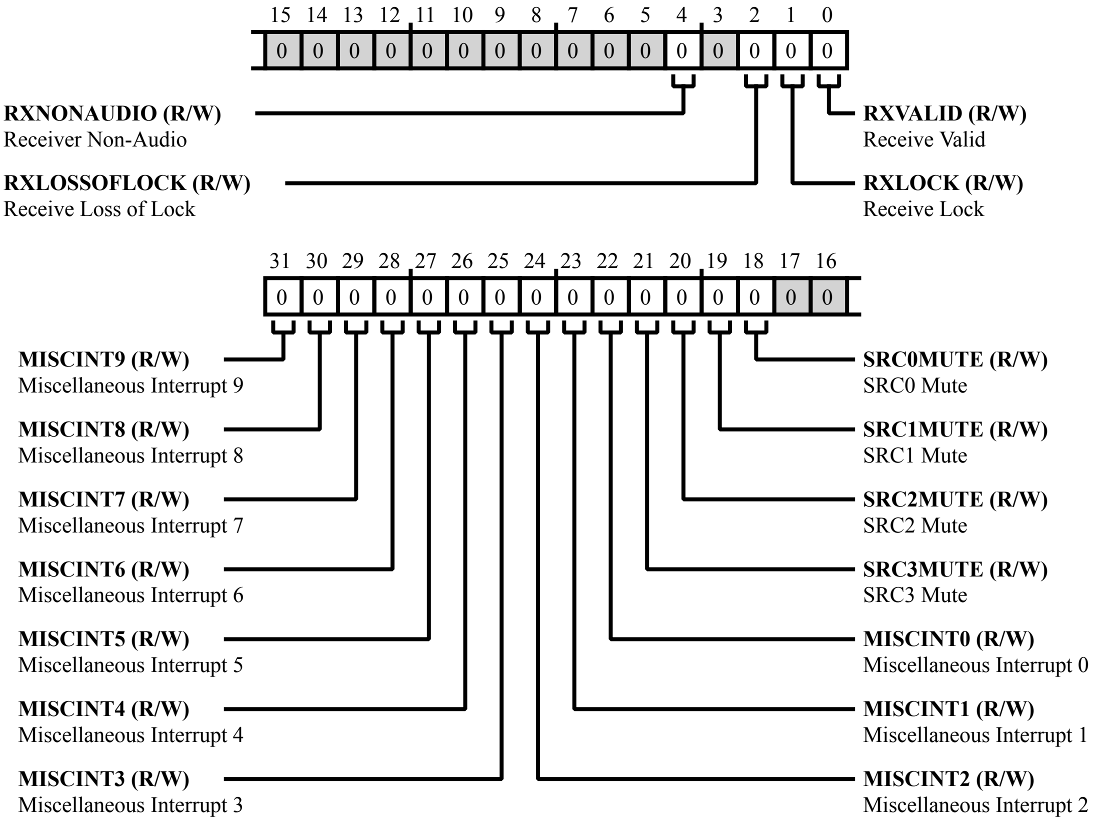

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (R/W)
Receiver Non-Audio
RXLOSSOFLOCK (R/W)
Receive Loss of Lock
RXVALID (R/W)
Receive Valid
RXLOCK (R/W)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (R/W)
Miscellaneous Interrupt 9
MISCINT8 (R/W)
Miscellaneous Interrupt 8
MISCINT7 (R/W)
Miscellaneous Interrupt 7
MISCINT6 (R/W)
Miscellaneous Interrupt 6
MISCINT5 (R/W)
Miscellaneous Interrupt 5
MISCINT4 (R/W)
Miscellaneous Interrupt 4
MISCINT3 (R/W)
Miscellaneous Interrupt 3
SRC0MUTE (R/W)
SRC0 Mute
SRC1MUTE (R/W)
SRC1 Mute
SRC2MUTE (R/W)
SRC2 Mute
SRC3MUTE (R/W)
SRC3 Mute
MISCINT0 (R/W)
Miscellaneous Interrupt 0
MISCINT1 (R/W)
Miscellaneous Interrupt 1
MISCINT2 (R/W)
Miscellaneous Interrupt 2

Figure 33-35: DAI_IMSK_PRI Register Diagram

Table 33-46: DAI_IMSK_PRI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. If the `DAI_IMSK_PRI.MISCINT9` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT9` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. If the `DAI_IMSK_PRI.MISCINT8` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT8` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.

Table 33-46: DAI_IMSK_PRI Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. If the `DAI_IMSK_PRI.MISCINT7` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT7` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. If the `DAI_IMSK_PRI.MISCINT6` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT6` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. If the `DAI_IMSK_PRI.MISCINT5` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT5` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. If the `DAI_IMSK_PRI.MISCINT4` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT4` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. If the `DAI_IMSK_PRI.MISCINT3` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT3` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. If the `DAI_IMSK_PRI.MISCINT2` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT2` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. If the `DAI_IMSK_PRI.MISCINT1` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT1` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. If the `DAI_IMSK_PRI.MISCINT0` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.MISCINT0` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
21 (R/W)	SRC3MUTE	SRC3 Mute. If the `DAI_IMSK_PRI.SRC3MUTE` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.SRC3MUTE` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.

Table 33-46: DAI_IMSK_PRI Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	SRC2MUTE	SRC2 Mute. If the `DAI_IMSK_PRI.SRC2MUTE` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.SRC2MUTE` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
19 (R/W)	SRC1MUTE	SRC1 Mute. If the `DAI_IMSK_PRI.SRC1MUTE` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.SRC1MUTE` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
18 (R/W)	SRC0MUTE	SRC0 Mute. If the `DAI_IMSK_PRI.SRC0MUTE` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.SRC0MUTE` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
4 (R/W)	RXNONAUDIO	Receiver Non-Audio. If the `DAI_IMSK_PRI.RXNONAUDIO` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.RXNONAUDIO` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IMSK_PRI.RXLOSSOFLOCK` bit masks or unmasks the emphasis loss of lock interrupt for the falling edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver looses the lock of biphase stream.
1 (R/W)	RXLOCK	Receive Lock. If the `DAI_IMSK_PRI.RXLOCK` bit is cleared (=0), the interrupt is mapped to INTR_DAI_IRQL signal from the SEC. If the `DAI_IMSK_PRI.RXLOCK` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.
0 (R/W)	RXVALID	Receive Valid. If the `DAI_IMSK_PRI.RXVALID` bit is cleared (=0), the interrupt is mapped to the INTR_DAI_IRQL from the SEC. If the `DAI_IMSK_PRI.RXVALID` bit is set (=1) the interrupt is mapped to INTR_DAI_IRQH signal from the SEC.

Rising-Edge Interrupt Mask Register

The `DAI_IMSK_RE` register masks and unmasks interrupts generated on the rising edge of a waveform. Note that any of the Group E signals can be mapped to any of the miscellaneous interrupts (9-0).

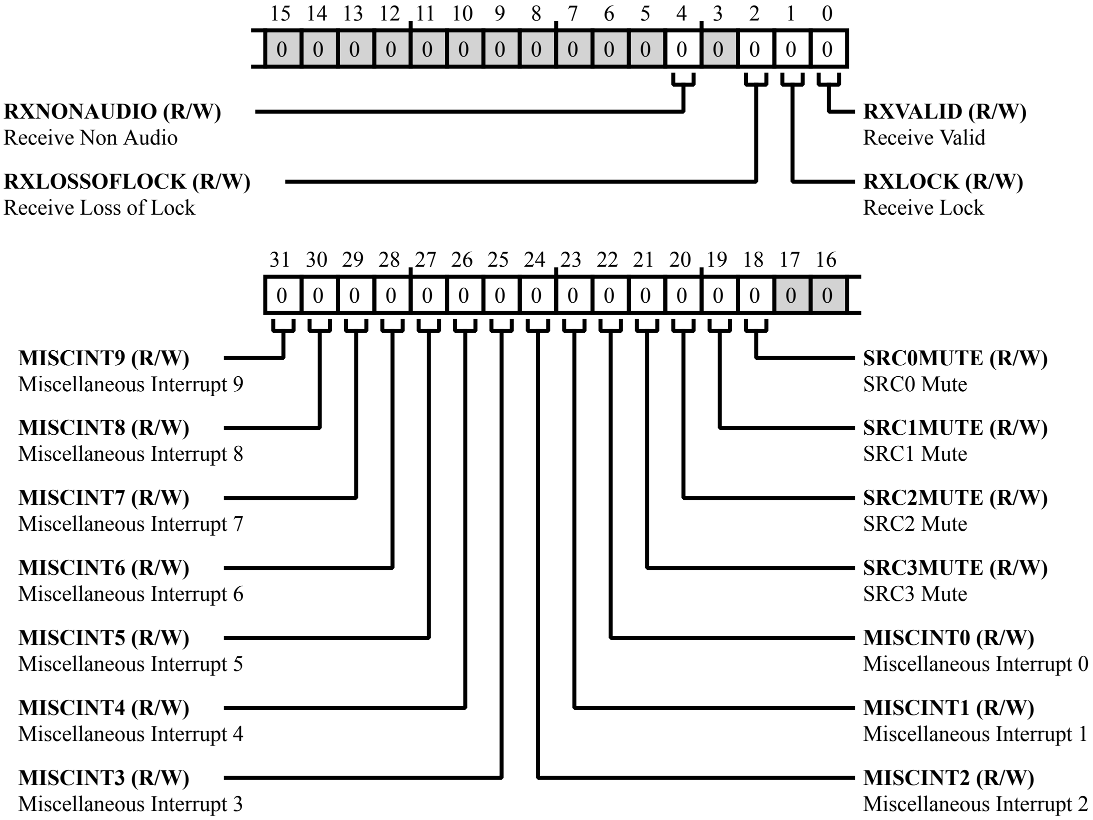

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (R/W)
Receive Non Audio
RXLOSSOFLOCK (R/W)
Receive Loss of Lock
RXVALID (R/W)
Receive Valid
RXLOCK (R/W)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (R/W)
Miscellaneous Interrupt 9
MISCINT8 (R/W)
Miscellaneous Interrupt 8
MISCINT7 (R/W)
Miscellaneous Interrupt 7
MISCINT6 (R/W)
Miscellaneous Interrupt 6
MISCINT5 (R/W)
Miscellaneous Interrupt 5
MISCINT4 (R/W)
Miscellaneous Interrupt 4
MISCINT3 (R/W)
Miscellaneous Interrupt 3
SRC0MUTE (R/W)
SRC0 Mute
SRC1MUTE (R/W)
SRC1 Mute
SRC2MUTE (R/W)
SRC2 Mute
SRC3MUTE (R/W)
SRC3 Mute
MISCINT0 (R/W)
Miscellaneous Interrupt 0
MISCINT1 (R/W)
Miscellaneous Interrupt 1
MISCINT2 (R/W)
Miscellaneous Interrupt 2

Figure 33-36: DAI_IMSK_RE Register Diagram

Table 33-47: DAI_IMSK_RE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	MISCINT9	Miscellaneous Interrupt 9. Setting the `DAI_IMSK_RE.MISCINT9` bit unmasks the interrupt for the rising edge of the routed signal.
30 (R/W)	MISCINT8	Miscellaneous Interrupt 8. Setting the `DAI_IMSK_RE.MISCINT8` bit unmasks the interrupt for the rising edge of the routed signal.
29 (R/W)	MISCINT7	Miscellaneous Interrupt 7. Setting the `DAI_IMSK_RE.MISCINT7` bit unmasks the interrupt for the rising edge of the routed signal.

Table 33-47: DAI_IMSK_RE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	MISCINT6	Miscellaneous Interrupt 6. Setting the `DAI_IMSK_RE.MISCINT6` bit unmasks the interrupt for the rising edge of the routed signal.
27 (R/W)	MISCINT5	Miscellaneous Interrupt 5. Setting the `DAI_IMSK_RE.MISCINT5` bit unmasks the interrupt for the rising edge of the routed signal.
26 (R/W)	MISCINT4	Miscellaneous Interrupt 4. Setting the `DAI_IMSK_RE.MISCINT4` bit unmasks the interrupt for the rising edge of the routed signal.
25 (R/W)	MISCINT3	Miscellaneous Interrupt 3. Setting the `DAI_IMSK_RE.MISCINT3` bit unmasks the interrupt for the rising edge of the routed signal.
24 (R/W)	MISCINT2	Miscellaneous Interrupt 2. Setting the `DAI_IMSK_RE.MISCINT2` bit unmasks the interrupt for the rising edge of the routed signal.
23 (R/W)	MISCINT1	Miscellaneous Interrupt 1. Setting the `DAI_IMSK_RE.MISCINT1` bit unmasks the interrupt for the rising edge of the routed signal.
22 (R/W)	MISCINT0	Miscellaneous Interrupt 0. Setting the `DAI_IMSK_RE.MISCINT0` bit unmasks the interrupt for the rising edge of the routed signal.
21 (R/W)	SRC3MUTE	SRC3 Mute. The `DAI_IMSK_RE.SRC3MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
20 (R/W)	SRC2MUTE	SRC2 Mute. The `DAI_IMSK_RE.SRC2MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
19 (R/W)	SRC1MUTE	SRC1 Mute. The `DAI_IMSK_RE.SRC1MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.

Table 33-47: DAI_IMSK_RE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SRC0MUTE	SRC0 Mute. The `DAI_IMSK_RE.SRC0MUTE` bit masks or unmasks the corresponding SRC mute out interrupt for the rising edge of this interrupt. This interrupt can be generated either entering mute, exiting muting or both.
4 (R/W)	RXNONAUDIO	Receive Non Audio. The `DAI_IMSK_RE.RXNONAUDIO` bit masks or unmasks the non audio frame mode interrupt for the rising edge of this interrupt. If the channel status indicates non-PCM audio, the NONAUDIO bit flag is set.
2 (R/W)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IMSK_RE.RXLOSSOFLOCK` bit masks or unmasks the emphasis loss of lock interrupt for the rising edge of this interrupt. The loss of lock status is set by the S/PDIF receiver if receiver looses the lock of biphase stream.
1 (R/W)	RXLOCK	Receive Lock. The `DAI_IMSK_RE.RXLOCK` bit masks or unmasks the S/PDIF receiver lock for the rising edge of this interrupt. This interrupt occurs when the S/PDIF receiver locks to the S/PDIF stream.
0 (R/W)	RXVALID	Receive Valid. Setting the `DAI_IMSK_RE.RXVALID` bit unmasks the interrupt for the rising edge of the routed signal. This interrupt is set based on whether data received by the S/PDIF is linear PCM or non-linear audio data.

High Priority Interrupt Latch Register

The DAI_IRPTL_H register holds the high priority latched interrupt status for interrupt requests that have been unmasked (enabled) by the DAI_IMSK_FE, DAI_IMSK_RE registers and mapped to the INTR_DAI_IRQH signal in the SEC by DAI_IMSK_PRI registers. If a bit in this register is already set and the corresponding interrupt is masked in the DAI_IMSK_FE, DAI_IMSK_RE or DAI_IMSK_PRI registers, the latch holds its old value, leaving the interrupt asserted until it is reset by software with a W1C operation.

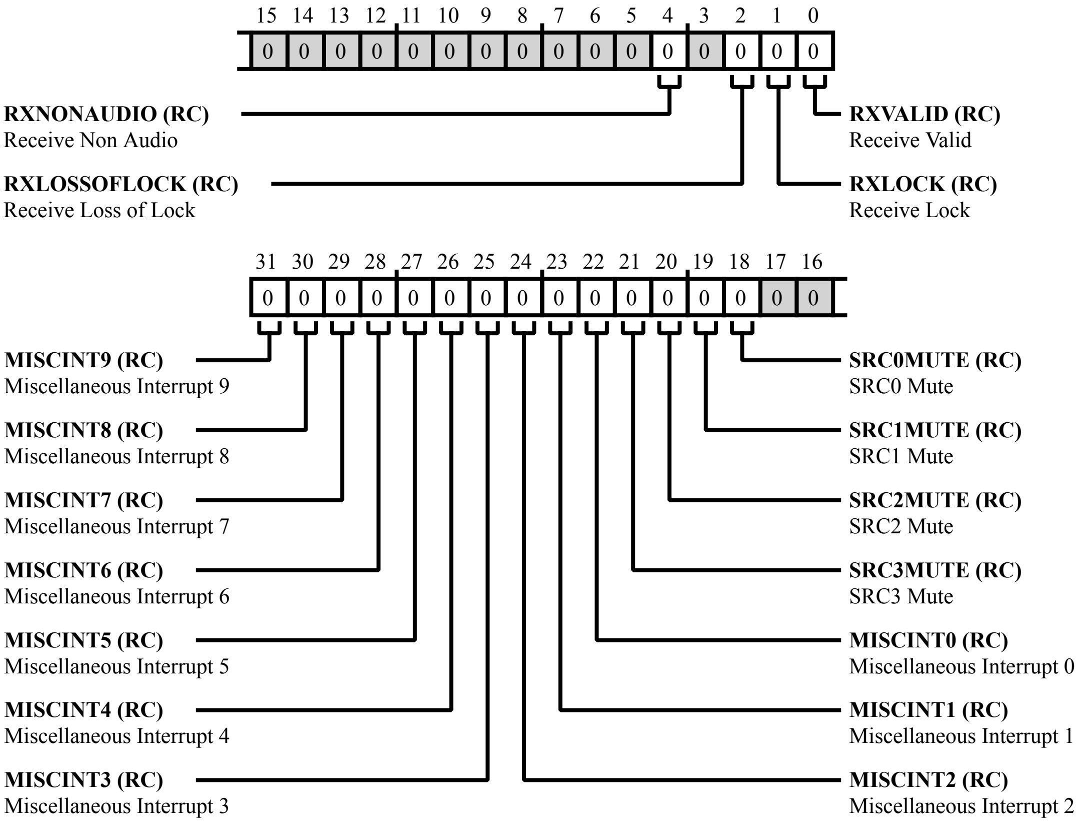

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (RC)
Receive Non Audio
RXLOSSOFLOCK (RC)
Receive Loss of Lock
RXVALID (RC)
Receive Valid
RXLOCK (RC)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (RC)
Miscellaneous Interrupt 9
MISCINT8 (RC)
Miscellaneous Interrupt 8
MISCINT7 (RC)
Miscellaneous Interrupt 7
MISCINT6 (RC)
Miscellaneous Interrupt 6
MISCINT5 (RC)
Miscellaneous Interrupt 5
MISCINT4 (RC)
Miscellaneous Interrupt 4
MISCINT3 (RC)
Miscellaneous Interrupt 3
SRC0MUTE (RC)
SRC0 Mute
SRC1MUTE (RC)
SRC1 Mute
SRC2MUTE (RC)
SRC2 Mute
SRC3MUTE (RC)
SRC3 Mute
MISCINT0 (RC)
Miscellaneous Interrupt 0
MISCINT1 (RC)
Miscellaneous Interrupt 1
MISCINT2 (RC)
Miscellaneous Interrupt 2

Figure 33-37: DAI_IRPTL_H Register Diagram

Table 33-48: DAI_IRPTL_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (RC/NW)	MISCINT9	Miscellaneous Interrupt 9. The DAI_IRPTL_H.MISCINT9 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.
30 (RC/NW)	MISCINT8	Miscellaneous Interrupt 8. The DAI_IRPTL_H.MISCINT8 bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the DAI_IMSK_PRI register.

Table 33-48: DAI_IRPTL_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (RC/NW)	MISCINT7	Miscellaneous Interrupt 7. The `DAI_IRPTL_H.MISCINT7` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
28 (RC/NW)	MISCINT6	Miscellaneous Interrupt 6. The `DAI_IRPTL_H.MISCINT6` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
27 (RC/NW)	MISCINT5	Miscellaneous Interrupt 5. The `DAI_IRPTL_H.MISCINT5` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
26 (RC/NW)	MISCINT4	Miscellaneous Interrupt 4. The `DAI_IRPTL_H.MISCINT4` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
25 (RC/NW)	MISCINT3	Miscellaneous Interrupt 3. The `DAI_IRPTL_H.MISCINT3` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
24 (RC/NW)	MISCINT2	Miscellaneous Interrupt 2. The `DAI_IRPTL_H.MISCINT2` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
23 (RC/NW)	MISCINT1	Miscellaneous Interrupt 1. The `DAI_IRPTL_H.MISCINT1` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
22 (RC/NW)	MISCINT0	Miscellaneous Interrupt 0. The `DAI_IRPTL_H.MISCINT0` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
21 (RC/NW)	SRC3MUTE	SRC3 Mute. The `DAI_IRPTL_H.SRC3MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
20 (RC/NW)	SRC2MUTE	SRC2 Mute. The `DAI_IRPTL_H.SRC2MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
19 (RC/NW)	SRC1MUTE	SRC1 Mute. The `DAI_IRPTL_H.SRC1MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.

Table 33-48: DAI_IRPTL_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RC/NW)	SRC0MUTE	SRC0 Mute. The `DAI_IRPTL_H.SRC0MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
4 (RC/NW)	RXNONAUDIO	Receive Non Audio. The `DAI_IRPTL_H.RXNONAUDIO` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
2 (RC/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IRPTL_H.RXLOSSOFLOCK` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
1 (RC/NW)	RXLOCK	Receive Lock. The `DAI_IRPTL_H.RXLOCK` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.
0 (RC/NW)	RXVALID	Receive Valid. The `DAI_IRPTL_H.RXVALID` bit is set if the interrupt is mapped to the INTR_DAI_IRQH signal in the SEC using the `DAI_IMSK_PRI` register.

Shadow High Priority Interrupt Latch Register

The `DAI_IRPTL_HS` register is the shadow register of the `DAI_IRPTL_H` register. Its content is the same as the `DAI_IRPTL_H` register. Reading the `DAI_IRPTL_HS` register does not affect its contents while reading the contents of the `DAI_IRPTL_H` registers clears it.

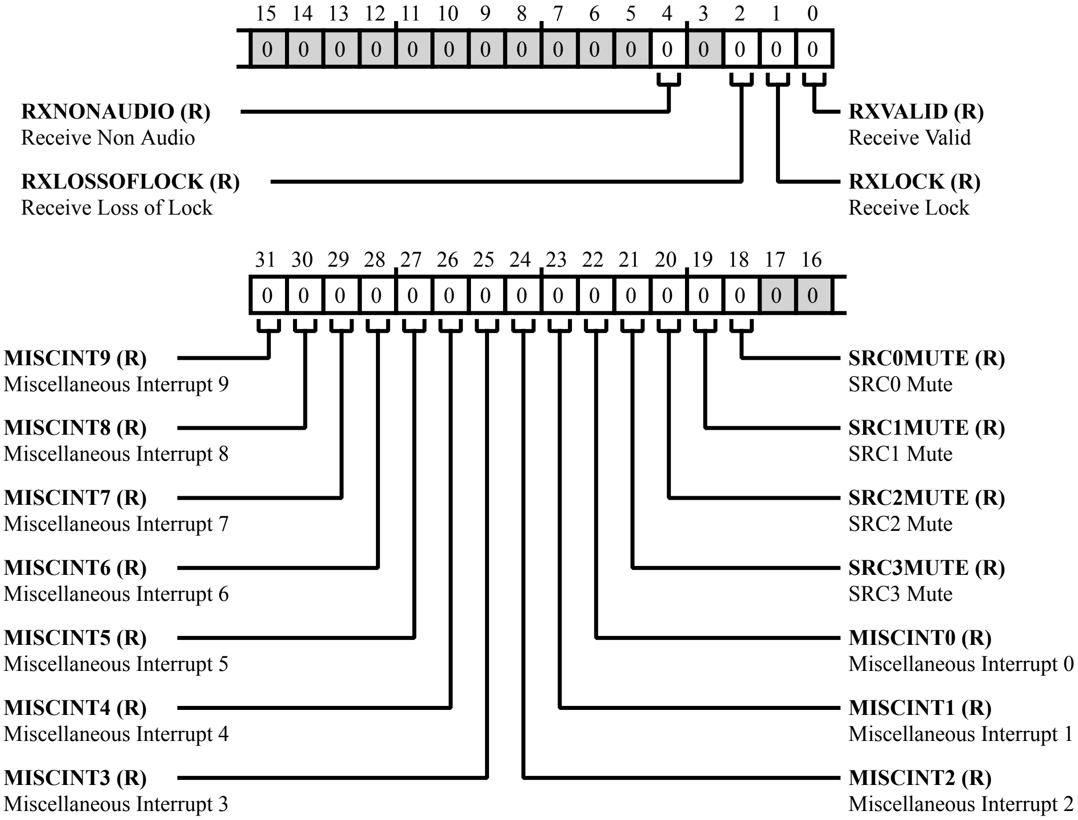

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (R)
Receive Non Audio
RXLOSSOFLOCK (R)
Receive Loss of Lock
RXVALID (R)
Receive Valid
RXLOCK (R)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (R)
Miscellaneous Interrupt 9
MISCINT8 (R)
Miscellaneous Interrupt 8
MISCINT7 (R)
Miscellaneous Interrupt 7
MISCINT6 (R)
Miscellaneous Interrupt 6
MISCINT5 (R)
Miscellaneous Interrupt 5
MISCINT4 (R)
Miscellaneous Interrupt 4
MISCINT3 (R)
Miscellaneous Interrupt 3
SRC0MUTE (R)
SRC0 Mute
SRC1MUTE (R)
SRC1 Mute
SRC2MUTE (R)
SRC2 Mute
SRC3MUTE (R)
SRC3 Mute
MISCINT0 (R)
Miscellaneous Interrupt 0
MISCINT1 (R)
Miscellaneous Interrupt 1
MISCINT2 (R)
Miscellaneous Interrupt 2

Figure 33-38: DAI_IRPTL_HS Register Diagram

Table 33-49: DAI_IRPTL_HS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MISCINT9	Miscellaneous Interrupt 9. The `DAI_IRPTL_HS.MISCINT9` bit is the shadow of the `DAI_IRPTL_H.MISCINT9` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT9` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT9` bit clears it.
30 (R/NW)	MISCINT8	Miscellaneous Interrupt 8. The `DAI_IRPTL_HS.MISCINT8` bit is the shadow of the `DAI_IRPTL_H.MISCINT8` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT8` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT8` bit clears it.

Table 33-49: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	MISCINT7	Miscellaneous Interrupt 7. The `DAI_IRPTL_HS.MISCINT7` bit is the shadow of the `DAI_IRPTL_H.MISCINT7` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT7` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT7` bit clears it.
28 (R/NW)	MISCINT6	Miscellaneous Interrupt 6. The `DAI_IRPTL_HS.MISCINT6` bit is the shadow of the `DAI_IRPTL_H.MISCINT6` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT6` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT6` bit clears it.
27 (R/NW)	MISCINT5	Miscellaneous Interrupt 5. The `DAI_IRPTL_HS.MISCINT5` bit is the shadow of the `DAI_IRPTL_H.MISCINT5` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT5` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT5` bit clears it.
26 (R/NW)	MISCINT4	Miscellaneous Interrupt 4. The `DAI_IRPTL_HS.MISCINT4` bit is the shadow of the `DAI_IRPTL_H.MISCINT4` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT4` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT4` bit clears it.
25 (R/NW)	MISCINT3	Miscellaneous Interrupt 3. The `DAI_IRPTL_HS.MISCINT3` bit is the shadow of the `DAI_IRPTL_H.MISCINT3` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT3` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT3` bit clears it.
24 (R/NW)	MISCINT2	Miscellaneous Interrupt 2. The `DAI_IRPTL_HS.MISCINT2` bit is the shadow of the `DAI_IRPTL_H.MISCINT2` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT2` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT2` bit clears it.
23 (R/NW)	MISCINT1	Miscellaneous Interrupt 1. The `DAI_IRPTL_HS.MISCINT1` bit is the shadow of the `DAI_IRPTL_H.MISCINT1` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT1` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT1` bit clears it.

Table 33-49: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/NW)	MISCINT0	Miscellaneous Interrupt 0. The `DAI_IRPTL_HS.MISCINT0` bit is the shadow of the `DAI_IRPTL_H.MISCINT0` bit and contains the same content. Reading the `DAI_IRPTL_HS.MISCINT0` bit does not affect its contents while reading the `DAI_IRPTL_H.MISCINT0` bit clears it.
21 (R/NW)	SRC3MUTE	SRC3 Mute. The `DAI_IRPTL_HS.SRC3MUTE` bit is the shadow of the `DAI_IRPTL_H.SRC3MUTE` bit and contains the same content. Reading the `DAI_IRPTL_HS.SRC3MUTE` bit does not affect its contents while reading the `DAI_IRPTL_H.SRC3MUTE` bit clears it.
20 (R/NW)	SRC2MUTE	SRC2 Mute. The `DAI_IRPTL_HS.SRC2MUTE` bit is the shadow of the `DAI_IRPTL_H.SRC2MUTE` bit and contains the same content. Reading the `DAI_IRPTL_HS.SRC2MUTE` bit does not affect its contents while reading the `DAI_IRPTL_H.SRC2MUTE` bit clears it.
19 (R/NW)	SRC1MUTE	SRC1 Mute. The `DAI_IRPTL_HS.SRC1MUTE` bit is the shadow of the `DAI_IRPTL_H.SRC1MUTE` bit and contains the same content. Reading the `DAI_IRPTL_HS.SRC1MUTE` bit does not affect its contents while reading the `DAI_IRPTL_H.SRC1MUTE` bit clears it.
18 (R/NW)	SRC0MUTE	SRC0 Mute. The `DAI_IRPTL_HS.SRC0MUTE` bit is the shadow of the `DAI_IRPTL_H.SRC0MUTE` bit and contains the same content. Reading the `DAI_IRPTL_HS.SRC0MUTE` bit does not affect its contents while reading the `DAI_IRPTL_H.SRC0MUTE` bit clears it.
4 (R/NW)	RXNONAUDIO	Receive Non Audio. The `DAI_IRPTL_HS.RXNONAUDIO` bit is the shadow of `DAI_IRPTL_H.RXNONAUDIO` bit and contains the same content. Reading this bit does not affect its content.
2 (R/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IRPTL_HS.RXLOSSOFLOCK` bit is the shadow of `DAI_IRPTL_H.RXLOSSOFLOCK` bit and contains the same content. Reading this bit does not affect its content.
1 (R/NW)	RXLOCK	Receive Lock. The `DAI_IRPTL_HS.RXLOCK` bit is the shadow of `DAI_IRPTL_H.RXLOCK` bit and contains the same content. Reading this bit does not affect its content.

Table 33-49: DAI_IRPTL_HS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	RXVALID	Receive Valid. Setting the `DAI_IRPTL_HS.RXVALID` bit is the shadow of `DAI_IRPTL_H.RXVALID` bit and contains the same content. Reading this bit does not affect its content.

Low Priority Interrupt Latch Register

The DAI_IRPTL_L register holds the low priority latched interrupt status for interrupt requests that have been unmasked (enabled) by the DAI_IMSK_FE, DAI_IMSK_RE or DAI_IMSK_PRI registers. If a bit in DAI_IRPTL_L is already set and the corresponding interrupt is masked in DAI_IMSK_FE, DAI_IMSK_RE or DAI_IMSK_PRI registers, the latch holds its old value, leaving the interrupt asserted until it is reset by software with a W1C operation.

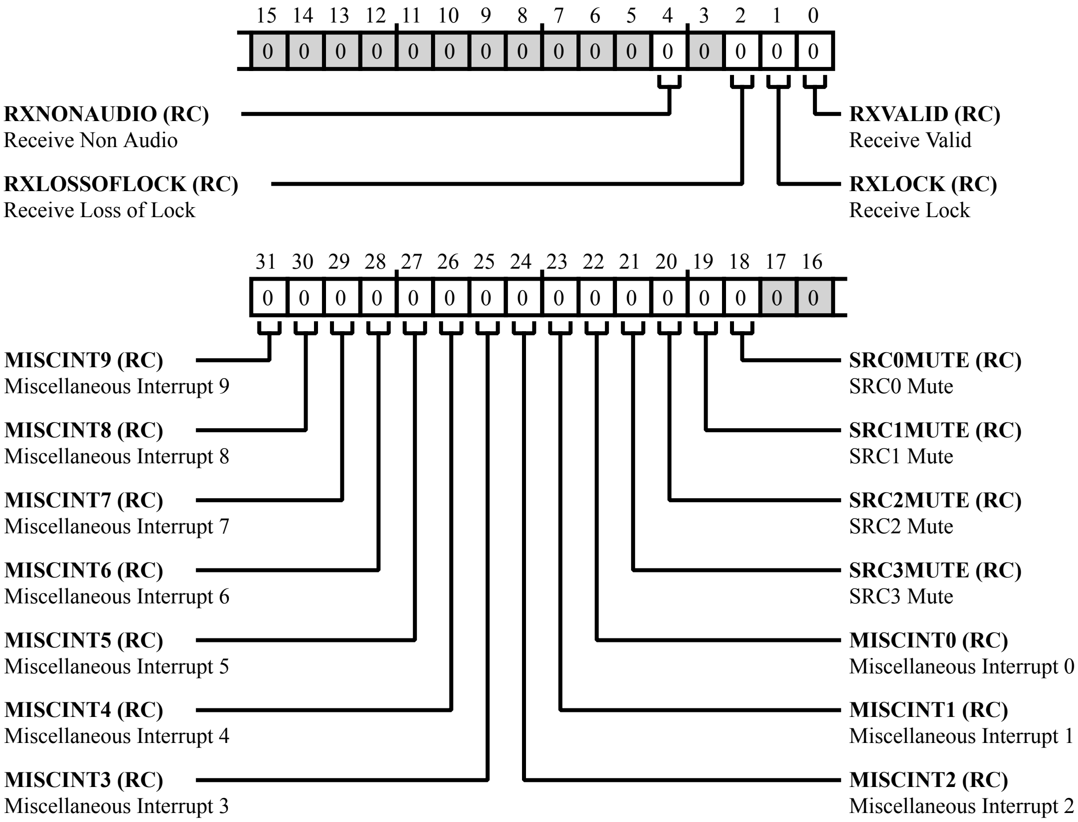

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (RC)
Receive Non Audio
RXLOSSOFLOCK (RC)
Receive Loss of Lock
RXVALID (RC)
Receive Valid
RXLOCK (RC)
Receive Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (RC)
Miscellaneous Interrupt 9
MISCINT8 (RC)
Miscellaneous Interrupt 8
MISCINT7 (RC)
Miscellaneous Interrupt 7
MISCINT6 (RC)
Miscellaneous Interrupt 6
MISCINT5 (RC)
Miscellaneous Interrupt 5
MISCINT4 (RC)
Miscellaneous Interrupt 4
MISCINT3 (RC)
Miscellaneous Interrupt 3
SRC0MUTE (RC)
SRC0 Mute
SRC1MUTE (RC)
SRC1 Mute
SRC2MUTE (RC)
SRC2 Mute
SRC3MUTE (RC)
SRC3 Mute
MISCINT0 (RC)
Miscellaneous Interrupt 0
MISCINT1 (RC)
Miscellaneous Interrupt 1
MISCINT2 (RC)
Miscellaneous Interrupt 2

Figure 33-39: DAI_IRPTL_L Register Diagram

Table 33-50: DAI_IRPTL_L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (RC/NW)	MISCINT9	Miscellaneous Interrupt 9. The DAI_IRPTL_L.MISCINT9 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
30 (RC/NW)	MISCINT8	Miscellaneous Interrupt 8. The DAI_IRPTL_L.MISCINT8 bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.

Table 33-50: DAI_IRPTL_L Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (RC/NW)	MISCINT7	Miscellaneous Interrupt 7. The `DAI_IRPTL_L.MISCINT7` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
28 (RC/NW)	MISCINT6	Miscellaneous Interrupt 6. The `DAI_IRPTL_L.MISCINT6` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
27 (RC/NW)	MISCINT5	Miscellaneous Interrupt 5. The `DAI_IRPTL_L.MISCINT5` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
26 (RC/NW)	MISCINT4	Miscellaneous Interrupt 4. The `DAI_IRPTL_L.MISCINT4` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
25 (RC/NW)	MISCINT3	Miscellaneous Interrupt 3. The `DAI_IRPTL_L.MISCINT3` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
24 (RC/NW)	MISCINT2	Miscellaneous Interrupt 2. The `DAI_IRPTL_L.MISCINT2` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
23 (RC/NW)	MISCINT1	Miscellaneous Interrupt 1. The `DAI_IRPTL_L.MISCINT1` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
22 (RC/NW)	MISCINT0	Miscellaneous Interrupt 0. The `DAI_IRPTL_L.MISCINT0` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
21 (RC/NW)	SRC3MUTE	SRC3 Mute. The `DAI_IRPTL_L.SRC3MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
20 (RC/NW)	SRC2MUTE	SRC2 Mute. The `DAI_IRPTL_L.SRC2MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
19 (RC/NW)	SRC1MUTE	SRC1 Mute. The `DAI_IRPTL_L.SRC1MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.

Table 33-50: DAI_IRPTL_L Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RC/NW)	SRC0MUTE	SRC0 Mute. The `DAI_IRPTL_L.SRC0MUTE` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the `DAI_IMSK_PRI` register.
4 (RC/NW)	RXNONAUDIO	Receive Non Audio. The `DAI_IRPTL_L.RXNONAUDIO` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
2 (RC/NW)	RXLOSSOFLOCK	Receive Loss of Lock. The `DAI_IRPTL_L.RXLOSSOFLOCK` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
1 (RC/NW)	RXLOCK	Receive Lock. The `DAI_IRPTL_L.RXLOCK` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.
0 (RC/NW)	RXVALID	Receive Valid. The `DAI_IRPTL_L.RXVALID` bit is set if the interrupt is mapped to the INTR_DAI_IRQL signal in the SEC using the DAI_IMSK_PRI register.

Shadow Low Priority Interrupt Latch Register

The `DAI_IRPTL_LS` register is the shadow register of the `DAI_IRPTL_L` register. Its content is the same as the `DAI_IRPTL_L` register. Reading the `DAI_IRPTL_LS` register does not affect its contents while reading the contents of the `DAI_IRPTL_L` registers clears it.

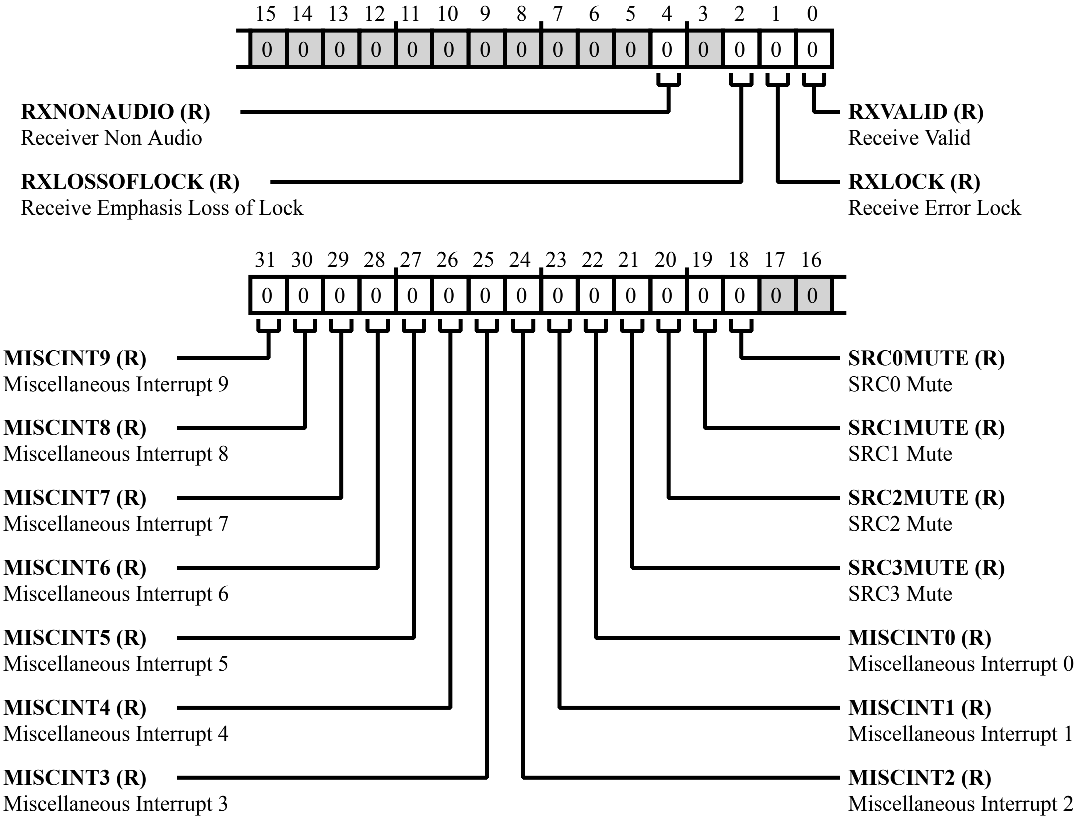

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXNONAUDIO (R)
Receiver Non Audio
RXLOSSOFLOCK (R)
Receive Emphasis Loss of Lock
RXVALID (R)
Receive Valid
RXLOCK (R)
Receive Error Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MISCINT9 (R)
Miscellaneous Interrupt 9
MISCINT8 (R)
Miscellaneous Interrupt 8
MISCINT7 (R)
Miscellaneous Interrupt 7
MISCINT6 (R)
Miscellaneous Interrupt 6
MISCINT5 (R)
Miscellaneous Interrupt 5
MISCINT4 (R)
Miscellaneous Interrupt 4
MISCINT3 (R)
Miscellaneous Interrupt 3
SRC0MUTE (R)
SRC0 Mute
SRC1MUTE (R)
SRC1 Mute
SRC2MUTE (R)
SRC2 Mute
SRC3MUTE (R)
SRC3 Mute
MISCINT0 (R)
Miscellaneous Interrupt 0
MISCINT1 (R)
Miscellaneous Interrupt 1
MISCINT2 (R)
Miscellaneous Interrupt 2

Figure 33-40: DAI_IRPTL_LS Register Diagram

Table 33-51: DAI_IRPTL_LS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MISCINT9	Miscellaneous Interrupt 9. The `DAI_IRPTL_LS.MISCINT9` bit is the shadow of the `DAI_IRPTL_L.MISCINT9` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT9` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT9` bit clears it.
30 (R/NW)	MISCINT8	Miscellaneous Interrupt 8. The `DAI_IRPTL_LS.MISCINT8` bit is the shadow of the `DAI_IRPTL_L.MISCINT8` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT8` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT8` bit clears it.

Table 33-51: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	MISCINT7	Miscellaneous Interrupt 7. The `DAI_IRPTL_LS.MISCINT7` bit is the shadow of the `DAI_IRPTL_L.MISCINT7` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT7` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT7` bit clears it.
28 (R/NW)	MISCINT6	Miscellaneous Interrupt 6. The `DAI_IRPTL_LS.MISCINT6` bit is the shadow of the `DAI_IRPTL_L.MISCINT6` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT6` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT6` bit clears it.
27 (R/NW)	MISCINT5	Miscellaneous Interrupt 5. The `DAI_IRPTL_LS.MISCINT5` bit is the shadow of the `DAI_IRPTL_L.MISCINT5` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT5` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT5` bit clears it.
26 (R/NW)	MISCINT4	Miscellaneous Interrupt 4. The `DAI_IRPTL_LS.MISCINT4` bit is the shadow of the `DAI_IRPTL_L.MISCINT4` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT4` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT4` bit clears it.
25 (R/NW)	MISCINT3	Miscellaneous Interrupt 3. The `DAI_IRPTL_LS.MISCINT3` bit is the shadow of the `DAI_IRPTL_L.MISCINT3` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT3` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT3` bit clears it.
24 (R/NW)	MISCINT2	Miscellaneous Interrupt 2. The `DAI_IRPTL_LS.MISCINT2` bit is the shadow of the `DAI_IRPTL_L.MISCINT2` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT2` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT2` bit clears it.
23 (R/NW)	MISCINT1	Miscellaneous Interrupt 1. The `DAI_IRPTL_LS.MISCINT1` bit is the shadow of the `DAI_IRPTL_L.MISCINT1` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT1` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT1` bit clears it.

Table 33-51: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/NW)	MISCINT0	Miscellaneous Interrupt 0. The `DAI_IRPTL_LS.MISCINT0` bit is the shadow of the `DAI_IRPTL_L.MISCINT0` bit and contains the same content. Reading the `DAI_IRPTL_LS.MISCINT0` bit does not affect its contents while reading the `DAI_IRPTL_L.MISCINT0` bit clears it.
21 (R/NW)	SRC3MUTE	SRC3 Mute. The `DAI_IRPTL_LS.SRC3MUTE` bit is the shadow of the `DAI_IRPTL_L.SRC3MUTE` bit and contains the same content. Reading the `DAI_IRPTL_LS.SRC3MUTE` bit does not affect its contents while reading the `DAI_IRPTL_L.SRC3MUTE` bit clears it.
20 (R/NW)	SRC2MUTE	SRC2 Mute. The `DAI_IRPTL_LS.SRC2MUTE` bit is the shadow of the `DAI_IRPTL_L.SRC2MUTE` bit and contains the same content. Reading the `DAI_IRPTL_LS.SRC2MUTE` bit does not affect its contents while reading the `DAI_IRPTL_L.SRC2MUTE` bit clears it.
19 (R/NW)	SRC1MUTE	SRC1 Mute. The `DAI_IRPTL_LS.SRC1MUTE` bit is the shadow of the `DAI_IRPTL_L.SRC1MUTE` bit and contains the same content. Reading the `DAI_IRPTL_LS.SRC1MUTE` bit does not affect its contents while reading the `DAI_IRPTL_L.SRC1MUTE` bit clears it.
18 (R/NW)	SRC0MUTE	SRC0 Mute. The `DAI_IRPTL_LS.SRC0MUTE` bit is the shadow of the `DAI_IRPTL_L.SRC0MUTE` bit and contains the same content. Reading the `DAI_IRPTL_LS.SRC0MUTE` bit does not affect its contents while reading the `DAI_IRPTL_L.SRC0MUTE` bit clears it.
4 (R/NW)	RXNONAUDIO	Receiver Non Audio. The `DAI_IRPTL_LS.RXNONAUDIO` bit is the shadow of the `DAI_IRPTL_L.RXNONAUDIO` bit and contains the same content. Reading the `DAI_IRPTL_LS.RXNONAUDIO` bit does not affect its contents while reading the `DAI_IRPTL_L.RXNONAUDIO` bit clears it.
2 (R/NW)	RXLOSSOFLOCK	Receive Emphasis Loss of Lock. The `DAI_IRPTL_LS.RXLOSSOFLOCK` bit is the shadow of `DAI_IRPTL_L.RXLOSSOFLOCK` bit and contains the same content. Reading this bit does not affect its content. Reading the `DAI_IRPTL_L.RXLOSSOFLOCK` bit clears it.
1 (R/NW)	RXLOCK	Receive Error Lock. The `DAI_IRPTL_LS.RXLOCK` bit is the shadow of `DAI_IRPTL_L.RXLOCK` bit and contains the same content. Reading this bit does not affect its content. Reading the `DAI_IRPTL_L.RXLOCK` bit clears it.

Table 33-51: DAI_IRPTL_LS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	RXVALID	Receive Valid. The `DAI_IRPTL_LS.RXVALID` bit is the shadow of `DAI_IRPTL_L.RXVALID` bit and contains the same content. Reading this bit does not affect its content. Reading the `DAI_IRPTL_L.RXVALID` bit clears it.

Miscellaneous Control Register 0

The DAI_MISC0 register allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion. This register belongs to group E which routes control signals and provides a means of connecting signals between groups.

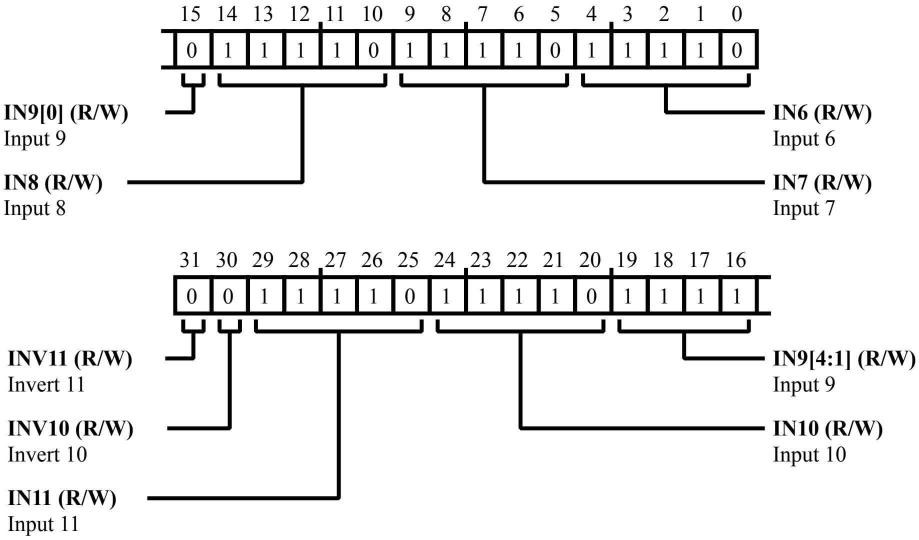

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN9[0] (R/W)
Input 9
IN8 (R/W)
Input 8
IN6 (R/W)
Input 6
IN7 (R/W)
Input 7
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
INV11 (R/W)
Invert 11
INV10 (R/W)
Invert 10
IN11 (R/W)
Input 11
IN9[4:1] (R/W)
Input 9
IN10 (R/W)
Input 10

Figure 33-41: DAI_MISC0 Register Diagram

Table 33-52: DAI_MISC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	INV11	Invert 11. The `DAI_MISC0.INV11` bit field inverts miscellaneous A5 input.
30 (R/W)	INV10	Invert 10. The `DAI_MISC0.INV10` bit field inverts miscellaneous A4 input.
29:25 (R/W)	IN11	Input 11. The `DAI_MISC0.IN11` bit field configures miscellaneous A5 input.
24:20 (R/W)	IN10	Input 10. The `DAI_MISC0.IN10` bit field configures miscellaneous A4 input.
19:15 (R/W)	IN9	Input 9. The `DAI_MISC0.IN9` bit field configures miscellaneous A3 input/miscellaneous Interrupt 9(DAI interrupt 31).
14:10 (R/W)	IN8	Input 8. The `DAI_MISC0.IN8` bit field configures miscellaneous A2 input/miscellaneous Interrupt 8(DAI interrupt 30).

Table 33-52: DAI_MISC0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:5 (R/W)	IN7	Input 7. The `DAI_MISC0.IN7` bit field configures miscellaneous A1 input/miscellaneous Interrupt 7(DAI interrupt 29).
4:0 (R/W)	IN6	Input 6. The `DAI_MISC0.IN6` bit field configures miscellaneous A0 input/miscellaneous Interrupt 6(DAI interrupt 28).

Miscellaneous Control Register 1

The DAI_MISC1 register allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion. This register belongs to group E which routes control signals and provides a means of connecting signals between groups.

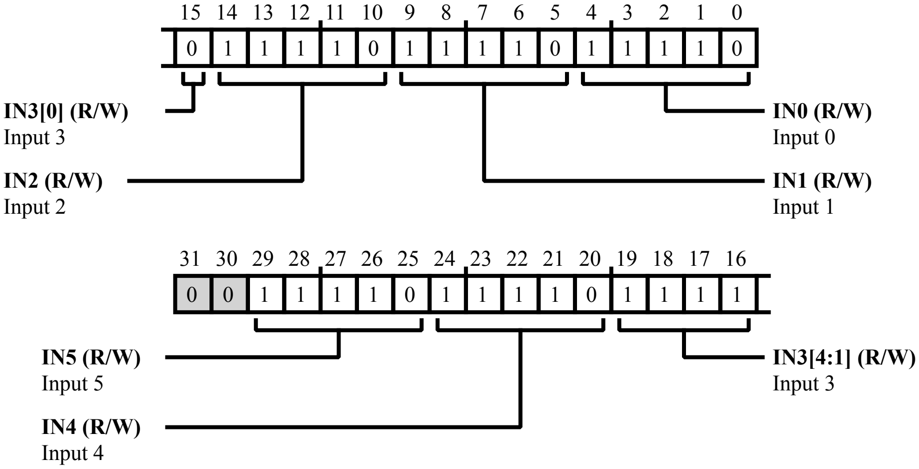

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0
IN3[0] (R/W)
Input 3
IN2 (R/W)
Input 2
IN0 (R/W)
Input 0
IN1 (R/W)
Input 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
IN5 (R/W)
Input 5
IN4 (R/W)
Input 4
IN3[4:1] (R/W)
Input 3

Figure 33-42: DAI_MISC1 Register Diagram

Table 33-53: DAI_MISC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:25 (R/W)	IN5	Input 5. The `DAI_MISC1.IN5` bit field configures miscellaneous Interrupt 5(DAI interrupt 27).
24:20 (R/W)	IN4	Input 4. The `DAI_MISC1.IN4` bit field configures miscellaneous Interrupt 4(DAI interrupt 26).
19:15 (R/W)	IN3	Input 3. The `DAI_MISC1.IN3` bit field configures miscellaneous Interrupt 3(DAI interrupt 25).
14:10 (R/W)	IN2	Input 2. The `DAI_MISC1.IN2` bit field configures miscellaneous Interrupt 2(DAI interrupt 24).
9:5 (R/W)	IN1	Input 1. The `DAI_MISC1.IN1` bit field configures miscellaneous Interrupt 1(DAI interrupt 23).

Table 33-53: DAI_MISC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	IN0	Input 0. The `DAI_MISC1.IN0` bit field configures miscellaneous Interrupt 0(DAI interrupt 22).

Pin Buffer Enable Register 0

The `DAI_PBEN0` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

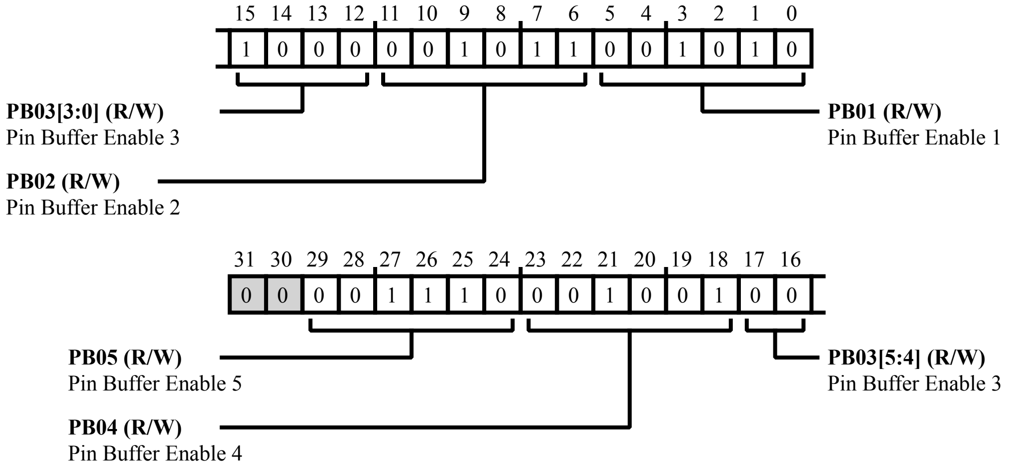

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0
PB03[3:0] (R/W)
Pin Buffer Enable 3
PB02 (R/W)
Pin Buffer Enable 2
PB01 (R/W)
Pin Buffer Enable 1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 1 1 1 0 0 0 1 0 0 1 0 0
PB05 (R/W)
Pin Buffer Enable 5
PB04 (R/W)
Pin Buffer Enable 4
PB03[5:4] (R/W)
Pin Buffer Enable 3

Figure 33-43: DAI_PBEN0 Register Diagram

Table 33-54: DAI_PBEN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB05	Pin Buffer Enable 5. The `DAI_PBEN0.PB05` bit field is the pin buffer enable for DAI port 5.
23:18 (R/W)	PB04	Pin Buffer Enable 4. The `DAI_PBEN0.PB04` bit field is the pin buffer enable for DAI port 4.
17:12 (R/W)	PB03	Pin Buffer Enable 3. The `DAI_PBEN0.PB03` bit field is the pin buffer enable for DAI port 3.
11:6 (R/W)	PB02	Pin Buffer Enable 2. The `DAI_PBEN0.PB02` bit field is the pin buffer enable for DAI port 2.
5:0 (R/W)	PB01	Pin Buffer Enable 1. The `DAI_PBEN0.PB01` bit field is the pin buffer enable for DAI port 1.

Pin Buffer Enable Register 1

The `DAI_PBEN1` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

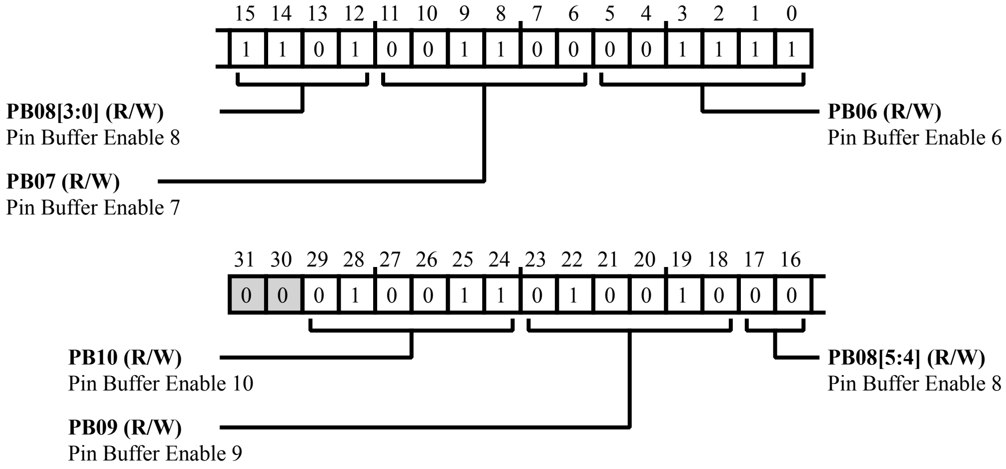

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 0 1 0 0 1 1 0 0 0 0 1 1 1 1
PB08[3:0] (R/W)
Pin Buffer Enable 8
PB07 (R/W)
Pin Buffer Enable 7
PB06 (R/W)
Pin Buffer Enable 6
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 0
PB10 (R/W)
Pin Buffer Enable 10
PB09 (R/W)
Pin Buffer Enable 9
PB08[5:4] (R/W)
Pin Buffer Enable 8

Figure 33-44: DAI_PBEN1 Register Diagram

Table 33-55: DAI_PBEN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB10	Pin Buffer Enable 10. The `DAI_PBEN1.PB10` bit field is the pin buffer enable for DAI port 10.
23:18 (R/W)	PB09	Pin Buffer Enable 9. The `DAI_PBEN1.PB09` bit field is the pin buffer enable for DAI port 9.
17:12 (R/W)	PB08	Pin Buffer Enable 8. The `DAI_PBEN1.PB08` bit field is the pin buffer enable for DAI port 8.
11:6 (R/W)	PB07	Pin Buffer Enable 7. The `DAI_PBEN1.PB07` bit field is the pin buffer enable for DAI port 7.
5:0 (R/W)	PB06	Pin Buffer Enable 6. The `DAI_PBEN1.PB06` bit field is the pin buffer enable for DAI port 6.

Pin Buffer Enable Register 2

The `DAI_PBEN2` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

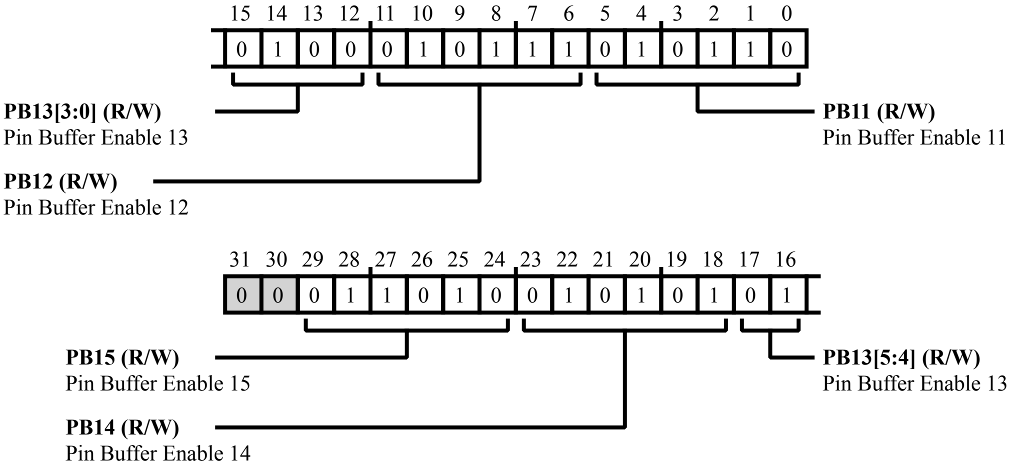

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 0
PB13[3:0] (R/W)
Pin Buffer Enable 13
PB12 (R/W)
Pin Buffer Enable 12
PB11 (R/W)
Pin Buffer Enable 11
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1
PB15 (R/W)
Pin Buffer Enable 15
PB14 (R/W)
Pin Buffer Enable 14
PB13[5:4] (R/W)
Pin Buffer Enable 13

Figure 33-45: DAI_PBEN2 Register Diagram

Table 33-56: DAI_PBEN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB15	Pin Buffer Enable 15. The `DAI_PBEN2.PB15` bit field is the pin buffer enable for DAI port 15.
23:18 (R/W)	PB14	Pin Buffer Enable 14. The `DAI_PBEN2.PB14` bit field is the pin buffer enable for DAI port 14.
17:12 (R/W)	PB13	Pin Buffer Enable 13. The `DAI_PBEN2.PB13` bit field is the pin buffer enable for DAI port 13.
11:6 (R/W)	PB12	Pin Buffer Enable 12. The `DAI_PBEN2.PB12` bit field is the pin buffer enable for DAI port 12.
5:0 (R/W)	PB11	Pin Buffer Enable 11. The `DAI_PBEN2.PB11` bit field is the pin buffer enable for DAI port 11.

Pin Buffer Enable Register 3

The `DAI_PBEN3` register routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input.

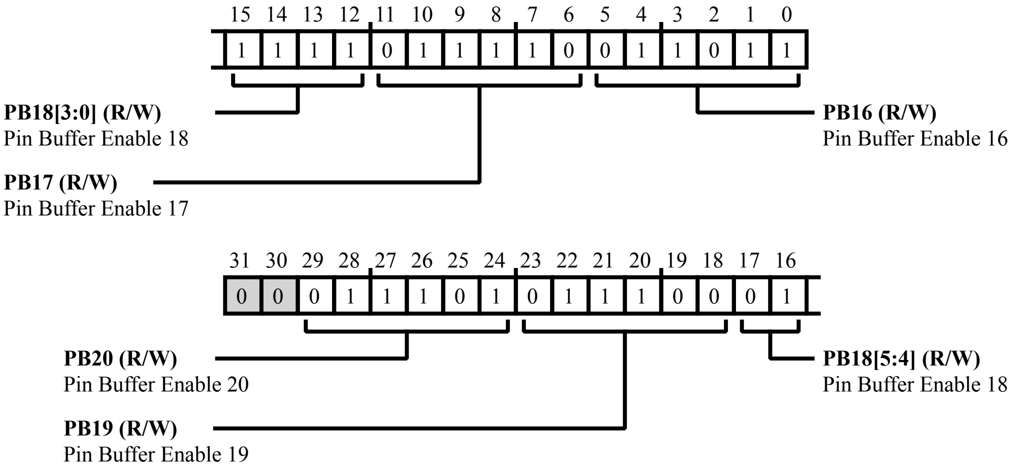

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1
PB18[3:0] (R/W)
Pin Buffer Enable 18
PB17 (R/W)
Pin Buffer Enable 17
PB16 (R/W)
Pin Buffer Enable 16
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1
PB20 (R/W)
Pin Buffer Enable 20
PB19 (R/W)
Pin Buffer Enable 19
PB18[5:4] (R/W)
Pin Buffer Enable 18

Figure 33-46: DAI_PBEN3 Register Diagram

Table 33-57: DAI_PBEN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PB20	Pin Buffer Enable 20. The `DAI_PBEN3.PB20` bit field is the pin buffer enable for DAI port 20.
23:18 (R/W)	PB19	Pin Buffer Enable 19. The `DAI_PBEN3.PB19` bit field is the pin buffer enable for DAI port 19.
17:12 (R/W)	PB18	Pin Buffer Enable 18. The `DAI_PBEN3.PB18` bit field is the pin buffer enable for DAI port 18.
11:6 (R/W)	PB17	Pin Buffer Enable 17. The `DAI_PBEN3.PB17` bit field is the pin buffer enable for DAI port 17.
5:0 (R/W)	PB16	Pin Buffer Enable 16. The `DAI_PBEN3.PB16` bit field is the pin buffer enable for DAI port 16.

Pin Buffer Assignment Register 0

The DAI_PIN0 register routes physical pins that are connected to a bonded pad.

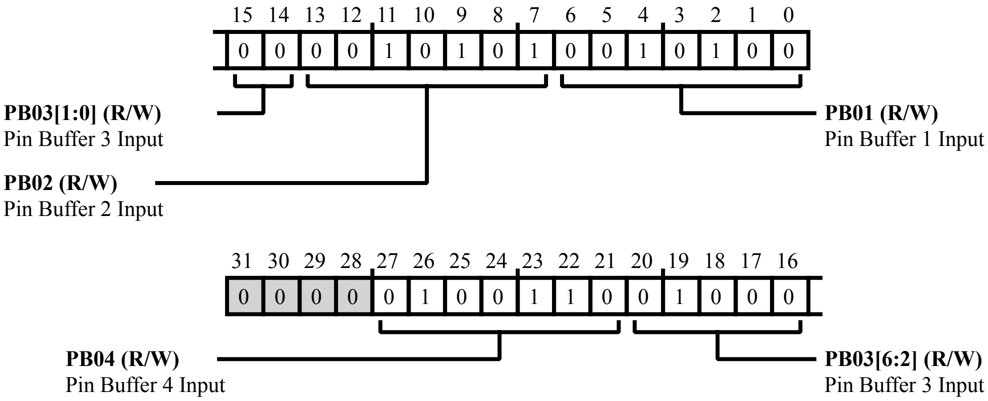

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0
PB03[1:0] (R/W)
Pin Buffer 3 Input
PB02 (R/W)
Pin Buffer 2 Input
PB01 (R/W)
Pin Buffer 1 Input
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0
PB04 (R/W)
Pin Buffer 4 Input
PB03[6:2] (R/W)
Pin Buffer 3 Input

Figure 33-47: DAI_PIN0 Register Diagram

Table 33-58: DAI_PIN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB04	Pin Buffer 4 Input. DAI_PIN0.PB04 holds the Source signal assignment that will be routed to the DAI_PIN0.PB04 Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB03	Pin Buffer 3 Input. DAI_PIN0.PB03 holds the Source signal assignment that will be routed to the DAI_PIN0.PB03 Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB02	Pin Buffer 2 Input. DAI_PIN0.PB02 holds the Source signal assignment that will be routed to the DAI_PIN0.PB02 Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB01	Pin Buffer 1 Input. DAI_PIN0.PB01 holds the Source signal assignment that will be routed to the DAI_PIN0.PB01 Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 1

The `DAI_PIN1` register routes physical pins that are connected to a bonded pad.

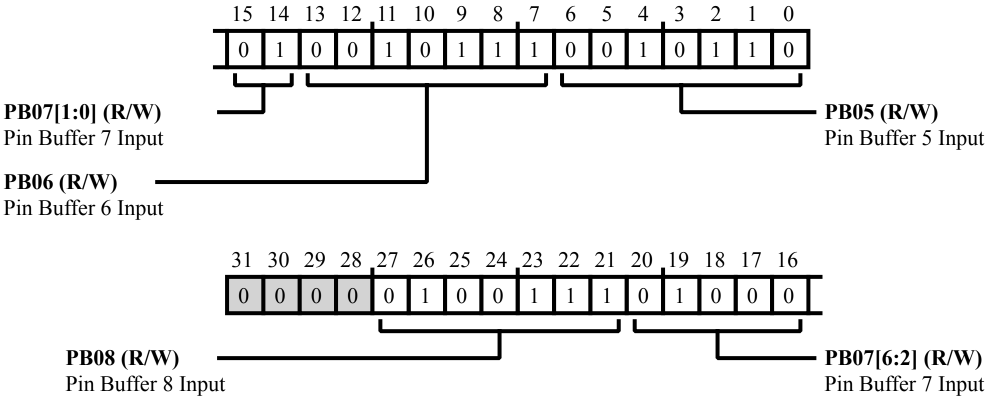

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 1 0 1 1 1 0 0 1 0 1 1 0
PB07[1:0] (R/W)
Pin Buffer 7 Input
PB06 (R/W)
Pin Buffer 6 Input
PB05 (R/W)
Pin Buffer 5 Input
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0
PB08 (R/W)
Pin Buffer 8 Input
PB07[6:2] (R/W)
Pin Buffer 7 Input

Figure 33-48: DAI_PIN1 Register Diagram

Table 33-59: DAI_PIN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB08	Pin Buffer 8 Input. `DAI_PIN1.PB08` holds the Source signal assignment that will be routed to the `DAI_PIN1.PB08` Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB07	Pin Buffer 7 Input. `DAI_PIN1.PB07` holds the Source signal assignment that will be routed to the `DAI_PIN1.PB07` Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB06	Pin Buffer 6 Input. `DAI_PIN1.PB06` holds the Source signal assignment that will be routed to the `DAI_PIN1.PB06` Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB05	Pin Buffer 5 Input. `DAI_PIN1.PB05` holds the Source signal assignment that will be routed to the `DAI_PIN1.PB05` Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 2

The DAI_PIN2 register routes physical pins that are connected to a bonded pad.

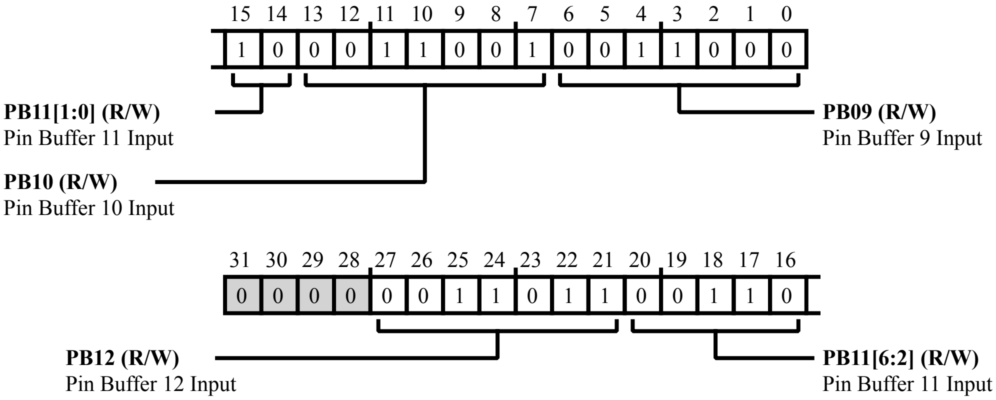

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0
PB11[1:0] (R/W)
Pin Buffer 11 Input
PB10 (R/W)
Pin Buffer 10 Input
PB09 (R/W)
Pin Buffer 9 Input
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0
PB12 (R/W)
Pin Buffer 12 Input
PB11[6:2] (R/W)
Pin Buffer 11 Input

Figure 33-49: DAI_PIN2 Register Diagram

Table 33-60: DAI_PIN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB12	Pin Buffer 12 Input. `DAI_PIN2.PB12` holds the Source signal assignment that will be routed to the `DAI_PIN2.PB12` Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB11	Pin Buffer 11 Input. `DAI_PIN2.PB11` holds the Source signal assignment that will be routed to the `DAI_PIN2.PB11` Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB10	Pin Buffer 10 Input. `DAI_PIN2.PB10` holds the Source signal assignment that will be routed to the `DAI_PIN2.PB10` Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB09	Pin Buffer 9 Input. `DAI_PIN2.PB09` holds the Source signal assignment that will be routed to the `DAI_PIN2.PB09` Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 3

The `DAI_PIN3` register routes physical pins that are connected to a bonded pad.

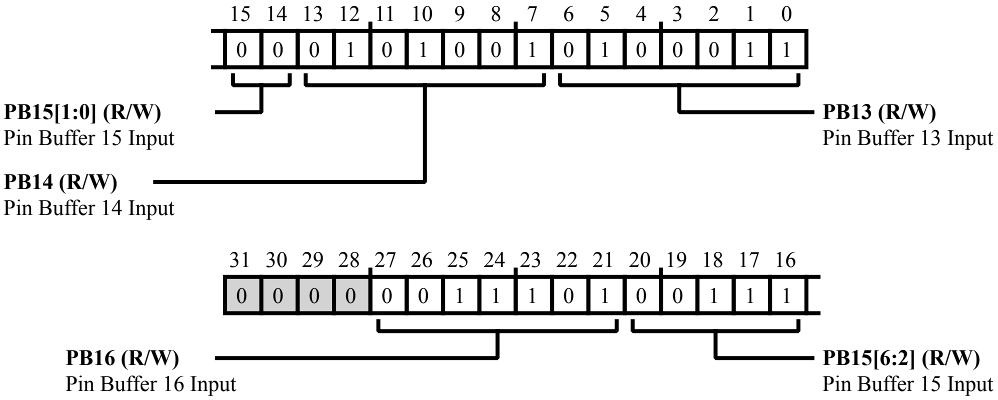

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1
PB15[1:0] (R/W)
Pin Buffer 15 Input
PB14 (R/W)
Pin Buffer 14 Input
PB13 (R/W)
Pin Buffer 13 Input
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 1 1 1 0 1 0 0 1 1 1
PB16 (R/W)
Pin Buffer 16 Input
PB15[6:2] (R/W)
Pin Buffer 15 Input

Figure 33-50: DAI_PIN3 Register Diagram

Table 33-61: DAI_PIN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:21 (R/W)	PB16	Pin Buffer 16 Input. The `DAI_PIN3.PB16` bit field is the pin buffer 16 input.
20:14 (R/W)	PB15	Pin Buffer 15 Input. `DAI_PIN3.PB15` holds the Source signal assignment that will be routed to the `DAI_PIN3.PB15` Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB14	Pin Buffer 14 Input. `DAI_PIN3.PB14` holds the Source signal assignment that will be routed to the `DAI_PIN3.PB14` Destination. Refer to the Group D Signals table for Source and Destination mappings.
6:0 (R/W)	PB13	Pin Buffer 13 Input. `DAI_PIN3.PB13` holds the Source signal assignment that will be routed to the `DAI_PIN3.PB13` Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Buffer Assignment Register 4

The DAI_PIN4 register routes physical pins that are connected to a bonded pad.

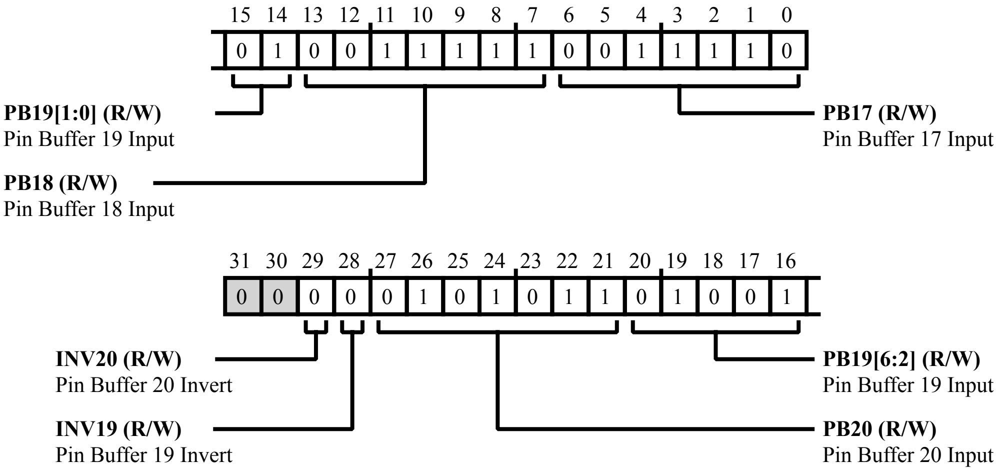

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0
PB19[1:0] (R/W)
Pin Buffer 19 Input
PB18 (R/W)
Pin Buffer 18 Input
PB17 (R/W)
Pin Buffer 17 Input
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1
INV20 (R/W)
Pin Buffer 20 Invert
INV19 (R/W)
Pin Buffer 19 Invert
PB19[6:2] (R/W)
Pin Buffer 19 Input
PB20 (R/W)
Pin Buffer 20 Input

Figure 33-51: DAI_PIN4 Register Diagram

Table 33-62: DAI_PIN4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	INV20	Pin Buffer 20 Invert. `DAI_PIN4.INV20` holds the Source signal assignment that will be routed to the `DAI_PIN4.INV20` Destination. Refer to the Group D Signals table for Source and Destination mappings.
28 (R/W)	INV19	Pin Buffer 19 Invert. `DAI_PIN4.INV19` holds the Source signal assignment that will be routed to the `DAI_PIN4.INV19` Destination. Refer to the Group D Signals table for Source and Destination mappings.
27:21 (R/W)	PB20	Pin Buffer 20 Input. `DAI_PIN4.PB20` holds the Source signal assignment that will be routed to the `DAI_PIN4.PB20` Destination. Refer to the Group D Signals table for Source and Destination mappings.
20:14 (R/W)	PB19	Pin Buffer 19 Input. `DAI_PIN4.PB19` holds the Source signal assignment that will be routed to the `DAI_PIN4.PB19` Destination. Refer to the Group D Signals table for Source and Destination mappings.
13:7 (R/W)	PB18	Pin Buffer 18 Input. `DAI_PIN4.PB18` holds the Source signal assignment that will be routed to the `DAI_PIN4.PB18` Destination. Refer to the Group D Signals table for Source and Destination mappings.

Table 33-62: DAI_PIN4 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	PB17	Pin Buffer 17 Input. `DAI_PIN4.PB17` holds the Source signal assignment that will be routed to the `DAI_PIN4.PB17` Destination. Refer to the Group D Signals table for Source and Destination mappings.

Pin Status Register

The DAI_PIN_STAT register bits indicate the status (signal high or low) for each pin. The individual bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

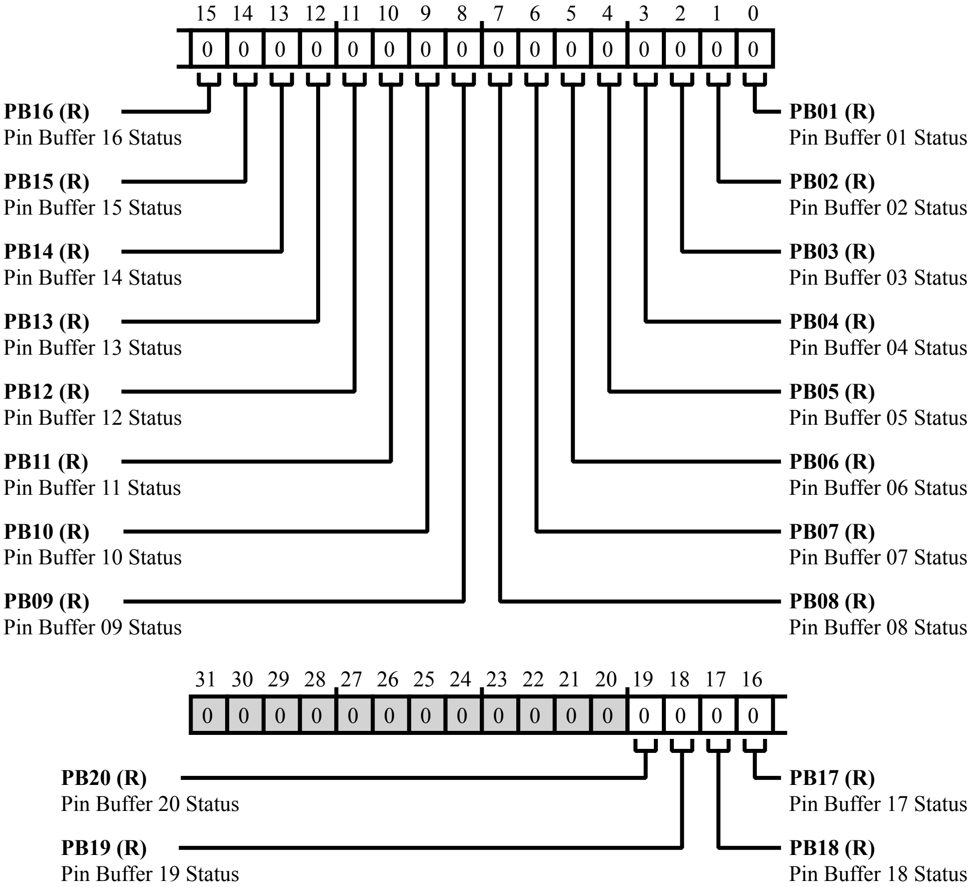

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PB16 (R) Pin Buffer 16 Status
PB15 (R) Pin Buffer 15 Status
PB14 (R) Pin Buffer 14 Status
PB13 (R) Pin Buffer 13 Status
PB12 (R) Pin Buffer 12 Status
PB11 (R) Pin Buffer 11 Status
PB10 (R) Pin Buffer 10 Status
PB09 (R) Pin Buffer 09 Status
PB01 (R) Pin Buffer 01 Status
PB02 (R) Pin Buffer 02 Status
PB03 (R) Pin Buffer 03 Status
PB04 (R) Pin Buffer 04 Status
PB05 (R) Pin Buffer 05 Status
PB06 (R) Pin Buffer 06 Status
PB07 (R) Pin Buffer 07 Status
PB08 (R) Pin Buffer 08 Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PB20 (R) Pin Buffer 20 Status
PB19 (R) Pin Buffer 19 Status
PB17 (R) Pin Buffer 17 Status
PB18 (R) Pin Buffer 18 Status

Figure 33-52: DAI_PIN_STAT Register Diagram

Table 33-63: DAI_PIN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R/NW)	PB20	Pin Buffer 20 Status. The `DAI_PIN_STAT.PB20` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
18 (R/NW)	PB19	Pin Buffer 19 Status. The `DAI_PIN_STAT.PB19` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

Table 33-63: DAI_PIN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/NW)	PB18	Pin Buffer 18 Status. The `DAI_PIN_STAT.PB18` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
16 (R/NW)	PB17	Pin Buffer 17 Status. The `DAI_PIN_STAT.PB17` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
15 (R/NW)	PB16	Pin Buffer 16 Status. The `DAI_PIN_STAT.PB16` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
14 (R/NW)	PB15	Pin Buffer 15 Status. The `DAI_PIN_STAT.PB15` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
13 (R/NW)	PB14	Pin Buffer 14 Status. The `DAI_PIN_STAT.PB14` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
12 (R/NW)	PB13	Pin Buffer 13 Status. The `DAI_PIN_STAT.PB13` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
11 (R/NW)	PB12	Pin Buffer 12 Status. The `DAI_PIN_STAT.PB12` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
10 (R/NW)	PB11	Pin Buffer 11 Status. The `DAI_PIN_STAT.PB11` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
9 (R/NW)	PB10	Pin Buffer 10 Status. The `DAI_PIN_STAT.PB10` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
8 (R/NW)	PB09	Pin Buffer 09 Status. The `DAI_PIN_STAT.PB09` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
7 (R/NW)	PB08	Pin Buffer 08 Status. The `DAI_PIN_STAT.PB08` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

Table 33-63: DAI_PIN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/NW)	PB07	Pin Buffer 07 Status. The `DAI_PIN_STAT.PB07` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
5 (R/NW)	PB06	Pin Buffer 06 Status. The `DAI_PIN_STAT.PB06` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
4 (R/NW)	PB05	Pin Buffer 05 Status. The `DAI_PIN_STAT.PB05` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
3 (R/NW)	PB04	Pin Buffer 04 Status. The `DAI_PIN_STAT.PB04` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
2 (R/NW)	PB03	Pin Buffer 03 Status. The `DAI_PIN_STAT.PB03` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
1 (R/NW)	PB02	Pin Buffer 02 Status. The `DAI_PIN_STAT.PB02` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.
0 (R/NW)	PB01	Pin Buffer 01 Status. The `DAI_PIN_STAT.PB01` bit reads 1 if the signal to this pin is high and reads 0 if signal to this pin is low.

34 Serial Port (SPORT)

The programmable serial ports (SPORTs) support various protocols for serial data communication and provide a glueless hardware interface to many industry-standard data converters and codecs. They have high data rates and dual half-duplex datapaths and are ideal for establishing a direct serial connection among two or more processors in a multiprocessor system, as many processors provide compatible serial interfaces.

The SPORT top module consists of two half SPORTs with identical functionality and programming requirements. Each half SPORT can be independently configured as either a transmitter or receiver and can be coupled with the other half SPORT within the same SPORT top module. Further, each half SPORT provides two synchronous half-duplex data lines to double the total supported throughput. As such, a single SPORT top module can be used to provide up to four unidirctional or up to two full-duplex data streams. Further channels are possible as well, but utilization of multiple SPORT top modules is required, thus requiring external connections to provide a common time base.

Features

An individual SPORT top module consists of two independently configurable SPORT halves with identical functionality. These SPORT halves offer the following features:

- Up to two bidirectional data lines - each half SPORT supports up to two transmit or receive channels, thus allowing two unidirectional streams into or out of each half SPORT and providing greater flexibility for serial communications. If full-duplex functionality is desired, two SPORT halves can be combined to enable dual-stream bidirectional communication.
- Six operating modes:
  1. Standard DSP serial mode
  2. I^{2}S mode
  3. Left-Justified mode
  4. Right-Justified mode
  5. Multichannel (TDM) mode
  6. Packed mode

- Supports internally or externally generated clock.
- Support for both even and odd SCLK0_0 to SPORT clock (SPORT_CLK) ratios. If both data lines of a half SPORT are active, the maximum throughput is 2 x SPORT_CLK bps.
- Configurable rising or falling edge of the SPORT_CLK for driving and sampling data and frame syncs.
- Gated clock mode support for internallly or externally generated clocks in DSP serial mode and stereo modes (left-justified and I^2S mode).
- Supports frameless operation.
- Supports internally or externally generated frame sync signals.
- Programmable frame sync polarity.
- Programmable frame sync timing (synchronous to data or 1 SPORT clock in advance of it).
- Detection of prematurely received external frame syncs (with optional interrupt request generation).
- Programmable level-/edge-sensitivity for external frame syncs.
- Programmable (4–32-bit) data length, either in most significant bit (MSB) first or in least significant bit (LSB) first format, with optional sign-extension on received data.
- Optional 16-bit to 32-bit word packing (as receiver) and 32-bit to 16-bit word unpacking (as transmitter).
- Support for A-law and μ-law compression/decompression hardware companding, according to the G.711 specification, on transmitted/received words in all operating modes.
- Transmit underrun and receive overflow detection (with optional interrupt request generation).
- TDM mode transfers data on 128 contiguous channels from a stream of up to 1024 total channels (useful for H.100/H.110 and other telephony interfaces as a network communication scheme for multiple processors).
- Performs interrupt-driven, single word core transfers to and from on-chip or off-chip memory.
- Dedicated DMA channel for each SPORT half supporting autobuffer (for a repeated, identical range of transfers) and numerous descriptor-based (individual or repeated ranges of transfers with differing DMA parameters) modes.
- Master and slave trigger functionality.
- Unique transfer finish interrupt (TFI) signaling when the last transmit word is fully out of the transmit shift register.
- Multiplexer to internally connect critical timing signals between SPORT halves.

Signal Descriptions

Each half SPORT module has five dedicated signals, as described in the ***SPORT Signal Descriptions*** table. The actual pin name varies with different SPORT halves. Individual SPORT halves do not share any of its signals across the pair that comprises the SPORT top module; however, it is possible to connect the clock and frame sync signals between the SPORT half pair, as explained in the Multiplexer Logic section.

All of the SPORT signals are multiplexed on the PORT pins, possibly sharing functionality with other peripherals on the device. By default, the PORT pins are in GPIO mode and must be reconfigured for SPORT functionality by setting the appropriate bits in the `PORT_FER` and `PORT_MUX` registers. Consult the processor datasheet for details regarding which ports the SPORT signals are available on, and be sure to configure the `PORT_MUX` register before the `PORT_FER` register.

Table 34-1: SPORT Signal Descriptions

Internal Node	Direction	Description
SPORTx_CLK	I/O	Transmit or receive serial clock. Data and frame syncs are driven or sampled on this clock's edges. This signal can be either internally or externally generated.
SPORTx_FS	I/O	Transmit or receive frame sync. The frame sync pulse initiates shifting of serial data. This signal is either internally or externally generated.
SPORTx_D0	I/O	Primary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORTx_D1	I/O	Secondary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORTx_TDV	O	Multichannel transmit data valid. This signal is only active in multichannel transmit mode and is asserted during enabled slots, as defined by the channel selection registers (`SPORT_CS0_A` through `SPORT_CS3_B`).

The data channel signals are transmit signals when the serial port is configured in transmit mode (`SPORT_CTL_A.SPTRAN` = 1). They are receive signals when the serial port is configured in receive mode (`SPORT_CTL_A.SPTRAN` = 0). The following sections further describe the SPORT signals.

NOTE: These sections refer explicitly to registers associated with half SPORT A, but the same concepts also apply to half SPORT B.

Serial Clock

The serial port clock (`SPORT_ACLK`) is either a receive serial clock or a transmit serial clock, depending on the transfer direction (`SPORT_CTL_A.SPTRAN`), governing when the data bits are serially shifted into or out of the SPORT and when the frame sync signal is driven (in internal frame sync mode) or sampled (in external frame sync mode). It can be internally generated from the processor's system clock (SCLK0_0) or externally provided. If the half SPORT is configured in internal clock mode (`SPORT_CTL_A.ICLK`= 1), then the `SPORT_DIV_A.CLKDIV` field specifies the divisor applied to SCLK0_0 to generate the SPORT clock. As it is a 16-bit divisor, a wide range of serial clock rates is possible. Use the following equation to calculate the serial clock frequency:

`SPORT_ACLK` = [SCLK0_0 ÷ (`SPORT_DIV_A.CLKDIV` + 1)]

From this, the following equation can be used to determine the value of `SPORT_DIV_A.CLKDIV`, given the SCLK0_0 frequency and the desired frequency of the SPORT clock:

`SPORT_DIV_A.CLKDIV`= [(SCLK0_0 ÷ `SPORT_ACLK`) - 1]

The half SPORT also supports a 1:1 `SPORT_ACLK` to SCLK0_0 ratio (per the equations above, program the clock divisor field to zero). In this case, the resulting SPORT clock frequency is equal to SCLK0_0.

NOTE: Be careful not to exceed the maximum `SPORT_ACLK` frequency specified in the processor datasheet.

In certain operating modes, the SPORT can be configured to generate a gated clock, which is active only during valid data. In some applications, a SPORT uses it to generate a general-purpose clock in the system. In this case, enable the SPORT with the appropriate `SPORT_DIV_A.CLKDIV` divisor field in internal clock mode.

If a SPORT is configured in external clock mode (`SPORT_CTL_A.ICLK`= 0), then the serial clock is an input signal, thus making the SPORT operate in slave mode. In this mode, the `SPORT_DIV_A.CLKDIV` is irrelevant and is ignored. The optional loopback capability provided by the internal SPORT multiplexer (SPMUX) block allows the slave SPORT to use the serial clock from the neighboring SPORT half in the same SPORT top module.

An externally-supplied serial clock does not need to be synchronous with the processor clocks. Further, the external clock can be a gated clock, but it must comply with the requirements described in the Gated Clock Mode section.

Refer to the product datasheet for exact AC timing specifications.

Frame Sync

The SPORT frame sync (`SPORT_AFS`) signal is either a receive frame sync or a transmit frame sync, depending on the transfer direction (`SPORT_CTL_A.SPTRAN`), which is used to determine the start of a new word or frame. When this signal goes active, the serial port starts serially shifting data into or out of the SPORT. The frame sync signal can be internally generated based on its serial clock (`SPORT_ACLK`) or externally provided, as configured by the `SPORT_CTL_A.IFS` bit.

If the half SPORT is configured to generate frame syncs (`SPORT_CTL_A.IFS`= 1), then the `SPORT_DIV_A.FSDIV` field specifies the divisor used to generate the periodic `SPORT_AFS` signal from the SPORT clock. As this is a 16-bit divisor, a wide range of frame sync rates to initiate periodic transfers is possible. Whether the serial clock is internally or externally generated, this divisor is a count of SPORT clock cycles between frame sync pulses, the formula for which is:

Number of serial clocks between frame syncs = (`SPORT_DIV_A.FSDIV` + 1)

From this, the following equation can be used to determine the value of `SPORT_DIV_A.FSDIV`, given the serial clock frequency and the desired frame sync frequency:

`SPORT_DIV_A.FSDIV` = [(`SPORT_ACLK` ÷ `SPORT_AFS`) – 1]

The frame sync is continuously active when `SPORT_DIV_A.FSDIV`= 0. The value of `SPORT_DIV_A.FSDIV` cannot be less than the serial word length minus one (the value of the `SPORT_CTL_A.SLEN` bit field). Failure to adhere to this guideline can cause an external device to abort the current operation or cause other unpredictable results.

NOTE: After enabling the SPORT, the first internal frame sync appears after a delay of `SPORT_DIV_A.FSDIV` + 3 SPORT clock cycles.

If a SPORT is configured for external frame syncs (`SPORT_CTL_A.IFS` = 0), then `SPORT_AFS` is an input signal and the `SPORT_DIV_A.FSDIV` field of the `SPORT_DIV_A` register is irrelevant and ignored. By default, this external signal is level-sensitive, but it can be configured as an edge-sensitive signal by setting the `SPORT_CTL_A.FSED` bit. The frame sync is expected to be synchronous with the serial clock. If not, it must meet the timing requirements that appear in the datasheet.

The SPORT can be used as a counter for dividing an external clock to generate periodic pulses or periodic interrupts. To do so, enable the SPORT with the appropriate `SPORT_DIV_A.FSDIV` divisor field with the SPORT configured for an external clock and internal data-independent frame syncs.

In some of the operating modes, the SPORT can be programmed to treat the frame sync signal as an optional signal by clearing the `SPORT_CTL_A.FSR` bit. Even with this bit cleared, the SPORT requires a single frame sync assertion to start the continuous transfers, after which it is ignored (for externally supplied frame syncs) or not generated (for internally-generated frame syncs). Characteristics of the frame sync depend on the settings in the SPORT control registers and the operating mode of the SPORT. For more information, refer to the `SPORT_CTL_A` register.

Data Signals

Each half SPORT has two bidirectional data lines known as the primary (`SPORT_AD0`) and secondary (`SPORT_AD1`) data channels. Both of the data lines can be configured as either transmitters or receivers using the `SPORT_CTL_A.SPTRAN` bit, thus permitting dual unidirectional data streams to increase the data throughput of the SPORT.

NOTE: Configuring one transmit data channel and one receive data channel on a single half SPORT is not supported.

The primary and secondary data lines can be individually enabled or disabled using the `SPORT_CTL_A.SPENPRI` and the `SPORT_CTL_A.SPENSEC` bits, respectively. However, if using both, enable or disable them concurrently. These data lines operate in a synchronous manner (sharing a clock and frame sync) but have separate datapaths with unique data buffers, shift registers and optional companding logic. All of the SPORT control settings are common for both channels, but the single DMA channel per half SPORT serves both the primary and secondary data channels.

When a SPORT is configured in multichannel transmit mode, the data pins three-state during inactive channel slots, thus allowing multiple transmitters to operate on the same bus with different active channels.

See the Architectural Concepts section for more details about data transfer operation.

Transmit Data Valid Signal

The transmit data valid (`SPORT_ATDV`) signal is available only in transmit multichannel modes (including packed mode). It is driven active during enabled multichannel slots, and it is driven inactive during the disabled channels. In other words, the `SPORT_ATDV` signal is active when data is being driven to the data pins and inactive when the data pins are being three-stated. As such, the `SPORT_ATDV` signal can serve as an output-enable signal for the data transmit pin.

SRU Programming

The SPORT uses the SRU (signal routing unit) to connect the SPORT data, serial clock, frame sync, and external sync (if external synchronization is required). Inputs also must be routed through the SRU. Program the corresponding SRU registers to connect the outputs to the required destinations. For details of the routing, see DAI Routing Capabilities in the *Digital Audio Interface (DAI)* chapter.

SRU SPORT Receive Master

If the SPORT is operating as receive master, it must feed its master output clock back to its input clock. This is required to trigger the SPORT's state machine. Using SPORT 0A as an example receive master, programs should route `SPT0_ACLK_O` to `SPT0_ACLK_I`. This is not required if the SPORT is operating as a transmitter in master mode.

Functional Description

The following sections provide general information about the functionality of the processor's serial ports:

- Architectural Concepts
- Data Types and Companding
- Transmit Path
- Receive Path

ADSP-SC58x SPORT Register List

The Serial Port (SPORT) controller, with its range of clock and frame synchronization options, supports a variety of serial communication protocols and provides a glueless hardware interface to many industry-standard data converters and CODECs. Each SPORT has two independent halves (A and B), and each half contains two channels (primary and secondary). A set of registers governs SPORT operations. For more information on SPORT functionality, see the SPORT register descriptions.

Table 34-2: ADSP-SC58x SPORT Register List

Name	Description
`SPORT_CS0_A`	Half SPORT 'A' Multichannel 0-31 Select Register
`SPORT_CS0_B`	Half SPORT 'B' Multichannel 0-31 Select Register
`SPORT_CS1_A`	Half SPORT 'A' Multichannel 32-63 Select Register
`SPORT_CS1_B`	Half SPORT 'B' Multichannel 32-63 Select Register
`SPORT_CS2_A`	Half SPORT 'A' Multichannel 64-95 Select Register

Table 34-2: ADSP-SC58x SPORT Register List (Continued)

Name	Description
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multichannel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multichannel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

ADSP-SC58x SPORT Interrupt List

Table 34-3: ADSP-SC58x SPORT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
52	SPORT0_A_DMA	SPORT0 Channel A DMA	Level	0
53	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
54	SPORT0_B_DMA	SPORT0 Channel B DMA	Level	1

Table 34-3: ADSP-SC58x SPORT Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
55	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
56	SPORT1_A_DMA	SPORT1 Channel A DMA	Level	2
57	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
58	SPORT1_B_DMA	SPORT1 Channel B DMA	Level	3
59	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
60	SPORT4_A_DMA	SPORT4 Channel A DMA	Level	10
61	SPORT4_A_STAT	SPORT4 Channel A Status	Level	
62	SPORT4_B_DMA	SPORT4 Channel B DMA	Level	11
63	SPORT4_B_STAT	SPORT4 Channel B Status	Level	
64	SPORT5_A_DMA	SPORT5 Channel A DMA	Level	12
65	SPORT5_A_STAT	SPORT5 Channel A Status	Level	
66	SPORT5_B_DMA	SPORT5 Channel B DMA	Level	13
67	SPORT5_B_STAT	SPORT5 Channel B Status	Level	
90	SPORT2_A_DMA	SPORT2 Channel A DMA	Level	4
91	SPORT2_A_STAT	SPORT2 Channel A Status	Level	
92	SPORT2_B_DMA	SPORT2 Channel B DMA	Level	5
93	SPORT2_B_STAT	SPORT2 Channel B Status	Level	
94	SPORT3_A_DMA	SPORT3 Channel A DMA	Level	6
95	SPORT3_A_STAT	SPORT3 Channel A Status	Level	
96	SPORT3_B_DMA	SPORT3 Channel B DMA	Level	7
97	SPORT3_B_STAT	SPORT3 Channel B Status	Level	
98	SPORT6_A_DMA	SPORT6 Channel A DMA	Level	14
99	SPORT6_A_STAT	SPORT6 Channel A Status	Level	
100	SPORT6_B_DMA	SPORT6 Channel B DMA	Level	15
101	SPORT6_B_STAT	SPORT6 Channel B Status	Level	
102	SPORT7_A_DMA	SPORT7 Channel A DMA	Level	16
103	SPORT7_A_STAT	SPORT7 Channel A Status	Level	
104	SPORT7_B_DMA	SPORT7 Channel B DMA	Level	17
105	SPORT7_B_STAT	SPORT7 Channel B Status	Level	
180	SPORT0_A_DMA_ERR	SPORT0 Channel A DMA Error	Level	
181	SPORT0_B_DMA_ERR	SPORT0 Channel B DMA Error	Level	

Table 34-3: ADSP-SC58x SPORT Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
182	SPORT1_A_DMA_ERR	SPORT1 Channel A DMA Error	Level	
183	SPORT1_B_DMA_ERR	SPORT1 Channel B DMA Error	Level	
184	SPORT4_A_DMA_ERR	SPORT4 Channel A DMA Error	Level	
185	SPORT4_B_DMA_ERR	SPORT4 Channel B DMA Error	Level	
186	SPORT5_A_DMA_ERR	SPORT5 Channel A DMA Error	Level	
187	SPORT5_B_DMA_ERR	SPORT5 Channel B DMA Error	Level	
190	SPORT2_A_DMA_ERR	SPORT2 Channel A DMA Error	Level	
191	SPORT2_B_DMA_ERR	SPORT2 Channel B DMA Error	Level	
192	SPORT3_A_DMA_ERR	SPORT3 Channel A DMA Error	Level	
193	SPORT3_B_DMA_ERR	SPORT3 Channel B DMA Error	Level	
194	SPORT6_A_DMA_ERR	SPORT6 Channel A DMA Error	Level	
195	SPORT6_B_DMA_ERR	SPORT6 Channel B DMA Error	Level	
196	SPORT7_A_DMA_ERR	SPORT7 Channel A DMA Error	Level	
197	SPORT7_B_DMA_ERR	SPORT7 Channel B DMA Error	Level	

ADSP-SC58x SPORT Trigger List

Table 34-4: ADSP-SC58x SPORT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
24	`SPORT0_A_DMA`	SPORT0 Channel A DMA	Edge
25	`SPORT0_B_DMA`	SPORT0 Channel B DMA	Edge
26	`SPORT1_A_DMA`	SPORT1 Channel A DMA	Edge
27	`SPORT1_B_DMA`	SPORT1 Channel B DMA	Edge
28	`SPORT2_A_DMA`	SPORT2 Channel A DMA	Edge
29	`SPORT2_B_DMA`	SPORT2 Channel B DMA	Edge
30	`SPORT3_A_DMA`	SPORT3 Channel A DMA	Edge
31	`SPORT3_B_DMA`	SPORT3 Channel B DMA	Edge
32	`SPORT4_A_DMA`	SPORT4 Channel A DMA	Edge
33	`SPORT4_B_DMA`	SPORT4 Channel B DMA	Edge
34	`SPORT5_A_DMA`	SPORT5 Channel A DMA	Edge
35	`SPORT5_B_DMA`	SPORT5 Channel B DMA	Edge

Table 34-4: ADSP-SC58x SPORT Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
36	`SPORT6_A_DMA`	SPORT6 Channel A DMA	Edge
37	`SPORT6_B_DMA`	SPORT6 Channel B DMA	Edge
38	`SPORT7_A_DMA`	SPORT7 Channel A DMA	Edge
39	`SPORT7_B_DMA`	SPORT7 Channel B DMA	Edge

Table 34-5: ADSP-SC58x SPORT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
17	`SPORT0_A_DMA`	SPORT0 Channel A DMA	Pulse
18	`SPORT0_B_DMA`	SPORT0 Channel B DMA	Pulse
19	`SPORT1_A_DMA`	SPORT1 Channel A DMA	Pulse
20	`SPORT1_B_DMA`	SPORT1 Channel B DMA	Pulse
21	`SPORT2_A_DMA`	SPORT2 Channel A DMA	Pulse
22	`SPORT2_B_DMA`	SPORT2 Channel B DMA	Pulse
23	`SPORT3_A_DMA`	SPORT3 Channel A DMA	Pulse
24	`SPORT3_B_DMA`	SPORT3 Channel B DMA	Pulse
25	`SPORT4_A_DMA`	SPORT4 Channel A DMA	Pulse
26	`SPORT4_B_DMA`	SPORT4 Channel B DMA	Pulse
27	`SPORT5_A_DMA`	SPORT5 Channel A DMA	Pulse
28	`SPORT5_B_DMA`	SPORT5 Channel B DMA	Pulse
29	`SPORT6_A_DMA`	SPORT6 Channel A DMA	Pulse
30	`SPORT6_B_DMA`	SPORT6 Channel B DMA	Pulse
31	`SPORT7_A_DMA`	SPORT7 Channel A DMA	Pulse
32	`SPORT7_B_DMA`	SPORT7 Channel B DMA	Pulse

ADSP-SC58x SPORT DMA Channel List

Table 34-6: ADSP-SC58x SPORT DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA1	SPORT0_B_DMA	SPORT0 Channel B DMA
DMA2	SPORT1_A_DMA	SPORT1 Channel A DMA
DMA3	SPORT1_B_DMA	SPORT1 Channel B DMA

Table 34-6: ADSP-SC58x SPORT DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA4	SPORT2_A_DMA	SPORT2 Channel A DMA
DMA5	SPORT2_B_DMA	SPORT2 Channel B DMA
DMA6	SPORT3_A_DMA	SPORT3 Channel A DMA
DMA7	SPORT3_B_DMA	SPORT3 Channel B DMA
DMA10	SPORT4_A_DMA	SPORT4 Channel A DMA
DMA11	SPORT4_B_DMA	SPORT4 Channel B DMA
DMA12	SPORT5_A_DMA	SPORT5 Channel A DMA
DMA13	SPORT5_B_DMA	SPORT5 Channel B DMA
DMA14	SPORT6_A_DMA	SPORT6 Channel A DMA
DMA15	SPORT6_B_DMA	SPORT6 Channel B DMA
DMA16	SPORT7_A_DMA	SPORT7 Channel A DMA
DMA17	SPORT7_B_DMA	SPORT7 Channel B DMA

Block Diagram

Each SPORT top module consists of two separate blocks, known as half SPORTs (HSPORT) A and B, each with identical functionality and programming models. The ***Half Serial Port Block Diagram*** shows a detailed block diagram of a half SPORT.

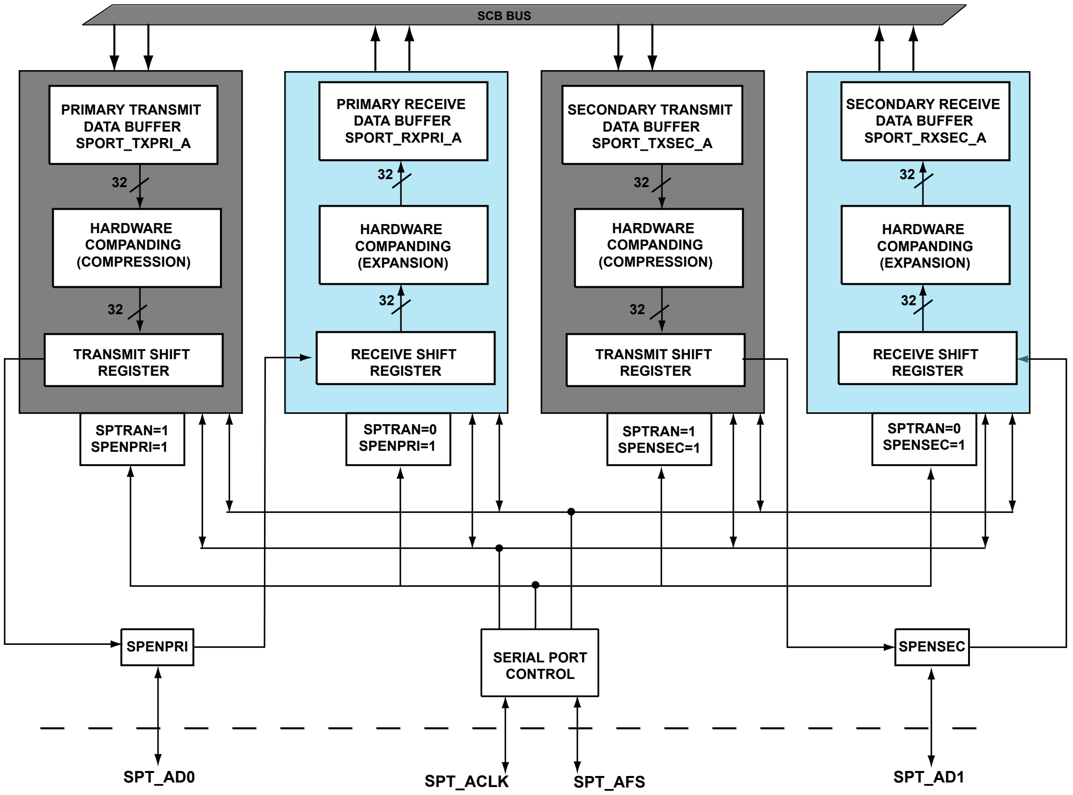

SCB BUS
PRIMARY TRANSMIT DATA BUFFER SPORT_TXPRI_A
32
HARDWARE COMPANDING (COMPRESSION)
32
TRANSMIT SHIFT REGISTER
SPTRAN=1 SPENPRI=1
PRIMARY RECEIVE DATA BUFFER SPORT_RXPRI_A
32
HARDWARE COMPANDING (EXPANSION)
32
RECEIVE SHIFT REGISTER
SPTRAN=0 SPENPRI=1
SECONDARY TRANSMIT DATA BUFFER SPORT_TXSEC_A
32
HARDWARE COMPANDING (COMPRESSION)
32
TRANSMIT SHIFT REGISTER
SPTRAN=1 SPENSEC=1
SECONDARY RECEIVE DATA BUFFER SPORT_RXSEC_A
32
HARDWARE COMPANDING (EXPANSION)
32
RECEIVE SHIFT REGISTER
SPTRAN=0 SPENSEC=1
SPENPRI
SERIAL PORT CONTROL
SPENSEC
SPT_AD0
SPT_ACLK
SPT_AFS
SPT_AD1

Figure 34-1: Half Serial Port Block Diagram

Architectural Concepts

Each half SPORT (HSPORT) block has its own set of control registers and data buffers, grouped per SPORT module. The HSPORT A and B blocks can be independently configured as either a transmitter or a receiver, with the option to be coupled together internally within the single SPORT top module. Each HSPORT also supports two synchronous bidirectional datapaths, referred to as the primary (`D0`) and secondary (`D1`) data lines, as shown in the ***Top-Level SPORT Diagram*** figure.

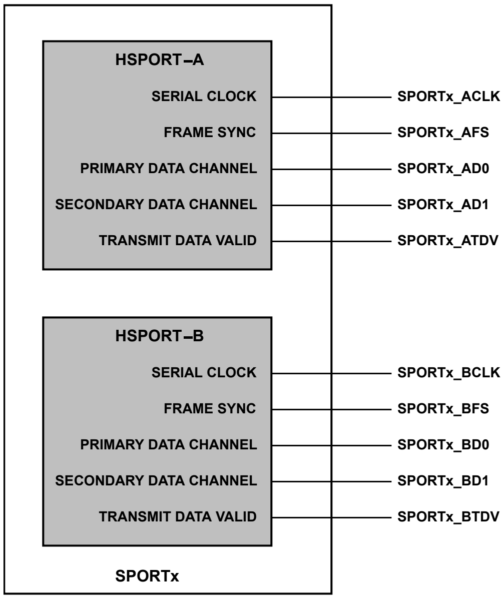

HSPORT–A
SERIAL CLOCK
FRAME SYNC
PRIMARY DATA CHANNEL
SECONDARY DATA CHANNEL
TRANSMIT DATA VALID
SPORTx_ACLK
SPORTx_AFS
SPORTx_AD0
SPORTx_AD1
SPORTx_ATDV
HSPORT–B
SERIAL CLOCK
FRAME SYNC
PRIMARY DATA CHANNEL
SECONDARY DATA CHANNEL
TRANSMIT DATA VALID
SPORTx_BCLK
SPORTx_BFS
SPORTx_BD0
SPORTx_BD1
SPORTx_BTDV
SPORTx

Figure 34-2: Top-Level SPORT Diagram

The `SPORT_CTL_A.SPTRAN` bit controls the direction for both datapaths of the HSPORT. Depending on whether the HSPORT is a transmitter or a receiver, the pair of data signals respectfully transmit or receive data bits synchronously. The dual data signals of each HSPORT cannot transmit and receive the data simultaneously in support of full-duplex operation, however, two HSPORTs can be combined to achieve this.

Serial communications are synchronized to the serial clock signal, where a valid clock pulse must accompany each data bit. Each HSPORT can take its clock from an external source or internally generate it from the processor's system clock using the `SPORT_DIV_A.CLKDIV` clock divisor bit field. Both primary and secondary data channels shift data based on the `SPORT_CLK` rate and the clock polarity defined by the `SPORT_CTL_A.CKRE` bit.

In addition to the serial clock signal, a frame synchronization signal is used to signify the beginning of an individual data word or a multichannel data stream (block of words). Each SPORT can take the frame sync signal from an external source or generate it (`SPORT_FS`), depending on the `SPORT_CTL_A.IFS` bit. An internally generated frame sync is derived from the SPORT clock using the `SPORT_DIV_A.FSDIV` divisor field. Both primary and secondary datapaths start shifting data either synchronous to or one serial clock in advance of detecting/generating a valid frame sync signal, as determined by the `SPORT_CTL_A.LAFS` bit. Various communication protocols for serial data can be emulated according to the frame sync format, and all frame sync options are available whether the signal is generated internally or externally.

NOTE: These SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol.

Multiplexer Logic

The SPORT multiplexing block (SPMUX) is situated between the SPORT hardware block and the processor's pin multiplexing logic. It allows the flexibility to route and share the clock and frame sync signals between the HSPORT A and B halves within each SPORT top module, which can double the data throughput (if both SPORT halves are transmitters or both are receivers) or provide full-duplex capabilities (if one HSPORT is a receiver and the other a transmitter) without the need to allocate pins for the peripheral or make physical connections outside the processor. The `SPORT_CTL2_A` register is used to configure this loopback feature.

NOTE: Throughout this section, HSPORT A is used as a reference, but all concepts also apply to HSPORT B.

The multiplexing depends on the configuration of the `SPORT_CTL_A.IFS` and `SPORT_CTL_A.ICLK` bits, and the `SPORT_CTL2_A.CKMUXSEL` and `SPORT_CTL2_A.FSMUXSEL` bit settings control the multiplexing. The ***Frame Sync Combinations*** and ***Clock Combinations*** tables show the valid combinations for the bit settings.

NOTE: All other settings are illegal. However, hardware does not check or prevent the illegal settings. Ensure that programs use only legal combinations.

The column headers in the ***Frame Sync Combinations*** table are defined as follows:

- FS Combination = Frame sync combination, referenced in the notes that follow the ***Clock Combinations*** table.
- HSA_IFS = the setting of the `SPORT_CTL_A.IFS` configuration bit.
- HSB_IFS = the setting of the `SPORT_CTL_B.IFS` configuration bit.
- FSAMUX = the setting of the `SPORT_CTL2_A.FSMUXSEL` configuration bit.
- FSBMUX = the setting of the `SPORT_CTL2_B.FSMUXSEL` configuration bit.

The Routing column in the ***Frame Sync Combinations*** table defines how the signals are used between the SPORT halves and which pin is used for the frame sync (whether it is an input or an output). Within the column, the inequality characters (≤ and ≥) are used to show the direction of the signal, and the following abbreviations are used (where x = A or B):

- HSx_FI = Frame sync input signal, provided by an external device or the complementing HSPORT.
- HSx_FO = Frame sync output signal.
- SPx_FS = HSPORT's frame sync pin, where the signal is either:
  - provided by an external source and distributed to both HSPORT frame sync signals, or
  - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT frame sync signal.

Table 34-7: Frame Sync Combinations

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
1	0	0	0	0	Native FS Operation
2	0	1	0	0	Native FS Operation
3	1	0	0	0	Native FS Operation
4	1	1	0	0	Native FS Operation

Table 34-7: Frame Sync Combinations (Continued)

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
5	0	0	1	0	HSA_FI ≤ SPB_FS; HSB_FI ≤ SPB_FS
6	0	1	1	0	HSA_FI ≤ HSB_FO ≥ SPB_FS
7	0	0	0	1	HSB_FI ≤ SPA_FS; HSA_FI ≤ SPA_FS
8	1	0	0	1	HSB_FI ≤ HSA_FO ≥ SPA_FS

The column headers in the ***Clock Combinations*** table are defined as follows:

- CLK Combination = Clock combination, referenced in the notes that follow the table.
- HSA_ICLK = the setting of the `SPORT_CTL_A.ICLK` configuration bit.
- HSB_ICLK = the setting of the `SPORT_CTL_B.ICLK` configuration bit.
- CKAMUX = the setting of the `SPORT_CTL2_A.CKMUXSEL` configuration bit.
- CKBMUX = the setting of the `SPORT_CTL2_B.CKMUXSEL` configuration bit.

The Routing column in the ***Clock Combinations*** table defines how the signals are used between the SPORT halves and which pin is used for the serial clock (whether it is an input or an output). Within the column, the inequality characters (≤ and ≥) are used to show the direction of the signal, and the following abbreviations are used (x = A or B):

- HSx_CI = Serial clock input signal, provided by an external device or the complementing HSPORT.
- HSx_CO = Serial clock output signal.
- SPx_CLK = HSPORT's serial clock pin, where the signal is either:
  - provided by an external source and distributed to both HSPORT serial clock signals, or
  - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT serial clock signal.

Table 34-8: Clock Combinations

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
9	0	0	0	0	Native CLK Operation
10	0	1	0	0	Native CLK Operation
11	1	0	0	0	Native CLK Operation
12	1	1	0	0	Native CLK Operation

Table 34-8: Clock Combinations (Continued)

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
13	0	0	1	0	HSA_CI ≤ SPB_CLK; HSB_CI ≤ SPB_CLK
14	0	1	1	0	HSA_CI ≤ HSB_CO ≥ SPB_CLK
15	0	0	0	1	HSB_CI ≤ SPA_CLK; HSA_CI ≤ SPA_CLK
16	1	0	0	1	HSB_CI ≤ HSA_CO ≥ SPA_CLK

The following is a comprehensive list of the legal combinations for the above described frame sync and clock multiplexing configurations:

- FS Combinations 1–4 are compatible with all CLK Combinations (9–16)
- CLK Combinations 9–12 are compatible with all FS Combinations (1–8)
- FS Combination 5 is only compatible with CLK Combination 13 (and vice versa)
- FS Combination 6 is only compatible with CLK Combination 14 (and vice versa)
- FS Combination 7 is only compatible with CLK Combination 15 (and vice versa)
- FS Combination 8 is only compatible with CLK Combination 16 (and vice versa)

NOTE: Program only the `SPORT_CTL2` register of the HSPORT that is accepting the signal from the other HSPORT. However, be sure to set the `SPORT_CTL_A.CKRE` and `SPORT_CTL_A.LFS` polarity bits to have identical settings between the HSPORTs when making internal connections via the SPMUX block.

Data Types and Companding

The SPORT uses the data type select field `SPORT_CTL_A.DTYPE` bit to specify one of the four data formats supported by serial ports. These formats apply to any of the operating modes of serial port.

Table 34-9: Data Type Bit Field Settings

DTYPE field	SPORT Receiver	SPORT Transmitter
00	Right-justify, zero-fill unused most significant bits	Normal operation
01	Right-justify, sign-extend unused most significant bits	Reserved
10	Expand using μ-law	Compress using μ-law
11	Expand using A-law	Compress using A-law

These formats apply to data words loaded into the SPORT transmit or receive data buffers. The first two data formats (`00` and `01` values of `SPORT_CTL_A.DTYPE`) are applicable only when SPORT is configured as receiver.

When SPORT is configured as transmitter, only the significant bits are transmitted (per the field defined in control register). Therefore, the transmit data buffers are not actually zero-filled or sign-extended.

The other two data formats enable the companding logic on the transmit or receive path. Companding (compressing or expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits sent. The SPORTs of the processor support the two most widely used companding algorithms, A-law, and μ-law. The algorithms are performed according to the CCITT G.711 specification.

If selected, companding applies to both the enabled data channels. When enabled as SPORT transmitter, writes to transmit buffer make the content compressed to 8 bits according to algorithm selected. (The content is zero filled to the width of the transmit word.) Similarly, if configured in receive mode, the 8 bits in the receive data buffers expand in right-justified, zero fill format per the algorithm selected. If companding is enabled in multichannel mode, it applies to all the active channels.

The compression for transmit data requires a minimum word length of 8 for proper function. If `SPORT_CTL_A.SLEN` is less than 7, then expansion does not work correctly. Also, if the data value is greater than 13-bit A-law or 14-bit μ-law maximum, it automatically compresses to the maximum value.

NOTE: The processor companding logic supports in-place companding feature. So, companding can be used for debug without enabling SPORT.

Companding as a Function

Since the values in the transmit and receive buffers are companded in place, the SPORT can use the companding hardware without transmitting or receiving data, which can be useful during testing or debugging. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting, use the following procedure:

1. Set the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN` = 1) with both primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI` = `SPORT_CTL_A.SPENSEC`= 0).
2. Enable companding in the `SPORT_CTL_A.DTYPE` field.
3. Write a 32-bit data word to the transmit buffer.
4. Wait two system clock cycles to allow the SPORT companding hardware to reload the transmit buffer with the companded value. Any instructions that do not access the transmit buffer can be used to cause this delay.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer.

Transmit Path

When the `SPORT_CTL_A.SPTRAN` control bit is set, the HSPORT is in transmit mode. Primary and secondary transmit data paths are then enabled using the `SPORT_CTL_A.SPENPRI` and `SPORT_CTL_A.SPENSEC` bits, respectively. The primary and secondary datapaths are unique and identical, each including its own transmit data buffer, optional companding logic, and transmit shift register.

The data buffer on the primary transmit data path is `SPORT_TXPRI_A`, and the data buffer on the secondary transmit data path is `SPORT_TXSEC_A`. The transmit data buffer and output shift register form a FIFO type of structure. When packing is disabled (`SPORT_CTL_A.PACK` = 0), the SPORT can hold as many as three data words. If packing is enabled (`SPORT_CTL_A.PACK` = 1), the serial port can hold two packed data words at any time.

The transmit data for primary and secondary channels is written to the `SPORT_TXPRI_A` and `SPORT_TXSEC_A` transmit data buffers, respectively. The transmit data buffers can be accessed in core mode through the peripheral bus or in DMA mode through the DMA bus. When a SPORT is configured in transmit mode, the receive paths are deactivated and do not respond to serial clock or frame sync signals. Because the receive data buffers and receive shift registers are also deactivated, reading from an empty and inactive receive data buffer can cause the core to hang indefinitely.

> **NOTE:** Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

This data can optionally be compressed in hardware according to the selected algorithm (μ-law or A-law) and then automatically transferred to the transmit shift register. The shift register, clocked by the `SPORT_ACLK` signal, then serially outputs this data on the `SPORT_AD0` and/or `SPORT_AD1` pins (if both are enabled, these output data bits are transmitted synchronously). If the SPORT uses a framing signal, the `SPORT_AFS` signal indicates the start of the serial word transmission.

When using DMA mode, a single DMA feeds the data buffers of the enabled channels (primary and/or secondary). When using both channels, interleave the data of these channels starting with the primary channel in the transmit buffer.

When the SPORT is configured in non-multichannel mode as a transmitter, the enabled SPORT data pins (`SPORT_AD0` and/or `SPORT_AD1`) are always driven. When a SPORT channel is enabled, data from the transmit data buffer is loaded into the transmit shift register. The shift register then immediately latches the first bit of data (either the LSB or MSB, depending on the `SPORT_CTL_A.LSBF` configuration bit) and drives it to the respective data pin such that it is ready when the frame sync signal asserts. Similarly, if the frame sync period exceeds the serial word length, then the data pins are driven with the first bit of the next word for transmission immediately after the active word completes, and the outputs are held during the inactive serial clock cycles (clock cycles between frame sync pulses).

When the SPORT is configured in multichannel mode, the data pins are driven only during active transmit channels and are always three-stated during inactive channel slots.

The SPORT provides status of transmit data buffers and also error detection logic for transmit errors such as an underrun condition. See the Error Detection (Status) Interrupt section for more details.

Receive Path

When the `SPORT_CTL_A.SPTRAN` bit is cleared, the SPORT is in receive mode. Primary and/or secondary receive data paths can be enabled by setting the `SPORT_CTL_A.SPENPRI` and `SPORT_CTL_A.SPENSEC` configuration bits, respectively. These data paths are unique but identical, each with a receive shift register, optional companding logic, and a receive data buffer.

The data buffer on the primary receive path is `SPORT_RXPRI_A`, and the data buffer on the secondary receive path is `SPORT_RXSEC_A`. The receive data paths act like a three-deep (32-bit words) FIFO because they have two data registers plus an input shift register.

Upon enabling the SPORT data channels, the input shift register shifts in data bits on the `SPORT_AD0` and/or `SPORT_AD1` pins, synchronous to the SPORT clock signal. If the SPORT uses a framing signal, the `SPORT_AFS` signal indicates the beginning of the serial word (or frame) to be received. When an entire word is shifted into the primary and secondary channels, the data can be optionally expanded in hardware according to a selected algorithm (μ-law or A-law) and then automatically transferred to the `SPORT_RXPRI_A` and/or `SPORT_RXSEC_A` data buffers.

The receive data buffers can be read in core mode through the peripheral bus or in DMA mode through the DMA bus. When the SPORT uses DMA mode, a single DMA reads the data buffers of the enabled channels (primary and/or secondary) and interleaves them in memory beginning with the primary channel when both channels are enabled. When using both channels, software must de-interleave the data of these channels.

The SPORT provides the status of receive data buffers and also error detection logic for receive errors such as overflow. See the Error Detection (Status) Interrupt section for more details.

When a SPORT is configured in receive mode, the transmit paths are deactivated and do not respond to serial clock or frame sync signals. As the transmit data buffers and transmit shift registers in the data paths are also deactivated, programs must not try to access them.

> **NOTE:** Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

Operating Modes and Options

The SPORT has a number of operating modes:

- Standard DSP Serial mode
- I^{2}S mode
- Left-Justified mode
- Right-Justified mode
- Multichannel (TDM) mode
- Packed I^{2}S mode

The SPORT halves within a SPORT top module can be independently configured in any of these operating modes unless they are coupled together using SPMUX logic, in which case they must be configured identically. Each half SPORT has its own set of control and data registers and is programmed similarly.

The ***Control Bits for SPORT Operating Modes*** table lists all the programmable configuration bits in the `SPORT_CTL_A` control register, which combine to determine the overall function and operating mode of the SPORT. The columns are arranged according to the setting of the `SPORT_CTL_A.OPMODE` bit that selects between standard DSP/multichannel modes and the various I^{2}S modes, and the cell contents are defined as follows:

- Yes – bit is programmable for this mode of operation and may be written
- Reserved – bit is not programmable for this mode of operation and must not be written
- = value – bit must be set to this value to enable this mode of operation
- FUNCTION – indicates alternate function for this bit in this mode

NOTE: When changing operating modes, first clear the `SPORT_CTL_A` register before again writing the register with the new configuration settings.

Table 34-10: Control Bits for SPORT Operating Modes

<table>
<tr><th>Name (Bit #)</th><th>Standard DSP Serial Mode</th><th>I^{2}S and Left-Justified Mode</th><th>Right-Justified Mode</th><th>Multichannel (TDM) Mode</th><th>Packed I^{2}S Mode</th></tr>
<tr><td>SPENPRI (0)</td><td colspan="5">Valid</td></tr>
<tr><td>DTYPE (2:1)</td><td>Valid</td><td colspan="2">Reserved</td><td colspan="2">Valid</td></tr>
<tr><td>LSBF (3)</td><td>Valid</td><td colspan="2">Reserved</td><td colspan="2">Valid</td></tr>
<tr><td>SLEN (8:4)</td><td colspan="5">Valid</td></tr>
<tr><td>PACK (9)</td><td colspan="5">Valid</td></tr>
<tr><td>ICLK (10)</td><td colspan="5">Valid</td></tr>
<tr><td>OPMODE (11)</td><td>= 0</td><td>= 1</td><td>= 1</td><td>= 0</td><td>= 1</td></tr>
<tr><td>CKRE (12)</td><td colspan="5">Valid</td></tr>
<tr><td>FSR (13)</td><td>Valid</td><td colspan="4">Reserved</td></tr>
<tr><td>IFS (14)</td><td colspan="5">Valid</td></tr>
<tr><td>DIFS (15)</td><td colspan="3">Valid</td><td colspan="2">Reserved</td></tr>
<tr><td>LFS (16)</td><td>Valid</td><td colspan="2">L_FIRST/PLFS</td><td>Valid</td><td>L_FIRST/PLFS</td></tr>
<tr><td>LAFS (17)</td><td>Valid</td><td>OPMODE2</td><td>Valid</td><td colspan="2">Reserved</td></tr>
<tr><td>RJUST (18)</td><td colspan="2">Reserved</td><td>= 1</td><td colspan="2">Reserved</td></tr>
<tr><td>FSED (19)</td><td>Valid</td><td colspan="2">Reserved</td><td>Valid</td><td>Reserved</td></tr>
<tr><td>TFIEN (20)</td><td colspan="5">Valid</td></tr>
<tr><td>GCLKEN (21)</td><td colspan="2">Valid</td><td colspan="3">Reserved</td></tr>
</table>

Table 34-10: Control Bits for SPORT Operating Modes (Continued)

Name (Bit #)	Standard DSP Serial Mode	I^2S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I^2S Mode
SPENSEC (24)	Valid				
SPTRAN (25)	Valid				

Serial Word Length

The SPORT uses the `SPORT_CTL_A.SLEN` field to determine the word length of the serial data to transmit or receive. Each half SPORT can independently handle word lengths up to 32 bits, and the value that must be programmed to the `SPORT_CTL_A.SLEN` field is obtained from:

SLEN = Desired SPORT word length - 1

The minimal word length depends on the selected operating mode. Words smaller than 32 bits are right-justified in the transmit or receive buffers; however, data can be shifted in or out in MSB or LSB first format, as configured by the `SPORT_CTL_A.LSBF` bit. The received word can also be sign-extended or zero-filled when storing the data to processor memory, as governed by the `SPORT_CTL_A.DTYPE` bit.

The ***Data Lengths for SPORT Operating Modes*** table shows the range of valid word lengths for each of the supported SPORT operating modes.

Table 34-11: Data Lengths for SPORT Operating Modes

Mode	SPORT Word Length (SLEN+1)
Standard DSP Serial	4–32
I^2S	5–32
Left-Justified	5–32
Right-Justified	5–32
Multichannel (TDM)	5–32
Packed I^2S	5–32

NOTE: If the companding feature is enabled on the datapath, it limits the word length settings. See Data Types and Companding for more details about word lengths required for companding. If more than 32 bits per frame sync are required to transmit or receive, use the multichannel mode to spread the data across numerous continuous channels.

Clock Sample and Drive Edges

The SPORT uses two control signals to sample or drive the serial data:

1. Serial clock (`SPORT_ACLK`) - bit clock for the serial data.
2. Frame sync (`SPORT_AFS`) - divides the incoming data stream into frames.

These control signals can be internally generated or externally provided, as determined by the `SPORT_CTL_A.ICLK` and `SPORT_CTL_A.IFS` bit settings, respectively.

Data and frame syncs can be sampled on the rising or falling edges of the SPORT clock signal, as determined by the `SPORT_CTL_A.CKRE` bit. By default, the `SPORT_CTL_A.CKRE` = 0 setting configures the falling edge of the `SPORT_ACLK` signal as the sampling edge for receive data and externally supplied frame syncs. The receive data and frame syncs can be sampled on the rising edges of `SPORT_ACLK` when `SPORT_CTL_A.CKRE` = 1.

NOTE: The SPORT drives transmit data and internal frame sync signals on the opposite serial clock edge of the sampling edge. Be sure to select the same value for `SPORT_CTL_A.CKRE` for transmit and receive functions for any two HSPORTs that are connected together, and always verify the correct polarity for any external device connected to the SPORT.

The ***Frame Sync and Data Driven on Rising Edge*** figure provides an example of the drive and sample edges when two HSPORTs are connected together, each with `SPORT_CTL_A.CKRE` = 0. In this example, the HSPORT that is configured as the transmitter drives the serial clock and frame sync signals, and both HSPORTs are configured for early, active high frame syncs and a word length of eight bits.

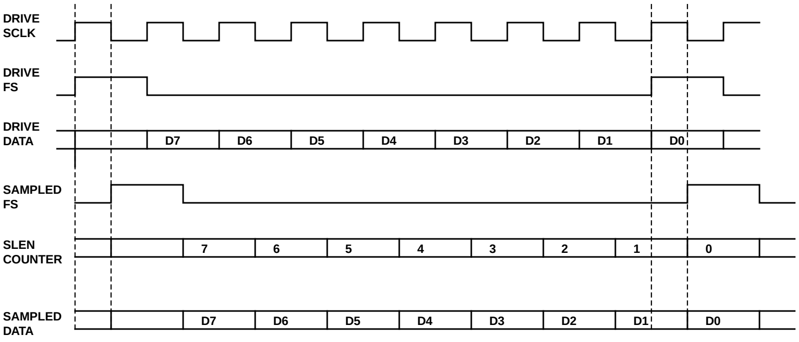


Figure 34-3: Frame Sync and Data Driven on Rising Edge

NOTE: The SCLK in the ***Frame Sync and Data Driven on Rising Edge*** figure is SCLK0_0.

As shown, the transmitting HSPORT provides the clock and generates the frame sync. Because the HSPORTs are configured for early frame mode, the first bit of data is driven one serial clock later, with subsequent bits being driven on the following rising clock edges in the signal train. When the receiving HSPORT samples the frame sync signal (as indicated in the SAMPLED FS waveform), the `SPORT_CTL_A.SLEN` bit counter is loaded with the `SPORT_CTL_A.SLEN` setting, after which each `SPORT_ACLK` decrements the `SPORT_CTL_A.SLEN` counter until the full word is received. In this figure, the DRIVE FS and SAMPLED FS waveforms show the frame sync required for the next word in a continuous data stream. Note that it is legal for this frame sync to be sampled synchronous to the last bit of the previous data being sampled, as the early frame mode means that the data lags the frame sync by one serial clock cycle. If the frame sync were sampled as asserted before the D0 bit is sampled, the frame sync error is logged in the receiver's status register.

Since the transmitter drives the internally-generated frame sync and data on the rising edge of the serial clock, the receiver must use the falling edge to sample the externally-supplied frame sync and data.

Frame Sync Options

The following sections provide details regarding the programmable aspects of the SPORT frame sync signal. See the specific operating mode sections for additional information regarding frame sync requirements and behavior for each specific operating modes.

Data-Dependent versus Data-Independent Frame Syncs

By default, the generation of a frame sync signal is data-dependent:

- When the SPORT is configured as a transmitter (`SPORT_CTL_A.SPTRAN` = 1), an internally generated transmit frame sync is output when a new data word has been loaded into the channel transmit buffer of the SPORT (by either the core or the DMA engine).
- When the SPORT is configured as a receiver (`SPORT_CTL_A.SPTRAN` = 0), an internally-generated receive frame sync is output only when the receive data buffer is not full.

The data-independent frame sync option, enabled by setting the `SPORT_CTL_A.DIFS` bit, allows for the generation of a periodic framing signal, regardless of the status of the data buffers. When this bit is set, the frame sync output will be continuous and periodic, according to the setting of the `SPORT_DIV_A.FSDIV` field.

Support for Edge-Detected and Level-Sensitive Frame Syncs

The level-sensitive nature of frame sync signals operates well in a noise-free environment. However, if noise corrupts the signals coming into the SPORT, the internal logic can lose synchronization. For example, excessive noise on the frame sync signal may cause the frame sync to be sampled as inactive on the clock edge that it is intended to be synchronous to, but then be sampled at the correct active level one cycle later. Similarly, a noisy clock signal can cause an unintended clock edge, resulting in potential premature sampling of the frame sync signal being applied to the pin.

The ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure describes a scenario where an external frame sync signal is corrupted due to noise, causing the receiving SPORT module to incorrectly sample the signal. If the frame sync is driven on the rising edge of the serial clock at t_A, the SPORT would normally sample the signal on the falling edge of the serial clock at t_B. Due to the noise, however, the SPORT misses the first edge of the frame sync and instead samples it at t_C.

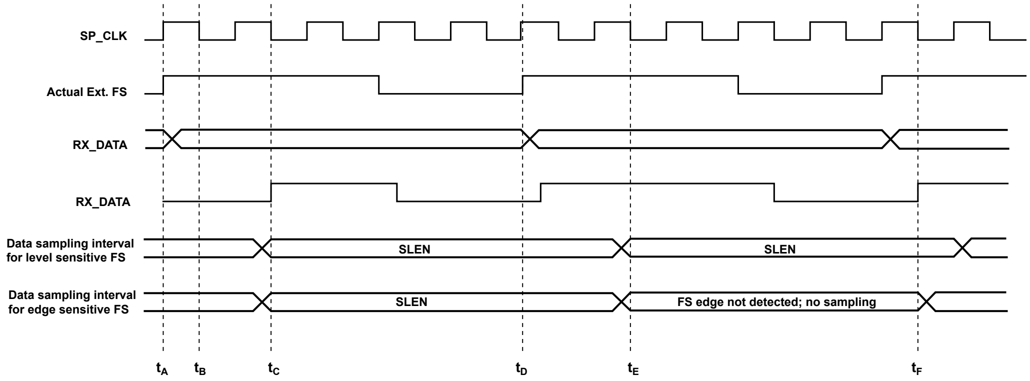

SP_CLK
Actual Ext. FS
RX_DATA
RX_DATA
Data sampling interval for level sensitive FS
SLEN
SLEN
Data sampling interval for edge sensitive FS
SLEN
FS edge not detected; no sampling
t_A
t_B
t_C
t_D
t_E
t_F

Figure 34-4: Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync

NOTE: SCLK in the ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure is SCLK0_0.

When the above occurs, the internal word length counter runs for a period equal to the `SPORT_CTL_A.SLEN` field of the control register, but it erroneously expires at t_E rather than at the appropriate point at t_D, thus receiving incorrect data. Further, if a new level-sensitive frame sync edge arrives at time t_D, the SPORT samples this framing signal again at t_E. As such, the frame sync sampling continues to be misaligned with the external data.

To help address this, the SPORT module provides an option to configure the frame sync signal to instead be edge-sensitive via the `SPORT_CTL_A.FSED` configuration bit. When this bit is set with active high frame syncs enabled (`SPORT_CTL_A.LFS` = 0), the rising edge of the frame sync is valid. Conversely, when the frame sync is active low (`SPORT_CTL_A.LFS`= 1), the falling edge is defined to be valid.

NOTE: `SPORT_CTL_A.FSED` is valid only in external frame sync mode. In internal frame sync mode, the setting of this bit is irrelevant and ignored.

In the above example, an edge-sensitive frame sync signal is not detected at t_E because the edge of the framing signal already occurred in the previous cycle (t_D) and there is no new edge to detect at t_E. As a result, the internal word length counter remains idle for this frame, thus ignoring the incorrect data, and the counter correctly resumes operation at t_F when a new frame sync edge is detected.

This activity sets the `SPORT_ERR_A.FSERRSTAT` bit and optionally generates a premature frame sync error interrupt.

Frame sync edge detection is used by default for stereo modes. MCM mode and DSP serial mode choose between edge detection and normal mode of FS detection.

NOTE: When the SPORT is first enabled, an already active externally applied frame sync will not commence operation. The SPORT will wait for a valid change in the frame sync's state from inactive to active before operation begins.

Early versus Late Frame Syncs

Frame sync signals can occur in the same serial clock cycle as the first bit of the data word (late) or one serial clock cycle before the first bit (early), as controlled by the `SPORT_CTL_A.LAFS` bit.

By default, the frame sync signal is configured to be early (`SPORT_CTL_A.LAFS` = 0). The first bit of the transmit data word will be driven one serial clock cycle after the frame sync is asserted (whether sensed externally or internally provided), and the first bit of the receive data word is expected to lag the frame sync by one serial clock cycle. The frame sync is not checked again until the entire word has been transferred.

If data transmission is continuous in early framing mode, then an internally-generated frame sync signal will be asserted (pulsed active for one serial clock cycle) synchronous to the last data bit of the current transfer, as the first bit of the next transfer will be immediately driven in the next serial clock cycle (no clocks are wasted). This event is not a premature frame sync error, so the `SPORT_ERR_A.FSERRSTAT` bit is not set.

The frame sync can alternatively be configured as late (`SPORT_CTL_A.LAFS` = 1), in which case the first bit of the transmit data word is available in the same serial clock cycle that the frame sync is asserted (whether sensed externally or internally provided), and the first bit of the receive data word is also latched in the same cycle. Serial clock edges latch the receive data bits, but the frame sync signal is checked only during the first bit of each word. Internally generated frame syncs remain asserted for the entire length of the data word in late framing mode.

The ***Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)*** figure illustrates these concepts.

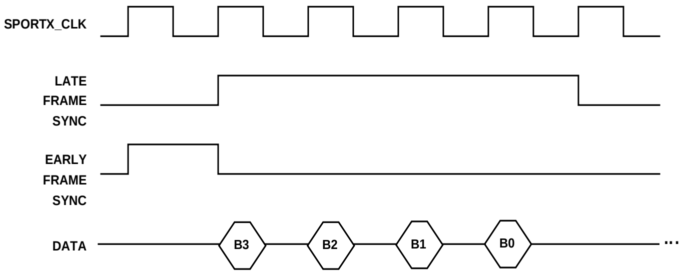


Figure 34-5: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

Framed versus Unframed Frame Syncs

The use of a frame sync signal is optional for SPORT operation, as controlled by the `SPORT_CTL_A.FSR` bit. When the frame sync is configured to be required (`SPORT_CTL_A.FSR` = 1), the data is defined to be framed (a frame sync signal must accompany every data word). To allow continuous transmission from the processor, ensure that a new data word is loaded into the transmit buffer before the ongoing transfer is completed (this is automatically cared for when DMA is used to transmit blocks of data).

Data words can be transferred continuously in what is referred to as unframed data mode, which is appropriate for continuous reception, by setting `SPORT_CTL_A.FSR` = 0. In this configuration, a single frame sync is still required to initiate communication, but it is subsequently unrequired once the communication begins. From that point onward, externally provided frame syncs are ignored and internally generated frame syncs are not driven. The ***Framed versus Unframed Data Stream*** figure shows the differences in SPORT operation between framed and unframed data modes with the frame sync configured to be early (`SPORT_CTL_A.LAFS` = 0).

NOTE: When DMA is enabled in a mode where frame syncs are not required, chaining can delay DMA requests. DMA requests are not always serviced frequently enough to guarantee continuous unframed data flow. Monitor status bits or check for a SPORT error interrupt to detect underflow or overflow of data.

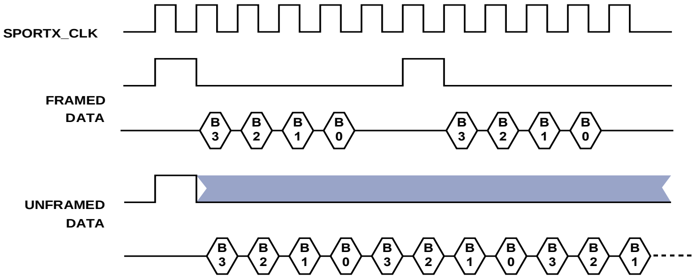


Figure 34-6: Framed versus Unframed Data Stream

Frame Sync Polarity

The framing signals can be active high or active low, as governed by the `SPORT_CTL_A.LFS` bit:

- When `SPORT_CTL_A.LFS` = 0, the corresponding frame sync signal is active high.
- When `SPORT_CTL_A.LFS` = 1, the corresponding frame sync signal is active low.

Active high is the default polarity of the frame sync signal.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external frame sync mode, any frame sync received during an active frame is premature and invalid. When this occurs, the `SPORT_ERR_A.FSERRSTAT` bit is set to indicate the framing error, and an optional error interrupt request can be generated for this event by setting the `SPORT_ERR_A.FSERRMSK` bit.

NOTE: The `SPORT_ERR_A.FSERRSTAT` bit is not set in the presence of uncleared underflow or overflow errors.

Refer to the ***Frame Sync Error Detection*** figure. The frame sync error bit gets set when an unexpected frame sync occurs during the ongoing data transfer (transmission or reception).

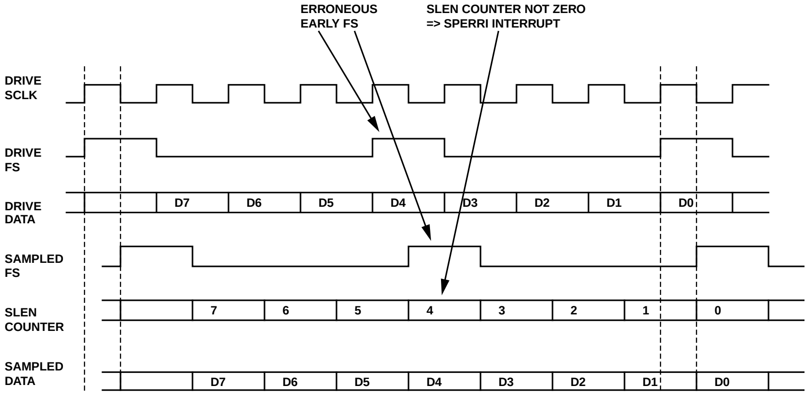


Figure 34-7: Frame Sync Error Detection

NOTE: SCLK in the ***Frame Sync Error Detection*** figure is SCLK0_0.

Whether a SPORT is receiving or transmitting, its bit count is set to the programmed serial word length when the frame sync is sampled, which is then decremented every subsequent serial clock cycle until the transfer has completed. At this point, the bit count reaches zero and will be reset to the programmed serial length when the next frame sync is sampled. As such, the bit count value is non-zero during an active transfer, and the frame sync error is asserted if a frame sync is sampled when this count is non-zero.

Mode Selection

The SPORT's operating mode is configured in the `SPORT_CTL_A` and `SPORT_MCTL_A` registers. The *SPORT Operating Modes* table provides specific guidance to properly program these SPORT control registers for the desired mode of operation.

Table 34-12: SPORT Operating Modes

Operating Modes	SPORT_CTL_A.OPMODE	SPORT_CTL_A.LAFS	SPORT_CTL_A.RJUST	SPORT_MCTL_A.MCE
Standard DSP Serial	0	Programmable	Reserved	0
I^2S	1	0	Reserved	0
Left-Justified	1	1	Reserved	0
Right-Justified	1	1	1	0
Multichannel	0	Reserved	Reserved	1
Packed I^2S mode	1	Reserved	Reserved	1

The following sections provide detailed information for each of the supported SPORT modes of operation.

Standard DSP Serial Mode

The SPORT can be configured in standard DSP serial mode by clearing the `SPORT_CTL_A.OPMODE` and `SPORT_MCTL_A.MCE` bits. This mode provides great flexibility in terms of programmable options to configure the SPORTs to communicate with various serial devices such as serial data converters and audio codecs. In order to properly connect to such devices, various clocking, framing, and data formatting options are available.

Timing Control Bits

Several bits in the `SPORT_CTL_A` control register define the configuration of the SPORT in standard DSP serial mode:

- SLEN: serial word length (4–32 bits)
- LSBF: shift LSB or MSB first
- ICLK: internally generated or externally provided serial clock
- CKRE: sample on rising or falling edge of the serial clock
- IFS: internally generated or externally provided frame sync
- FSR: framed or continuous operation
- DIFS: data-dependent or data-independent frame sync
- LFS: active high or active low frame sync
- LAFS: frame sync synchronous to data or one clock cycle before it
- PACK: 16-bit to 32-bit packing option

- GCLKEN: free-running or gated clock

Clocking Options

In standard DSP serial mode, the SPORTs can either accept an external serial clock or generate one internally, as controlled by the `SPORT_CTL_A.ICLK` bit. For internally generated serial clocks (`SPORT_CTL_A.ICLK` = 1), the `SPORT_DIV_A.CLKDIV` field configures the serial clock rate from the system clock.

The SPORT clock can also be gated, where it is only valid during an active transfer, as controlled by the `SPORT_CTL_A.GCLKEN` bit.

The SPORT clock edge used for driving and sampling of serial data and frame syncs is configured using the `SPORT_CTL_A.CKRE` bit:

- If `SPORT_CTL_A.CKRE` = 0, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If `SPORT_CTL_A.CKRE` = 1, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Stereo Modes

The SPORTs support three widely used stereo modes of operation:

- I^2S mode
- Left-Justified mode
- Right-Justified mode

In these modes, the serial data stream consists of left and right channels. The following sections describe these modes in more detail.

Channel Order

The active low frame sync (`SPORT_CTL_A.LFS`) bit is used to determine the polarity of the frame sync signal in the non-stereo modes of operation. For the stereo modes of operation, it instead controls whether the right or left channel is first in the data transfer. The ***Channel Order Bit Settings*** table shows which word is transmitted or received first, based on the setting of the `SPORT_CTL_A.LFS` bit.

Table 34-13: Channel Order Bit Settings

Mode	**`SPORT_CTL_A.LFS`**=0	**`SPORT_CTL_A.LFS`**=1
Left-Justified or Right-Justified	Left channel first	Right channel first
I^2S or Packed I^2S	Right channel first	Left channel first

I^{2}S Mode

I^{2}S mode is a commonly used stereo mode, where left and right channel data words are interleaved in the serial data stream and each transition of the frame sync signal is associated with one of the channels. The left channel data is transferred during the low segment of the frame sync signal, and the right channel data is transferred during the high segment of the frame sync signal. As such, the frame sync signal is considered to be a left-right (L/R) clock in this mode.

To set the SPORT up in I^{2}S mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.LFS` = 0
- `SPORT_MCTL_A.MCE` = 0

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to be compliant with the I^{2}S standard, but they can be otherwise configured to support non-standard operation as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 1 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The *Word Select Timing in I^{2}S Mode* figure shows the SPORT timing in I^{2}S mode. The data lags the L/R clock transition by one SCLK0_0 cycle, and the transfer begins with the left channel data word first.

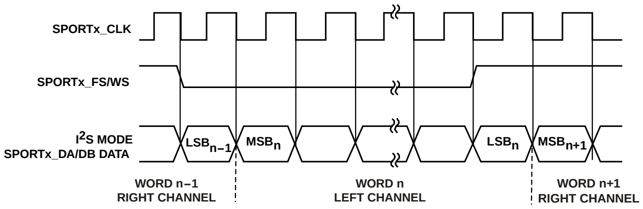


Figure 34-8: Word Select Timing in I^{2}S Mode

Left-Justified Mode

Left-justified mode is a stereo mode subset of the I^{2}S standard. As in I^{2}S mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between left-justified mode and standard I^{2}S mode is that the channel data is driven in the same bit clock cycle as the L/R clock transition (rather than one bit clock cycle later), such that the MSB is synchronous with the leading edge of the frame sync transition.

To set the SPORT up in left-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.LAFS` = 1
- `SPORT_MCTL_A.MCE` = 0

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in left-justified mode, but they can be otherwise configured as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 0 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The ***Word Select Timing in Left-Justified Mode*** figure shows the SPORT timing in left-justified mode. The start of a data sample is synchronous to the L/R clock transition, and the transfer begins with the left channel data word first.

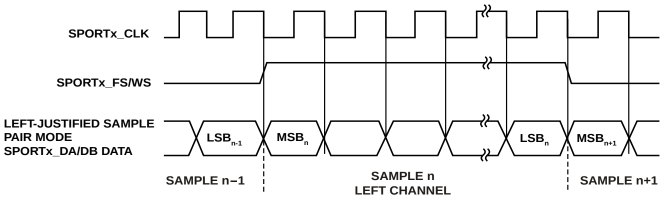


Figure 34-9: Word Select Timing in Left-Justified Mode

Right-Justified Mode

Right-justified mode is a stereo mode subset of the I^2S standard. As in I^2S mode and left-justified mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between right-justified mode and standard I^2S mode is that the LSB of the channel data ends at the point that the L/R clock transitions to frame the next sample (rather than one bit clock cycle after the L/R clock transition).

To set the SPORT up in right-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.RJUST` = 1
- `SPORT_MCTL_A.MCE` = 0

Timing Control Bits

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in right-justified mode, but they can be otherwise configured as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 0 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The *Word Select Timing in Right-Justified Mode* figure shows the SPORT timing in right-justified mode. The transmitter aligns the transmit data such that the last bit of the serial word is sent in the last clock cycle of the L/R clock (frame sync) signal marking the channels.

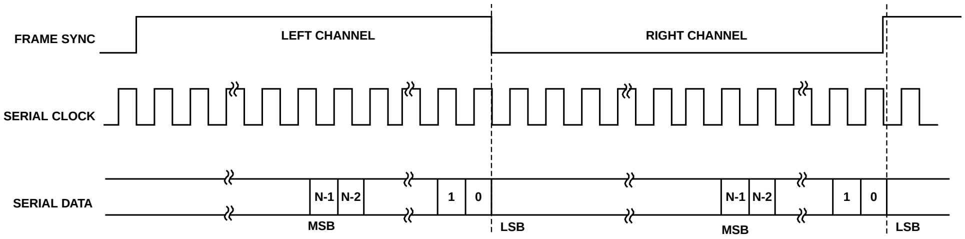

FRAME SYNC
LEFT CHANNEL
RIGHT CHANNEL
SERIAL CLOCK
SERIAL DATA
N-1
N-2
1
0
MSB
LSB
N-1
N-2
1
0
MSB
LSB

Figure 34-10: Word Select Timing in Right-Justified Mode

NOTE: For some SPORT-compatible ADCs or DACs such as the AD1871, right-justified mode is limited to commonly used ratios such as 64 FS and 128 FS. FS is the sampling frequency of ADCs and DACs, referrd to as the SPORT's L/R clock (frame sync) signal.

Consider the SPORT timing for right-justified mode, as shown in the ***Timing Comparison Between Different Stereo Modes*** figure. The frame sync width is limited to 32 SPORT clock periods (or 32 bits per channel) if:

- the SPORT's frame sync (L/R clock) runs at the FS rate, and
- the SPORT's serial bit clock runs at the 64 FS rate

The limitation applies to the frame sync width of either channel. If the data is confined to 24 bits, the SPORT introduces a 32–24=8-bit clock delay before it starts to transmit or capture data.

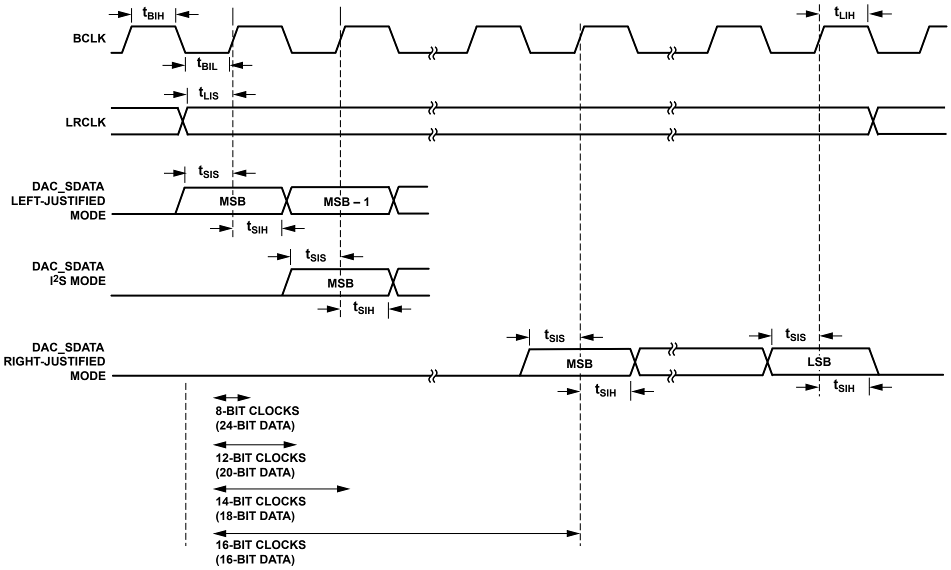

BCLK
t_{BIH}
t_{BIL}
t_{LIH}
LRCLK
t_{LIS}
DAC_SDATA LEFT-JUSTIFIED MODE
t_{SIS}
MSB
MSB – 1
t_{SIH}
DAC_SDATA I²S MODE
t_{SIS}
MSB
t_{SIH}
DAC_SDATA RIGHT-JUSTIFIED MODE
t_{SIS}
MSB
t_{SIH}
t_{SIS}
LSB
t_{SIH}
8-BIT CLOCKS (24-BIT DATA)
12-BIT CLOCKS (20-BIT DATA)
14-BIT CLOCKS (18-BIT DATA)
16-BIT CLOCKS (16-BIT DATA)

Figure 34-11: Timing Comparison Between Different Stereo Modes

Similarly, to support the 128 FS bit clock frequency, the frame sync width becomes 64 serial bit clock periods per channel. In this case, the delay can be a maximum of 59 bit clocks (64 – 5, which is the minimum serial data length in right-justified mode).

The starting point of the first bit is delayed so that the LSB of the serial data aligns properly with the end of the channel. A 6-bit counter is added for this purpose in the stereo mode counter, which is programmed by writing the least significant six bits of the `SPORT_MCTL_A.WOFFSET` field. Though this is a multichannel mode configuration register, the SPORT uses these bits in right-justified mode to configure the offset from the transition of the L/R clock to where the first data bit must be driven in order to have the end of the last bit align properly with the next L/R clock transition. Software must program this register with the appropriate delay.

Multichannel (TDM) Mode

The multichannel mode of SPORT operation allows the SPORT to communicate as part of a time division multiplexed (TDM) serial system. In TDM communications, a large frame of streamed serial data words is defined to be a particular length. It consists of a specific number of channels, and each channel contains one serial data word of the defined data length. For example, a 24-word block of 24-bit data can be defined to be a window within a frame, having a duration of 576 bit clocks and comprised of 24 continuous channels. The SPORT is configured to transfer on specific channels within a defined window in this frame.

To set the SPORT up in multichannel mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 0
- `SPORT_MCTL_A.MCE` = 1

In multichannel mode, the SPORT can selectively transfer data on up to a maximum window size of 128 continuous channels out of a maximum 1024-channel frame while ignoring all the disabled channels within the window and all the channels outside the window. The SPORT can do any of the following on each channel:

- Transmit data (`SPORT_CTL_A.SPTRAN` = 1)
- Receive data (`SPORT_CTL_A.SPTRAN` = 0)
- Do nothing (during inactive channels)

Channel selection is configured in the half SPORT multichannel select registers (`SPORT_CS0_A` - `SPORT_CS3_A`) before enabling SPORT operation for multichannel mode. Programming of these registers is especially important in DMA data unpacked mode, since the SPORT data buffers begin operation immediately after the SPORT data lines are enabled. Be sure to enable multichannel operation (set the `SPORT_MCTL_A.MCE` bit) prior to enabling the SPORT itself.

Clocking Options

In multichannel mode, the SPORTs can either accept an external serial clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` bit. For an internally-generated serial clock (`SPORT_CTL_A.ICLK` = 1), the `SPORT_DIV_A.CLKDIV` bit field is used to configure the serial clock rate, as derived from the system clock.

The serial clock edges used to drive and sample data and frame syncs are also configurable using the `SPORT_CTL_A.CKRE` bit:

- If `SPORT_CTL_A.CKRE`= 0, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If `SPORT_CTL_A.CKRE` = 1, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Frame Sync Options

The frame sync signal synchronizes the channels and restarts each multichannel sequence, starting with the channel 0 data word. For internally-generated frame syncs (`SPORT_CTL_A.IFS` = 1), the frame sync period in multichannel mode is defined as:

FS period = [(`SPORT_CTL_A.SLEN` + 1) × number of channels] - 1

The active level for the frame sync signal is also configurable by programming the `SPORT_CTL_A.LFS` bit. Set this bit to make the frame sync an active low signal, and clear it to make it active high.

In multichannel mode, frame sync timing resembles late framing mode (although the `SPORT_CTL_A.LAFS` bit is reserved in this mode). The first bit of the transmit data word is driven and the first bit of the receive data word is sampled in the same serial clock cycle as the frame sync, provided there is no programmed frame delay (`SPORT_MCTL_A.MFD` = 0).

Once the frame sync signal is asserted, word transfers are performed continuously for the duration of the active window, and no further frame syncs are required for different channels within the window. As such, internally-generated frame syncs are always data-independent, and the `SPORT_CTL_A.DIFS` bit is reserved.

Transmit Data Valid (TDV)

Each SPORT features a transmit data valid signal (`SPORT_ATDV`), which is driven high during enabled transmit channels. Because the SPORT output data signals are three-stated during inactive channels, the `SPORT_ATDV` signal signifies when the processor is actively driving the SPORT data outputs, thus serving as an output-enable signal for the data transmit pin(s).

Active Channel Selection Registers

In multichannel mode, the SPORT supports a window size of up to 128 channels for transmitting or receiving data, where it can selectively receive or transmit data in any of these 128 channels. Each channel can be individually enabled or disabled using the multichannel selection registers (`SPORT_CS0_A` to `SPORT_CS3_A`) to select the channels in which to transfer data during a multichannel communication stream. Data words associated with enabled channels are transmitted or received in the respective channels, while disabled channels cause a transmit SPORT to three-state the data output pins and a receive SPORT to ignore the data.

The four 32-bit multichannel selection registers combine to form up to a 128-bit meta-register to accommodate the maximum window size of 128 channels. Setting any bit within these registers enables the associated channel. The 128 channels are sequentially numbered from bit 0 in the `SPORT_CS0_A` register (corresponding to channel 0 of the window) to bit 31 of the `SPORT_CS3_A` register (corresponding to channel 127 of the window). For example, setting bit 13 of the `SPORT_CS1_A` register enables channel number 45 (add 32 for the channels in the `SPORT_CS0_A`). Likewise, setting bit 5 of the `SPORT_CS3_A` register enables channel number 101 (add 96 for the 32 channels in each of the `SPORT_CS0_A`, `SPORT_CS1_A`, and `SPORT_CS2_A` registers).

Multichannel Frame Delay (MFD)

The multichannel frame delay (`SPORT_MCTL_A.MFD`) field specifies the delay in serial bit clocks between the frame sync pulse and the first data bit in the frame. This configurability allows the processor to work with different types of telephony interface devices.

As `SPORT_MCTL_A.MFD` is a 4-bit field, the maximum value allowed for the frame delay is 15 serial clock cycles. When set to 0, the frame sync is concurrent with the first data bit. If `SPORT_MCTL_A.MFD`>0, a new frame sync can occur during the last channel(s) of a previous frame and still be valid (does not cause a frame sync error).

NOTE: If the required frame delay exceeds 15 serial clocks, use the window offset field (`SPORT_MCTL_A.WOFFSET`) to delay the start of channel 0 in increments of the serial word length, and then adjust `SPORT_MCTL_A.MFD`accordingly. For example, if the serial word length is 12 bits and the desired frame delay is 16 serial clock cycles, set the `SPORT_MCTL_A.WOFFSET` to 1 to insert a 12-bit delay after the frame sync to where the channel 0 data begins, and then program `SPORT_MCTL_A.MFD` to 4 (i.e., 16 - 12).

Window Size (WSIZE)

Select the number of channels used in multichannel operation by programming the 7-bit `SPORT_MCTL_A.WSIZE` field. This field must be set to the actual number of channels minus one (`SPORT_MCTL_A.WSIZE` = Number of channels -1).

The 10-bit `SPORT_MSTAT_A.CURCHAN` field holds the channel number currently being serviced during multichannel operation.

Window Offset (WOFFSET)

The window offset (`SPORT_MCTL_A.WOFFSET`) field specifies where in the 1024-channel frame to place the start of the active window (up to 128 channels long). A value of 0 specifies no channel offset from the frame sync (channel 0 immediately follows it). Any non-zero value indicates the number of channels that come between the frame sync and the start of channel 0 of the active frame, with 896 (for example, 1024–128) being the largest value that permits using all 128 channels.

As an example, a program could define an active window comprised of eight channels (`SPORT_MCTL_A.WSIZE` = 7) with a window offset of 93 (`SPORT_MCTL_A.WOFFSET` = 93). If configured in this fashion, the 8-channel window that the SPORT will transfer within resides in the channel range from 93 to 100 in the up-to-1024-channel frame.

Do not change the window offset or the number of multichannel slots (`SPORT_MCTL_A.WSIZE`) while the SPORT is enabled. If the combination of the window size and offset place any portion of the window out-of-range relative to the channel counter, none of the channels are enabled.

Companding Selection

Like the other operating modes, companding logic can optionally be applied to serial data (compression logic for transmit mode or expansion logic for receive mode). The two widely used companding algorithms, A-law and μ-law, are selectable using the `SPORT_CTL_A.DTYPE` field.

If companding is enabled, the companding algorithm is applied to both the primary and secondary datapaths. In multichannel mode, companding can be applied to either all or none of the enabled channels (companding cannot be selected on a per-channel basis).

Multichannel DMA Data Packing (MCPDE)

Multichannel DMA data packing and unpacking are enabled using the `SPORT_MCTL_A.MCPDE` bit.

When set, data is packed, and the SPORT expects the data in the DMA buffer to correspond only with enabled SPORT channels. For example, if only channels 1 and 9 are enabled in a 10-channel window (`SPORT_MCTL_A.WSIZE` = 9), the SPORT expects the buffer to be exactly two words in length, where channel 1 is associated with the first element in the buffer and channel 9 is associated with the second.

When cleared, data is unpacked, and the SPORT expects the DMA buffer to have a word for each of the channels in the active window, whether the channel is enabled or not. As such, the DMA buffer size must be exactly the size of the window. Using the same example as the packed case above, if only channels 1 and 9 are enabled in a 10-channel window (`SPORT_MCTL_A.WSIZE` = 9), then the DMA buffer size is ten words. The data at offsets 1 and 9 within the buffer are associated with the data transfers of channels 1 and 9, respectively. The rest of the words in the buffer are unused.

Packed I^{2}S Mode

The SPORT supports a packed I^{2}S mode, which can be used for audio codec communications using multiple channels. This mode allows applications to send more than the standard 32 bits per channel available through standard I^{2}S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode).

To set the SPORT up in packed I^{2}S mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_MCTL_A.MCE` = 1

Like multichannel mode, packed I^{2}S mode also supports a maximum of 128 channels, where up to 128 channels of data can be transferred for every transition of the frame sync signal acting as an L/R clock (for example, up to 128 left-channel words transfer during the high portion of the L/R clock, and up to 128 right-channel words transfer during the low portion).

As shown in the ***Packed I^{2}S Mode 128 Operation*** figure, the packed waveforms are the same as those waveforms used in multichannel mode, except the frame sync is toggled for every frame and emulates I^{2}S mode.

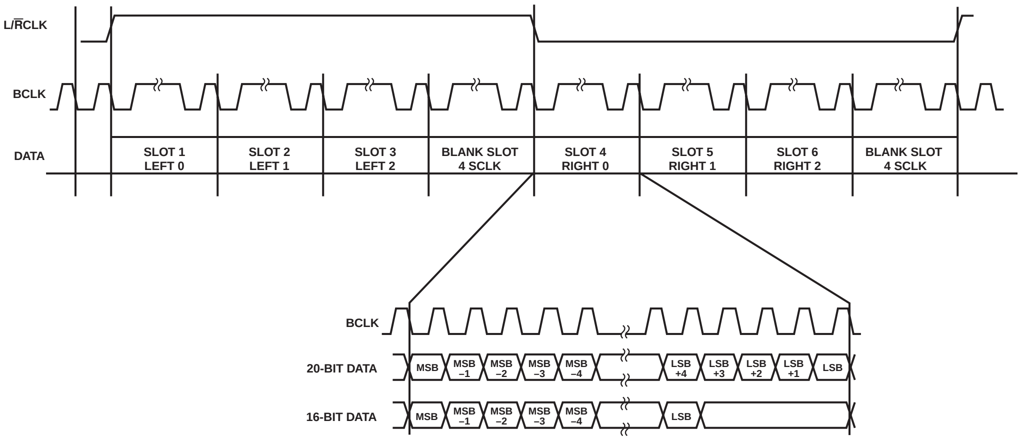

L/RCLK
BCLK
DATA
SLOT 1 LEFT 0
SLOT 2 LEFT 1
SLOT 3 LEFT 2
BLANK SLOT 4 SCLK
SLOT 4 RIGHT 0
SLOT 5 RIGHT 1
SLOT 6 RIGHT 2
BLANK SLOT 4 SCLK
BCLK
20-BIT DATA
MSB
MSB −1
MSB −2
MSB −3
MSB −4
LSB +4
LSB +3
LSB +2
LSB +1
LSB
16-BIT DATA
MSB
MSB −1
MSB −2
MSB −3
MSB −4
LSB

Figure 34-12: Packed I^{2}S Mode 128 Operation

Serial Bit Clock Options

In packed I^{2}S mode, the SPORTs can either accept an external serial bit clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` configuration bit. For an internally-generated serial bit clock (`SPORT_CTL_A.ICLK` = 1), use the `SPORT_DIV_A.CLKDIV` bit field to configure the serial bit clock rate from the system clock.

The serial bit clock edge that is used for sampling or driving data and frame syncs is programmable using the `SPORT_CTL_A.CKRE` bit.

L/R Clock (Frame Sync) Options

The frame sync period in packed I^{2}S mode is defined as:

FS period = [(`SPORT_CTL_A.SLEN` + 1) × number of channels] - 1.

The L/R clock can be supplied externally or internally generated depending on the `SPORT_CTL_A.IFS` bit setting. The logic level of the L/R clock associated with the left and right channel data can be changed using the `SPORT_CTL_A.LFS` configuration bit.

Gated Clock Mode

Some system components such as ADCs and DACs utilize a SPI-compatible protocol for the interface. To communicate with such devices, the SPORT must support a gated clock, where the data valid information is embedded in the clock (for example, the clock only toggles when data is valid). This gated clock feature is enabled using the `SPORT_CTL_A.GCLKEN` bit.

To enable the gated clock mode of operation, program the SPORT to comply with the following requirements.

- Do not enable gated clock functionality in right-justified or multichannel mode
- Gated clock mode has the following requirements for other control bits:

  - The serial clock and frame sync signals must have the same source (SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS)
  - Unframed mode is not supported (SPORT_CTL_A.FSR must be set)
  - Clear the SPORT_CTL_A.DIFS bit in transmit mode; set it in receive mode
- Satisfy the following necessary conditions when gated clock mode is enabled:
  - Seven serial clock cycles are required between enabling the SPORT and the first frame sync. If this requirement is not met, the SPORT can drop the first data (for subsequent data, this requirement is not applicable)
  - For externally-provided clock and frame sync, the frame sync must be inactive during clock synchronization after the SPORT has been enabled
  - For an edge-detected frame sync (SPORT_CTL_A.FSED = 1), the frame sync must transition back to the inactive state before the current word transfer is complete (or when the clock is still running). If this requirement is not met, the SPORT does not recognize the next valid frame sync and skips the channel. The SPORT continues to skip the frame syncs until the frame sync transitions back to an inactive state while the clock is active.

Data Transfers and Interrupts

SPORT data can be transferred to or from internal or external memory by two methods:

- Core-driven, single-word transfers
- DMA-driven, multiple-word transfers (optionally with multiple work units)

Core-driven transfers use SPORT interrupts to signal the processor core to perform MMR-based single-word transfers to or from the SPORT data buffers. DMA can be set up to automatically transfer a configurable number of serial words between the SPORT transmit/receive data buffers and memory, and then generate a data completion interrupt request when a work unit or a series of work units completes, thus signaling by the SEC to the processor core that a block of data has been transferred.

The following sections provide information on core-driven and DMA-driven data transfers.

Data Buffers

When programming the serial port data channels (primary or secondary) as a transmitter by setting SPORT_CTL_A.SPTRAN =1, only the corresponding transmit data buffers (SPORT_TXPRI_A and SPORT_TXSEC_A) become active. The receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) remain inactive. Similarly, when the SPORT data channels are programmed for receive operation (SPORT_CTL_A.SPTRAN =0), then only corresponding receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) are active. Do not attempt to read or write inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results can occur.

Each of these buffers is 32-bit wide (corresponds to maximum serial data word length). When using word lengths less than 32 bits for SPORT operation, the data in these buffers is automatically right-justified. (The LSB bit of data is at the bit 0 location of the buffer). The upper unused bits can be zero-filled or sign-extended depending on `SPORT_CTL_A.DTYPE` field.

Transmit Data Buffers (`SPORT_TXPRI_A` and `SPORT_TXSEC_A`)

When enabled as a transmitter (`SPORT_CTL_A.SPTRAN` =1), each SPORT half has its own set of transmit data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred to as `SPORT_TXPRI_A` and `SPORT_TXSEC_A` respectively.

These transmit data buffers are 32 bits wide. Load these buffers with the data for transmission on the primary and secondary data channels. The DMA controller loads the data automatically. Or, the program running on the processor core loads the data manually.

Together with the output shift register, transmit data buffers act like a two-location FIFO. If data packing is disabled (`SPORT_CTL_A.PACK` =0), the transmit path can hold as many as three data words. If data packing is enabled (`SPORT_CTL_A.PACK` =1), it can hold two packed data words at any given time.

When the transmit shift register becomes empty (transfer out all the bits of previous word), data in the transmit data buffer is automatically loaded into it. An interrupt occurs when the output transmit shift register has been loaded, signifying that the transmit data buffer is empty and ready to accept the next word. This interrupt does not occur when serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not write to the inactive secondary transmit data buffer and conversely. If the core keeps writing to the inactive buffer, the status of that transmit buffer becomes full. This state can cause the core to hang indefinitely, since data is never transmitted to the output shift register.

Receive Data Buffers (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`)

When enabled as receiver (`SPORT_CTL_A.SPTRAN` =0), each SPORT half has its own set of receive data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred as `SPORT_RXPRI_A` and `SPORT_RXSEC_A` respectively. Together with input shift register, the receive data buffers act like a three-location FIFO, as the receive path has two data registers.

These receive data buffers are the 32 bits wide. These buffers are automatically loaded from the receive shift register when a complete word has been received into it. An interrupt occurs when the receive data buffer is loaded, signifying that new data is available in the receive data buffer and is ready to read. This interrupt does not occur when the serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not read from the inactive secondary receive data buffer and conversely. If the core keeps reading from the inactive buffer, the status of that receive buffer becomes empty. This state can cause the core to hang indefinitely since new data is never received through the input shift register.

Data Buffer Status

The SPORT provides status information about its primary and secondary data buffers through the `SPORT_CTL_A.DXSPRI` and `SPORT_CTL_A.DXSSEC` bits, respectively. It also provides error status information through the corresponding `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` bits, respectively. Depending on the `SPORT_CTL_A.SPTRAN` bit setting, these bits reflect the status of either the pair of transmit (`SPORT_TXPRI_A` and `SPORT_TXSEC_A`) or receive (`SPORT_RXPRI_A` and `SPORT_RXSEC_A`) buffers, indicating whether the buffer is full, partially full, or empty.

When attempting to read from an empty receive buffer or write to a full transmit buffer, the SPORT delays access until the buffer is ready, potentially resulting in excessive MMR bus response times. To avoid this when doing core-driven transfers, always check the buffer status to determine if the access can be made. The SPORT updates the status bits in the `SPORT_CTL_A` register during reads and writes by the core processor.

NOTE: These status bits are updated during reads and writes from the core processor even when the SPORT is disabled.

Two complete 32-bit words can be stored in the receive buffer while a third word shifts in. Therefore, almost three complete words can be received without the receive buffer being read before an overflow occurs. After receiving the third word completely, the shift register contents overwrite the second word, which will occur if the first word has not yet been read by the processor core or the DMA controller. This receive overflow condition is flagged through the error status bits of the `SPORT_CTL_A` register on the last bit of the third word.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less, the received data words can be packed into a 32-bit word. Similarly, if the SPORT is configured as a transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted can be unpacked into 16-bit words. The `SPORT_CTL_A.PACK` bit is used to select this packing or unpacking feature.

When `SPORT_CTL_A.PACK` = 1, two consecutive received words are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This packing method applies to both receive (packing) and transmit (unpacking) operations. In this case, the transmit and receive interrupt requests are generated for the 32-bit packed words, not for each 16-bit word.

NOTE: When 16-bit received data is packed into 32-bit words and stored in normal word space in the processor's internal memory, the 16-bit words can be read or written using short word space addressing.

Single-Word (Core) Transfers

The SPORTs can transmit or receive individual data words with interrupt requests occurring as each data word is transferred. When a SPORT is enabled with the corresponding DMA channel disabled, interrupt requests are generated when:

- a complete word has been received in the receive data buffer or

- the transmit data buffer is not full

When performing core transfers, be sure to access only those buffers that are associated with enabled datapaths, as governed by the transfer direction (`SPORT_CTL_A.SPTRAN`) bit and the primary/secondary (`SPORT_CTL_A.SPENPRI`/`SPORT_CTL_A.SPENSEC`) data enable bits. If inactive SPORT data buffers are read from or written to by the core while the SPORT is enabled, the core can hang. For example, if a half SPORT is programmed to be a transmitter and the core reads from one of the receive buffers associated with that half SPORT, the core can hang as if it were reading an empty buffer that is active and awaiting new data to arrive. Because this is a transmitting HSPORT, that data will never arrive, thus locking the core up until the SPORT is reset. To avoid such a situation, be sure to check the status of the appropriate data buffer before attempting a core access to it by interrogating the `SPORT_CTL_A.DXSPRI` or `SPORT_CTL_A.DXSSEC` status bits.

DMA Transfers

Direct memory access (DMA) provides a mechanism for transferring an entire block of serial data before an interrupt is generated. The processor's on-chip DMA controller automatically handles the DMA transfer, thus allowing the processor core to run in parallel until the entire block of data is transferred. When the interrupt request occurs, a service routine can then process the entire block of data (rather than react to single words), thus significantly reducing overhead.

Each half SPORT has a dedicated DMA channel that serves both the primary and secondary datapaths. When configured as a transmitter (`SPORT_CTL_A.SPTRAN` = 1) with both the primary and secondary datapaths enabled (`SPORT_CTL_A.SPENPRI` = `SPORT_CTL_A.SPENSEC` = 1), the DMA channel requires that the source DMA buffer interleave the data beginning with the primary channel, as it will alternately load to the primary and secondary transmit data buffers once it is enabled. The complementary operation is true in receive mode (`SPORT_CTL_A.SPTRAN` = 0) when both datapaths are enabled, as the DMA channel alternately reads from the primary and secondary receive data buffers and interleaves them in the destination DMA buffer. As such, software must de-interleave the data corresponding to the primary and secondary channels from the receive DMA buffer.

If the SPORT is configured in stereo mode, the same DMA channel handles both the left and right channels of both datapaths (primary and/or secondary). Therefore, for a transmit DMA with only one datapath enabled, the source buffer must be populated such that the left- and right-channel data is interleaved. If both datapaths are enabled, the DMA channel alternately loads to the primary and secondary transmit data buffers once it is enabled. As such, the interleaving requirement is for the primary left-channel data to be followed by the secondary left-channel data, then the primary right-channel data, and finally the secondary right-channel data. The complementary operation is true in receive mode, where the DMA channel alternately reads from the primary and secondary receive data buffers and interleave them in the destination DMA buffer. For the stereo modes of operation, the destination DMA buffer is interleaved as left-right data for a single data input. If both datapaths are enabled, the destination DMA buffer is written with the primary and secondary left-channel data followed by the primary and secondary right-channel data. As such, software must de-interleave the primary and secondary left- and right-channel data from the receive DMA buffer, as defined by this scheme.

Since both the primary and secondary datapaths share the single DMA channel, each half SPORT has a single interrupt request for data completion, as well as an error interrupt request. The DMA controller can generate an interrupt request at the end of a chain of DMA work units (when using multiple descriptors) or at the end of individual DMA work unit.

The SPORT DMA channels are assigned a higher priority than all the other DMA channels (for example, the SPI port). Having higher priority causes the SPORT DMA transfers to execute first when multiple DMA requests occur in the same cycle. The SPORT DMA channels are numbered and prioritized in the DMA channel list table in the DMA chapter.

Although the most efficient DMA transfers execute with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by `SPORT_CTL_A.SLEN` field). If the serial data length is 16 bits or smaller, two pieces of data can be packed into 32-bit words for each DMA transfer, as selected by setting the `SPORT_CTL_A.PACK` bit. When this bit is set, the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit word. For more information, see the Data Buffer Status section.

NOTE: The SPORT DMA channel can access both internal memory and external memory of the processor without any core overhead.

Data Transfer Interrupt

Each half SPORT features a data transfer interrupt request that is shared by both the primary and secondary data channels in both transmit and receive modes. To determine the source of the data transfer interrupt request, applications can check the primary and secondary data buffer status bits (`SPORT_CTL_A.DXSPRI` and , `SPORT_CTL_A.DXSSEC` , respectively).

When using core-driven transfers, this interrupt's meaning depends on the direction of the SPORT:

- As transmitter (`SPORT_CTL_A.SPTRAN` = 1) - the transmit data buffer is empty
- As receiver (`SPORT_CTL_A.SPTRAN` = 0) - new data is available in the receive data buffer

NOTE: When data packing is enabled (`SPORT_CTL_A.PACK` = 1), the core-driven transmit and receive interrupt requests are generated for 32-bit packed words, not for each 16-bit word.

In both cases, the interrupt request can be used to signal the core that an individual transfer has completed. For transmit operations, it indicates that the transmit data buffer can be safely loaded (either the buffer is already empty or the last data has moved from the data buffer to the shift register). For receive operations, it indicates that new data has arrived and can be read (or must be read before a subsequent word overwrites it).

When the SPORT is configured to use DMA to move data between memory and the peripheral (the most generic way to use dedicated DMA for sport data transfers), the same data transfer interrupt request instead indicates the completion of the transfer of a block of serial data (rather than a single word). When DMA is used, the DMA count register must be initialized to specify the number of words to transfer. This count decrements after each DMA transfer on the channel, and the data transfer interrupt request signal is asserted when the word count reaches zero (for example, a DMA work unit has finished).

For transmit DMA, the interrupt request is raised when the last word in the DMA work unit is loaded from the source memory to the HSPORT FIFO. This interrupt request can signal to the core that a new DMA work unit can be configured or that other software threads can now run. The transmit interrupt request can optionally be deferred until the last word of the work unit has fully shifted out of the shift register (see the Transfer Finish Interrupt (TFI) section for details).

For receive DMA, the interrupt request is raised when the last word is loaded to the destination memory. In addition to that described for transmit DMA, this interrupt request also serves as an indication to the core that there is a buffer of newly acquired data that is ready to be processed.

See the DMA chapter for further details regarding enabling of the DMA interrupt requests associated with the various modes of DMA operation.

NOTE: As a single DMA channel services both the primary and secondary datapaths associated with the SPORT, there is a single DMA completion interrupt request.

Transfer Finish Interrupt (TFI)

When configured for transmit DMA (`SPORT_CTL_A.SPTRAN` =1), the data transfer interrupt request gets generated by the DMA engine itself when it decrements its count register upon loading the last element from memory to the HSPORT hardware. Alternately, the SPORT can use a Transmit Finish Interrupt (TFI) to signal the actual end of the transmission (for example, when the last bit of the last data word of the buffer has shifted out of the SPORT to the system) by setting the `SPORT_CTL_A.TFIEN` bit. When this bit is set, then DMA signal that would normally assert the data transfer interrupt request instead signals the SPORT that the DMA work unit is complete. The SPORT then waits until all the data in the FIFO is shifted out (including the transmit shift register) and asserts the TFI interrupt request upon completion.

NOTE: To enable this functionality in the DMA engine, be sure to configure the interrupt type field in the DMA configuration register for Peripheral interrupt. See the DMA chapter for further details.

Error Detection (Status) Interrupt

In addition to the dedicated data transfer interrupt request, each half SPORT also features an optional error status interrupt request that can be triggered when error conditions occur relative to data or frame syncs associated with the half SPORT.

Data-related errors depend on the direction of the SPORT and reflect overflow or underflow conditions, which are depicted in the `SPORT_CTL_A` control register as read-only sticky bits `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` (for the primary and secondary channels, respectively).

- When the SPORT is configured as a transmitter, these bits provide transmit data buffer underflow status. When the frame sync signal occurs when the transmit data buffer is empty, the underflow bit corresponding with the offending transmit data buffer is set, as the SPORT will transmit data whenever it detects a valid frame sync signal, whether new data is present or not.
- When the SPORT is configured as a receiver, these bits provide receive overflow status. When a channel receives new data while the receive buffer is already full, the new data overwrites the existing data, thus causing

an overflow. When this occurs, the overflow bit corresponding with the offending receive data buffer is set, as the SPORT receives data whenever it detects a valid frame sync signal, whether there is room in the receive buffer or not.

Each half SPORT also features an error register (SPORT_ERR_A), which is the source for the assertion of the described data-related error status bits. When a data-related error occurs on the primary or secondary datapaths, the error is logged in the SPORT_ERR_A.DERRPSTAT or SPORT_ERR_A.DERRSSTAT bits, respectively. To enable these status bits to generate the HSPORT status interrupt request in the SEC, the corresponding SPORT_ERR_A.DERRPMSK and SPORT_ERR_A.DERRSMSK bits must be set (for the primary and secondary datapaths, respectively).

The SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC channel error status bits are sticky read-only bits that can be cleared in two ways:

- Reset the error detection logic by disabling the channel associated with the error condition (clear the SPORT_CTL_A.SPENPRI or SPORT_CTL_A.SPENSEC control bit).
- Clear the source of the interrupt by writing-1-to-clear the SPORT_ERR_A.FSERRSTAT, SPORT_ERR_A.DERRPSTAT, or SPORT_ERR_A.DERRSSTAT status bits.

In addition to data-related errors, SPORT_ERR_A also tracks frame sync errors in the SPORT_ERR_A.FSERRSTAT status bit. Similar to the data-related errors, the frame sync error can be enabled as a source for raising the error status interrupt request via the SEC by setting the SPORT_ERR_A.FSERRMSK bit. A frame sync error occurs when the frame sync is detected prematurely, as explained in the Premature Frame Sync Error Detection section.

A frame sync error is not detected in the following cases:

- When there is no active transmit or receive data, and the frame sync pulse occurs due to noise on the input signal – if there is no active transfer, a noise-induced frame sync pulse is valid.
- If there is an active underflow or overflow error – frame sync errors cannot be detected because the SPORT error logic does not run after one of the data errors has occurred and remains unserviced.
- When the frame sync pulse doesn't meet minimum timing requirements – if the frame sync pulse is shorter than a SPORT clock period, there is no guarantee that it gets sampled at all and may go unnoticed.

SPORT Programming Model

The following sections provide programming guidance for setting up the SPORTs for use in an application:

- Initializing Core-Driven (Non-MCM) Transfers
- Initializing Multichannel Transfers
- Using DMA for SPORT Transfers
- Using Companding as a Function

Initializing Core-Driven (Non-MCM) Transfers

The following programming model applies to all of Standard DSP Serial Mode, I^{2}S Mode, Left-Justified Mode, and Right-Justified Mode for core-driven transfers. More steps are required to properly initialize the SEC to service the SPORT interrupts (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Clear the `SPORT_CTL_A` and `SPORT_MCTL_A` configuration registers.

   ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multichannel logic) is fully reset before attempting to reprogram it.

2. Optionally program the `SPORT_DIV_A` clock divisor register.

   ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the Serial Clock and Frame Sync sections.

3. Program the `SPORT_CTL_A` primary configuration register.

   ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the Operating Modes and Options section for details). Do not set the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` buffer enable bits in this step.

4. Optionally program the `SPORT_CTL2_A` secondary configuration register.

   ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the Multiplexer Logic section for details).

5. Optionally program the `SPORT_ERR_A` error register.

   ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt is desired (see the Error Detection (Status) Interrupt section for details).

6. For Right-Justified mode only, program the `SPORT_MCTL_A.WOFFSET` field.

   ADDITIONAL INFORMATION: In Right-Justified mode, this field serves as the delay count (`DCNT`) required to align the LSB of each stereo channel with the L/R clock transition and must be programmed manually (see the Right-Justified Mode section for details).

7. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

   ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate.

8. Write data to be transmitted to the transmit buffer (`SPORT_TXPRI_A` and/or `SPORT_TXSEC_A`) *or* read data that has been received from the receive buffer (`SPORT_RXPRI_A` and/or `SPORT_RXSEC_A`).

   ADDITIONAL INFORMATION: These accesses are typically performed in the context of an interrupt service routine. See the SEC chapter for further information. Do not attempt to read or write inactive data buffers. If the core attempts to access inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Initializing Multichannel Transfers

When in Multichannel (TDM) Mode or Packed I^{2}S Mode, the SPORT is in a multichannel operational mode. Follow the steps below to properly initialize the SPORT for multichannel modes of operation. More steps are required to properly initialize the SEC to service the SPORT interrupts (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate register changes, this example also applies to half SPORT B.

1. Clear the `SPORT_CTL_A` and `SPORT_MCTL_A` registers.

   ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multichannel logic) is fully reset before attempting to reprogram it.

2. Optionally program the `SPORT_DIV_A` clock divisor register.

   ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the Serial Clock and `SPORT_CTL2_A` sections.

3. Program the `SPORT_CS0_A` - `SPORT_CS3_A` channel select registers.

4. Program the `SPORT_MCTL_A` multichannel configuration register.

   ADDITIONAL INFORMATION: The SPORT supports many multichannel options. For more information, see the Multichannel (TDM) Mode section. Do not set the `SPORT_MCTL_A.MCE` enable bit in this step.

5. Program the `SPORT_CTL_A` primary configuration register.

   ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the Operating Modes and Options section for details). Do not set the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` buffer enable bits in this step.

6. Optionally program the `SPORT_CTL2_A` secondary configuration register.

   ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the Multiplexer Logic section for details).

7. Optionally program the `SPORT_ERR_A` error register.

ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt request is desired (see the Error Detection (Status) Interrupt section for details).

8. Set the `SPORT_MCTL_A.MCE` bit to enable multichannel mode.
9. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

   ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate. DMA mode is recommended for multichannel modes of operation. For more information, see the Using DMA for SPORT Transfers programming model.

Using DMA for SPORT Transfers

DMA is supported in all SPORT operating modes (Standard DSP Serial Mode, I^{2}S Mode, Left-Justified Mode, Right-Justified Mode, Multichannel (TDM) Mode or Packed I^{2}S Mode). To enable DMA operation with the SPORT, execute the steps described in this section after initializing and enabling the SPORT. Instead of using the single word read or write operations described in the referenced programming models, the DMA engine automates accesses to the enabled SPORT data buffers.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, it also applies to half SPORT B.

1. Follow the guidance in the multichannel (Initializing Multichannel Transfers) or non-multichannel (Initializing Core-Driven (Non-MCM) Transfers) programming models to properly initialize and enable the SPORT hardware.
2. Prepare the data buffers in memory.

   ADDITIONAL INFORMATION: Ensure that the DMA buffer is defined according to the DMA Transfers section. For the multichannel modes of operation, be sure to also consider the setting of the `SPORT_MCTL_A.MCPDE` bit, as described in the Multichannel DMA Data Packing (MCPDE) section.
3. Initialize and enable the DMA channel allocated for the SPORT, as described in the Direct Memory Access (DMA) chapter.

Using Companding as a Function

The data in the transmit and receive buffers are actually companded in place. As such, the following programming model can be used to exercise the companding hardware without transferring data, which is useful for test/debug purposes.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Configure the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN`=1) with both the primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI`=0 and `SPORT_CTL_A.SPENSEC`=0).

2. Enable the desired companding scheme in the `SPORT_CTL_A.DTYPE` field.
3. Write a 32-bit word to one of the transmit buffers.
4. Wait two system clock cycles.

   ADDITIONAL INFORMATION: This delay is required to allow the SPORT companding hardware to reload the transmit buffer with the companded result. Any instructions that do not access the transmit buffer can be used to cause this delay.
5. Read the 8-bit compressed value from the transmit buffer written above.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (`SPORT_CTL_A.SLEN`).

ADSP-SC58x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 34-14: ADSP-SC58x SPORT Register List

Name	Description
SPORT_CS0_A	Half SPORT 'A' Multichannel 0-31 Select Register
SPORT_CS0_B	Half SPORT 'B' Multichannel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multichannel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multichannel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multichannel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multichannel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register

Table 34-14: ADSP-SC58x SPORT Register List (Continued)

Name	Description
`SPORT_MSTAT_B`	Half SPORT 'B' Multichannel Status Register
`SPORT_RXPRI_A`	Half SPORT 'A' Rx Buffer (Primary) Register
`SPORT_RXPRI_B`	Half SPORT 'B' Rx Buffer (Primary) Register
`SPORT_RXSEC_A`	Half SPORT 'A' Rx Buffer (Secondary) Register
`SPORT_RXSEC_B`	Half SPORT 'B' Rx Buffer (Secondary) Register
`SPORT_TXPRI_A`	Half SPORT 'A' Tx Buffer (Primary) Register
`SPORT_TXPRI_B`	Half SPORT 'B' Tx Buffer (Primary) Register
`SPORT_TXSEC_A`	Half SPORT 'A' Tx Buffer (Secondary) Register
`SPORT_TXSEC_B`	Half SPORT 'B' Tx Buffer (Secondary) Register

Half SPORT 'A' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the `SPORT_CS0_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

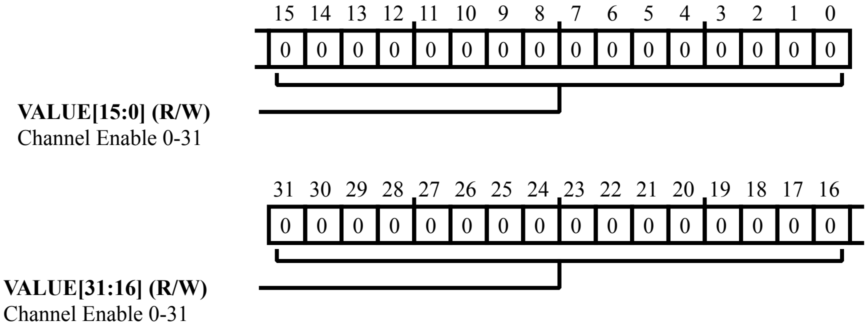

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 0-31
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 0-31

Figure 34-13: SPORT_CS0_A Register Diagram

Table 34-15: SPORT_CS0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'B' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the `SPORT_CS0_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

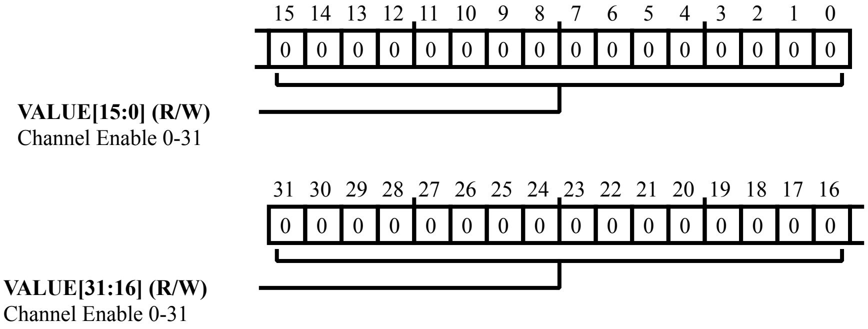

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 0-31
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 0-31

Figure 34-14: SPORT_CS0_B Register Diagram

Table 34-16: SPORT_CS0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'A' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the `SPORT_CS1_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

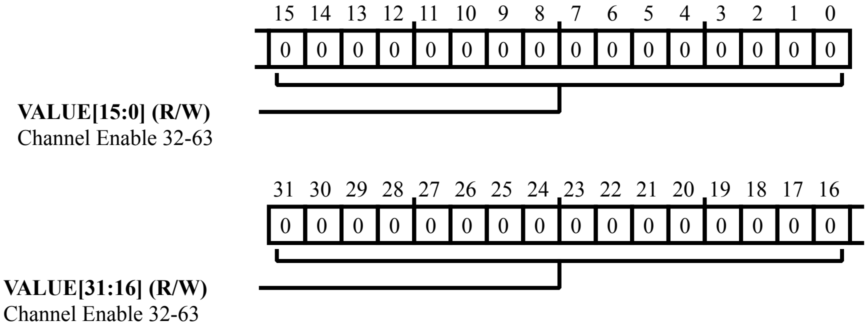

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 32-63
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 32-63

Figure 34-15: SPORT_CS1_A Register Diagram

Table 34-17: SPORT_CS1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'B' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the `SPORT_CS1_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

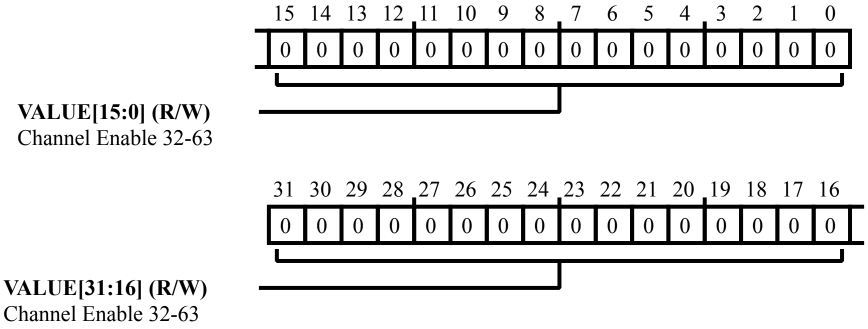

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 32-63
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 32-63

Figure 34-16: SPORT_CS1_B Register Diagram

Table 34-18: SPORT_CS1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'A' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the `SPORT_CS2_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

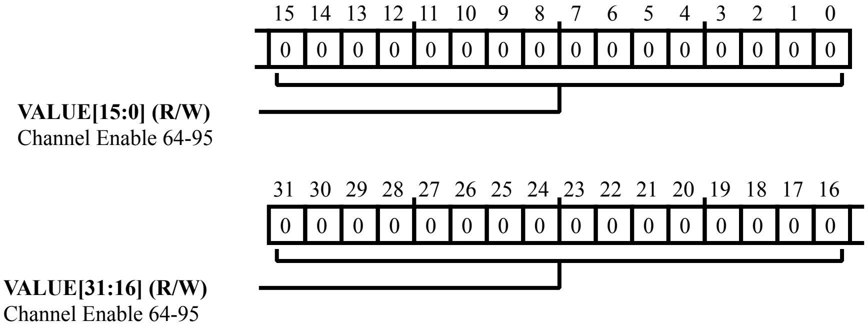

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 64-95
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 64-95

Figure 34-17: SPORT_CS2_A Register Diagram

Table 34-19: SPORT_CS2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'B' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the `SPORT_CS2_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

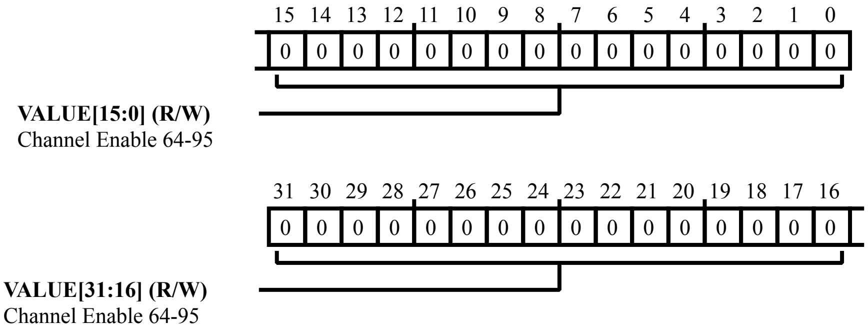

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 64-95
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 64-95

Figure 34-18: SPORT_CS2_B Register Diagram

Table 34-20: SPORT_CS2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'A' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the `SPORT_CS3_A` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

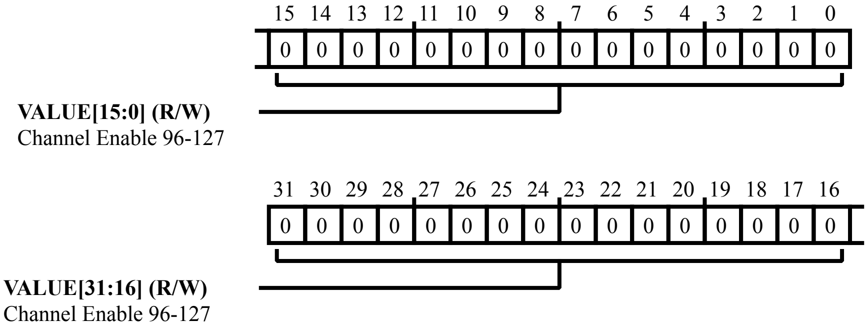

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 96-127
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 96-127

Figure 34-19: SPORT_CS3_A Register Diagram

Table 34-21: SPORT_CS3_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'B' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the `SPORT_CS3_B` register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

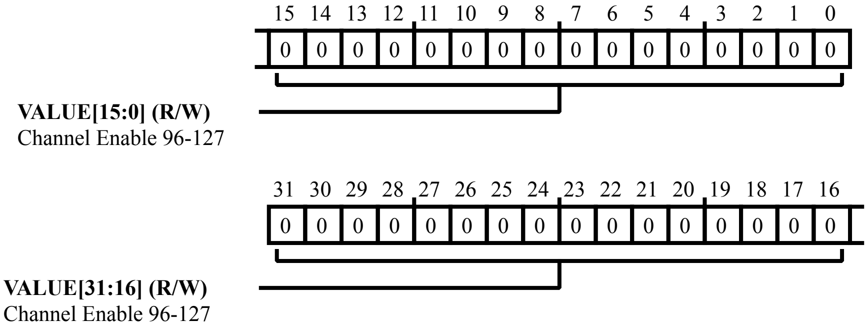

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Channel Enable 96-127
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Channel Enable 96-127

Figure 34-20: SPORT_CS3_B Register Diagram

Table 34-22: SPORT_CS3_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'A' Control 2 Register

The `SPORT_CTL2_A` register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

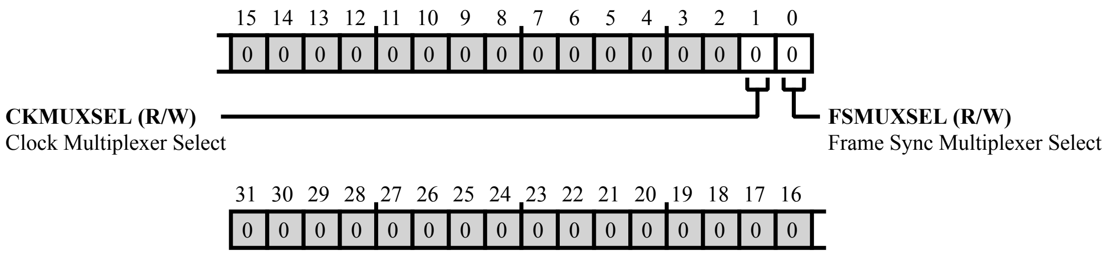

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CKMUXSEL (R/W)
Clock Multiplexer Select
FSMUXSEL (R/W)
Frame Sync Multiplexer Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-21: SPORT_CTL2_A Register Diagram

Table 34-23: SPORT_CTL2_A Register Fields

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">1
(R/W)</td><td rowspan="3">CKMUXSEL</td><td colspan="2">Clock Multiplexer Select.
The <code>SPORT_CTL2_A.CKMUXSEL</code> bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the <code>SPORT_CTL2_A.CKMUXSEL</code> bit is enabled, half SPORT 'A' uses <code>SPORT_BCLK</code> instead of <code>SPORT_ACLK</code>.</td></tr>
<tr><td>0</td><td>Disable serial clock multiplexing</td></tr>
<tr><td>1</td><td>Enable serial clock multiplexing</td></tr>
<tr><td rowspan="3">0
(R/W)</td><td rowspan="3">FSMUXSEL</td><td colspan="2">Frame Sync Multiplexer Select.
The <code>SPORT_CTL2_A.FSMUXSEL</code> bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the <code>SPORT_CTL2_A.FSMUXSEL</code> bit is enabled, half SPORT 'A' uses <code>SPORT_BFS</code> instead of <code>SPORT_AFS</code>.</td></tr>
<tr><td>0</td><td>Disable frame sync multiplexing</td></tr>
<tr><td>1</td><td>Enable frame sync multiplexing</td></tr>
</table>

Half SPORT 'B' Control 2 Register

The SPORT_CTL2_B register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

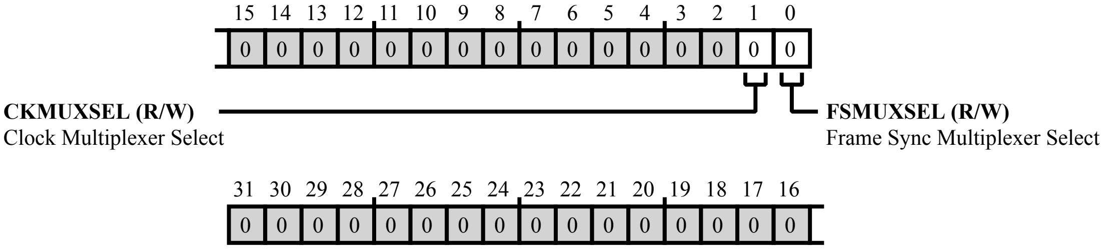

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CKMUXSEL (R/W)
Clock Multiplexer Select
FSMUXSEL (R/W)
Frame Sync Multiplexer Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-22: SPORT_CTL2_B Register Diagram

Table 34-24: SPORT_CTL2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_B.CKMUXSEL bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the SPORT_CTL2_B.CKMUXSEL bit is enabled, half SPORT 'B' uses SPORT_ACLK instead of SPORT_BCLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_B.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the SPORT_CTL2_B.FSMUXSEL bit is enabled, half SPORT 'B' uses SPORT_AFS instead of SPORT_BFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

Half SPORT 'A' Control Register

The `SPORT_CTL_A` register contains transmit and receive control bits for SPORT half 'A', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the `SPORT_CTL_A` register vary depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

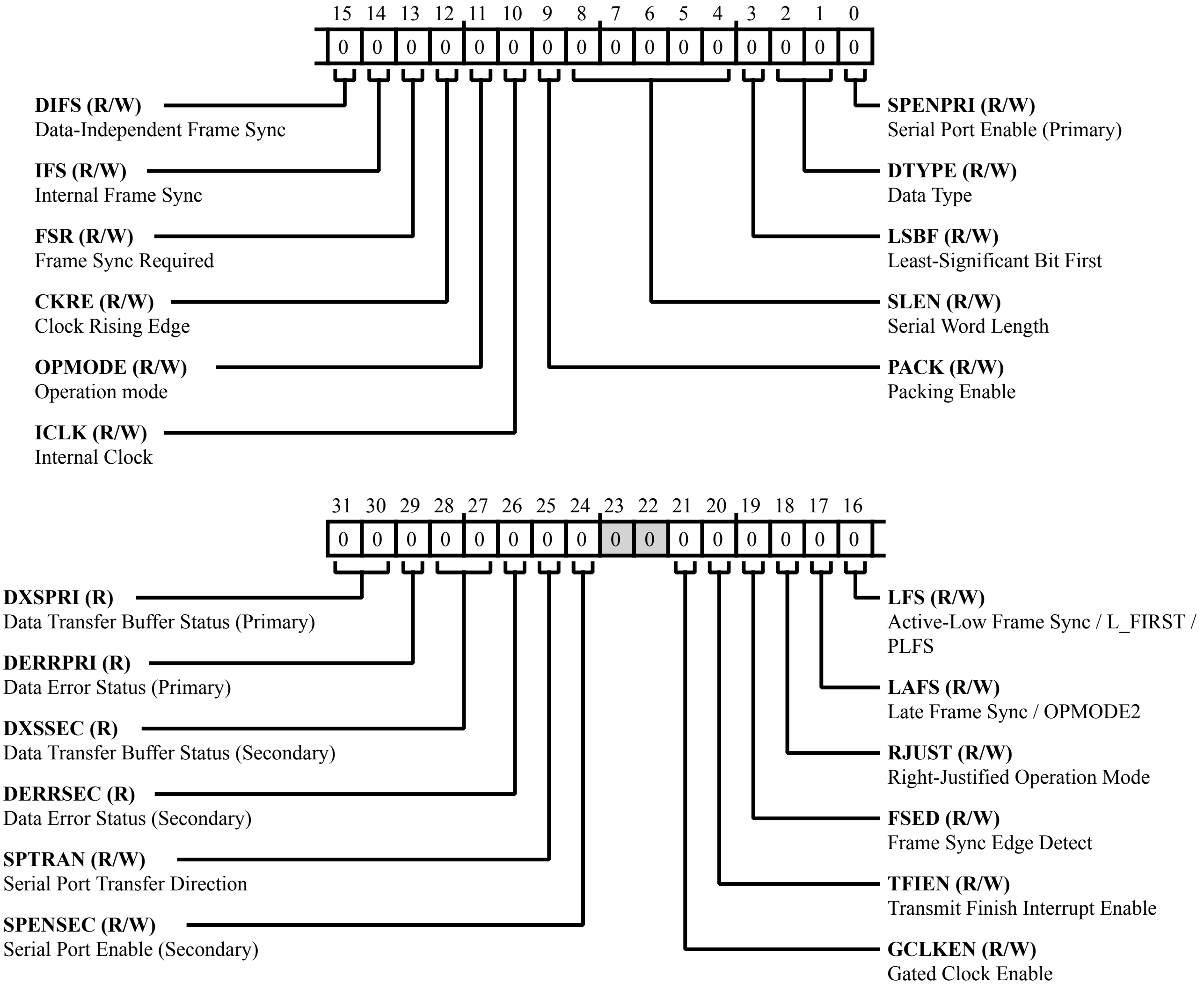

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIFS (R/W)
Data-Independent Frame Sync
IFS (R/W)
Internal Frame Sync
FSR (R/W)
Frame Sync Required
CKRE (R/W)
Clock Rising Edge
OPMODE (R/W)
Operation mode
ICLK (R/W)
Internal Clock
SPENPRI (R/W)
Serial Port Enable (Primary)
DTYPE (R/W)
Data Type
LSBF (R/W)
Least-Significant Bit First
SLEN (R/W)
Serial Word Length
PACK (R/W)
Packing Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DXSPRI (R)
Data Transfer Buffer Status (Primary)
DERRPRI (R)
Data Error Status (Primary)
DXSSEC (R)
Data Transfer Buffer Status (Secondary)
DERRSEC (R)
Data Error Status (Secondary)
SPTRAN (R/W)
Serial Port Transfer Direction
SPENSEC (R/W)
Serial Port Enable (Secondary)
LFS (R/W)
Active-Low Frame Sync / L_FIRST / PLFS
LAFS (R/W)
Late Frame Sync / OPMODE2
RJUST (R/W)
Right-Justified Operation Mode
FSED (R/W)
Frame Sync Edge Detect
TFIEN (R/W)
Transmit Finish Interrupt Enable
GCLKEN (R/W)
Gated Clock Enable

Figure 34-23: SPORT_CTL_A Register Diagram

Table 34-25: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The `SPORT_CTL_A.DXSPRI` bit field indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The `SPORT_CTL_A.DERRPRI` bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the `SPORT_CTL_A.FSR` bit =1, the `SPORT_CTL_A.DERRPRI` bit indicates whether the `SPORT_AFS` signal (from an internal or external source) occurred while the `SPORT_TXPRI_A` data buffer was empty (during transmit) or the `SPORT_RXPRI_A` data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the `SPORT_AFS` signal. It is important to note that, as a receiver, the `SPORT_CTL_A.DERRPRI` bit indicates when the channel has received new data while the `SPORT_RXPRI_A` receive buffer is full. This new data overwrites existing data. If the `SPORT_CTL_A.FSR` bit =0, the `SPORT_CTL_A.DERRPRI` bit is set whenever the SPORT is required to transmit while the `SPORT_TXPRI_A` transmit buffer is empty. It is also set whenever the SPORT is required to receive while the `SPORT_RXPRI_A` receive buffer is full. The SPORT clears the `SPORT_CTL_A.DERRPRI` bit if the `SPORT_ERR_A.DERRPSTAT` bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The `SPORT_CTL_A.DXSSEC` bit field indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 34-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The `SPORT_CTL_A.DERRSEC` bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the `SPORT_CTL_A.FSR` bit =1, the `SPORT_CTL_A.DERRSEC` bit indicates whether the `SPORT_AFS` signal (from an internal or external source) occurred while the `SPORT_TXSEC_A` data buffer was empty (during transmit) or the `SPORT_RXSEC_A` data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the `SPORT_AFS` signal. It is important to note that, as a receiver, the `SPORT_CTL_A.DERRSEC` bit indicates when the channel has received new data while the `SPORT_RXSEC_A` receive buffer is full. This new data overwrites existing data. If the `SPORT_CTL_A.FSR` bit =0, the `SPORT_CTL_A.DERRSEC` bit is set whenever the SPORT is required to transmit while the `SPORT_TXSEC_A` transmit buffer is empty. It is also set whenever the SPORT is required to receive while the `SPORT_RXSEC_A` receive buffer is full. The SPORT clears the `SPORT_CTL_A.DERRSEC` bit if the `SPORT_ERR_A.DERRSSTAT` bit is cleared. 0 — No error 1 — Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The `SPORT_CTL_A.SPTRAN` bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the `SPORT_ACLK` and `SPORT_AFS` pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the `SPORT_ACLK` and `SPORT_AFS` pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit. 0 — Receive 1 — Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The `SPORT_CTL_A.SPENSEC` bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers. 0 — Disable 1 — Enable

Table 34-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The `SPORT_CTL_A.GCLKEN` bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (`SPORT_CTL_A.OPMODE` = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (`SPORT_CTL_A.RJUST` =1) or multichannel mode (`SPORT_MCTL_A.MCE` =1). When the `SPORT_CTL_A.GCLKEN` bit is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The `SPORT_CTL_A.TFIEN` bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the `DMA_CFG.INT` configuration. When enabled (`SPORT_CTL_A.TFIEN` =1), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (`SPORT_CTL_A.TFIEN` =0), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The `SPORT_CTL_A.FSED` bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The `SPORT_CTL_A.FSED` bit may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (`SPORT_CTL_A.FSED` =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 34-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The `SPORT_CTL_A.RJUST` bit enables the half SPORT (if `SPORT_CTL_A.OPMODE` =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the `SPORT_MCTL_A.WOFFSET` field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (`SPORT_CTL_A.OPMODE` =0) or in right-justified mode (`SPORT_CTL_A.RJUST` =1), the `SPORT_CTL_A.LAFS` bit selects whether the half SPORT generates a late frame sync (`SPORT_AFS` during first data bit) or generates an early frame sync signal (`SPORT_AFS` during serial clock cycle before first data bit). When the half SPORT is in I^{2}S / left-justified mode (`SPORT_CTL_A.OPMODE` =1), the `SPORT_CTL_A.LAFS` bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I^{2}S mode. When the half SPORT is in multichannel mode (`SPORT_MCTL_A.MCE` =1), the `SPORT_CTL_A.LAFS` bit is reserved.	
		0	Early frame sync (or I^{2}S mode)
		1	Late frame sync (or left-justified mode)
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (`SPORT_CTL_A.OPMODE` =0), the `SPORT_CTL_A.LFS` bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I^{2}S / packed / left-justified mode (`SPORT_CTL_A.OPMODE` =1), the `SPORT_CTL_A.LFS` bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I^{2}S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I^{2}S/packed mode) or right channel first (left-justified mode)

Table 34-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	DIFS	Data-Independent Frame Sync. The `SPORT_CTL_A.DIFS` bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by the `SPORT_DIV_A.FSDIV` bit. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the `SPORT_CTL_A.DIFS` bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The `SPORT_CTL_A.IFS` bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync
13 (R/W)	FSR	Frame Sync Required. The `SPORT_CTL_A.FSR` bit selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I^{2}S / packed / left-justified mode (`SPORT_CTL_A.OPMODE` =1) or is in multichannel mode (`SPORT_MCTL_A.MCE` =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The `SPORT_CTL_A.CKRE` bit selects the rising or falling edge of the `SPORT_ACLK` clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the `SPORT_ACLK`. Also, note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for the `SPORT_CTL_A.CKRE` bit. This programming drives the internally-generated signals on one edge of `SPORT_ACLK` and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge

Table 34-25: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	OPMODE	Operation mode. The `SPORT_CTL_A.OPMODE` bit selects whether the half SPORT operates in DSP standard/multichannel mode or operates in I^{2}S/packed/left-justified mode. The mode selection affects the operation of the `SPORT_CTL_A.LAFS` and `SPORT_CTL_A.LFS` bits. Also, the `SPORT_CTL_A.OPMODE` bit enables or disables operation of the `SPORT_CTL_A.GCLKEN`, `SPORT_CTL_A.FSED`, `SPORT_CTL_A.RJUST`, `SPORT_CTL_A.DIFS`, `SPORT_CTL_A.FSR`, and `SPORT_CTL_A.CKRE` bits.	
		0	DSP standard/multichannel mode
		1	I^{2}S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (`SPORT_CTL_A.OPMODE` =0), the `SPORT_CTL_A.ICLK` bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the `SPORT_ACLK` clock signal, and `SPORT_ACLK` is an output. The `SPORT_DIV_A.CLKDIV` serial clock divisor value determines the clock frequency. For internal clock disabled, the `SPORT_ACLK` clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock
9 (R/W)	PACK	Packing Enable. The `SPORT_CTL_A.PACK` bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable

Table 34-25: SPORT_CTL_A Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td>8:4
(R/W)</td><td>SLEN</td><td colspan="2">Serial Word Length.
The `SPORT_CTL_A.SLEN` bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is:
`SPORT_CTL_A.SLEN` = (serial word length in bits) - 1
For DSP standard mode (`SPORT_CTL_A.OPMODE` =0), use `SPORT_CTL_A.SLEN` of 3 to 31 bits.
For I^2S / packed / left-justified mode (`SPORT_CTL_A.OPMODE` =1), use `SPORT_CTL_A.SLEN` of 4 to 31 bits.</td></tr>
<tr><td rowspan="3">3
(R/W)</td><td rowspan="3">LSBF</td><td colspan="2">Least-Significant Bit First.
The `SPORT_CTL_A.LSBF` bit selects whether the half SPORT transmits or receives data LSB first or MSB first.</td></tr>
<tr><td>0</td><td>MSB first sent/received (big endian)</td></tr>
<tr><td>1</td><td>LSB first sent/received (little endian)</td></tr>
<tr><td rowspan="5">2:1
(R/W)</td><td rowspan="5">DTYPE</td><td colspan="2">Data Type.
The `SPORT_CTL_A.DTYPE` bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (`SPORT_CTL_A.OPMODE` =0).</td></tr>
<tr><td>0</td><td>Right-justify data, zero-fill unused MSBs</td></tr>
<tr><td>1</td><td>Right-justify data, sign-extend unused MSBs</td></tr>
<tr><td>2</td><td>u-law compand data</td></tr>
<tr><td>3</td><td>A-law compand data</td></tr>
<tr><td rowspan="3">0
(R/W)</td><td rowspan="3">SPENPRI</td><td colspan="2">Serial Port Enable (Primary).
The `SPORT_CTL_A.SPENPRI` bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.</td></tr>
<tr><td>0</td><td>Disable</td></tr>
<tr><td>1</td><td>Enable</td></tr>
</table>

Half SPORT 'B' Control Register

The `SPORT_CTL_B` register contains transmit and receive control bits for SPORT half 'B', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the `SPORT_CTL_B` register vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

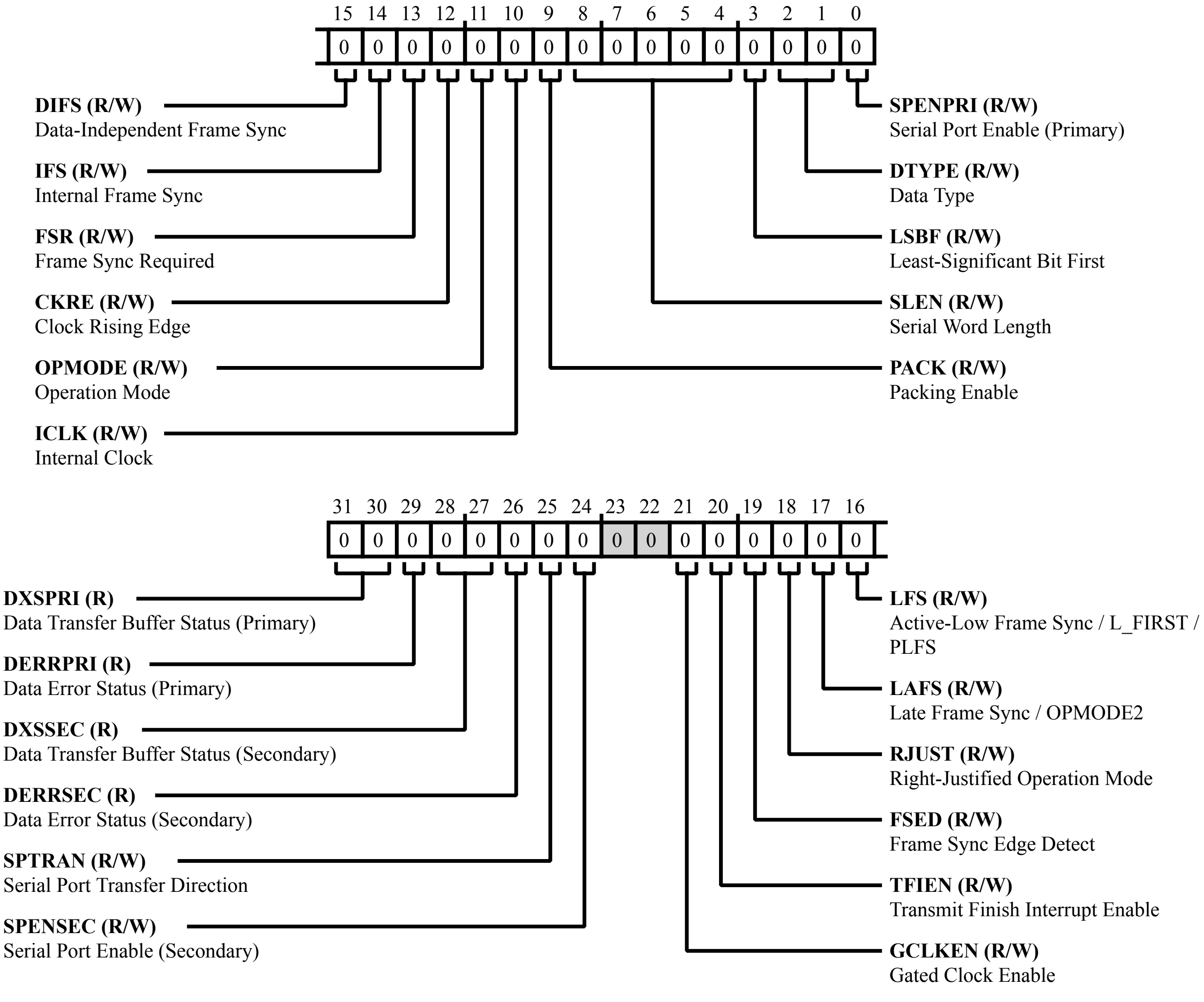

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIFS (R/W)
Data-Independent Frame Sync
IFS (R/W)
Internal Frame Sync
FSR (R/W)
Frame Sync Required
CKRE (R/W)
Clock Rising Edge
OPMODE (R/W)
Operation Mode
ICLK (R/W)
Internal Clock
SPENPRI (R/W)
Serial Port Enable (Primary)
DTYPE (R/W)
Data Type
LSBF (R/W)
Least-Significant Bit First
SLEN (R/W)
Serial Word Length
PACK (R/W)
Packing Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DXSPRI (R)
Data Transfer Buffer Status (Primary)
DERRPRI (R)
Data Error Status (Primary)
DXSSEC (R)
Data Transfer Buffer Status (Secondary)
DERRSEC (R)
Data Error Status (Secondary)
SPTRAN (R/W)
Serial Port Transfer Direction
SPENSEC (R/W)
Serial Port Enable (Secondary)
LFS (R/W)
Active-Low Frame Sync / L_FIRST / PLFS
LAFS (R/W)
Late Frame Sync / OPMODE2
RJUST (R/W)
Right-Justified Operation Mode
FSED (R/W)
Frame Sync Edge Detect
TFIEN (R/W)
Transmit Finish Interrupt Enable
GCLKEN (R/W)
Gated Clock Enable

Figure 34-24: SPORT_CTL_B Register Diagram

Table 34-26: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The `SPORT_CTL_B.DXSPRI` bit field indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The `SPORT_CTL_B.DERRPRI` bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the `SPORT_CTL_B.FSR` bit =1, the `SPORT_CTL_B.DERRPRI` bit indicates whether the `SPORT_BFS` signal (from an internal or external source) occurred while the `SPORT_TXPRI_B` data buffer was empty (during transmit) or the `SPORT_RXPRI_B` data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the `SPORT_BFS` signal. It is important to note that, as a receiver, the `SPORT_CTL_B.DERRPRI` bit indicates when the channel has received new data while the `SPORT_RXPRI_B` receive buffer is full. This new data overwrites existing data. If the `SPORT_CTL_B.FSR` bit =0, the `SPORT_CTL_B.DERRPRI` bit is set whenever the SPORT is required to transmit while the `SPORT_TXPRI_B` transmit buffer is empty and is set whenever the SPORT is required to receive while the `SPORT_RXPRI_B` receive buffer is full. The SPORT clears the `SPORT_CTL_B.DERRPRI` bit if the `SPORT_ERR_B.DERRPSTAT` bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The `SPORT_CTL_B.DXSSEC` bit field indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The `SPORT_CTL_B.DERRSEC` bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the `SPORT_CTL_B.FSR` bit =1, the `SPORT_CTL_B.DERRSEC` bit indicates whether the `SPORT_BFS` signal (from an internal or external source) occurred while the `SPORT_TXSEC_B` data buffer was empty (during transmit) or the `SPORT_RXSEC_B` data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the `SPORT_BFS` signal. It is important to note that, as a receiver, the `SPORT_CTL_B.DERRSEC` bit indicates when the channel has received new data while the `SPORT_RXSEC_B` receive buffer is full. This new data overwrites existing data. If the `SPORT_CTL_B.FSR` bit =0, the `SPORT_CTL_B.DERRSEC` bit is set whenever the SPORT is required to transmit while the `SPORT_TXSEC_B` transmit buffer is empty. It is also set whenever the SPORT is required to receive while the `SPORT_RXSEC_B` receive buffer is full. The SPORT clears the `SPORT_CTL_B.DERRSEC` bit if the `SPORT_ERR_B.DERRSSTAT` bit is cleared. 0 No error 1 Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The `SPORT_CTL_B.SPTRAN` bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the `SPORT_BCLK` and `SPORT_BFS` pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the `SPORT_BCLK` and `SPORT_BFS` pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit. 0 Receive 1 Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The `SPORT_CTL_B.SPENSEC` bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers. 0 Disable 1 Enable

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The `SPORT_CTL_B.GCLKEN` bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (`SPORT_CTL_B.OPMODE` = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (`SPORT_CTL_B.RJUST` =1) or multichannel mode (`SPORT_MCTL_B.MCE` =1). When `SPORT_CTL_B.GCLKEN` is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The `SPORT_CTL_B.TFIEN` bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the `DMA_CFG.INT` configuration. When enabled (`SPORT_CTL_B.TFIEN` =1), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (`SPORT_CTL_B.TFIEN` =0), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The `SPORT_CTL_B.FSED` bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The `SPORT_CTL_B.FSED` may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (`SPORT_CTL_B.FSED` =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The `SPORT_CTL_B.RJUST` bit enables the half SPORT (if `SPORT_CTL_B.OPMODE` =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the `SPORT_MCTL_B.WOFFSET` field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (`SPORT_CTL_B.OPMODE` =0) or in right-justified mode (`SPORT_CTL_B.RJUST` =1), the `SPORT_CTL_B.LAFS` bit selects whether the half SPORT generates a late frame sync (`SPORT_BFS` during first data bit) or generates an early frame sync signal (`SPORT_BFS` during serial clock cycle before first data bit). When the half SPORT is in I^{2}S / left-justified mode (`SPORT_CTL_B.OPMODE` =1), the `SPORT_CTL_B.LAFS` bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I^{2}S mode. When the half SPORT is in multichannel mode (`SPORT_MCTL_B.MCE` =1), the `SPORT_CTL_B.LAFS` bit is reserved.	
		0	Early frame sync (or I^{2}S mode)
		1	Late frame sync (or left-justified mode)

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (`SPORT_CTL_B.OPMODE` =0), the `SPORT_CTL_B.LFS` bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I^{2}S / packed / left-justified mode (`SPORT_CTL_B.OPMODE` =1), the `SPORT_CTL_B.LFS` bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I^{2}S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I^{2}S/packed mode) or right channel first (left-justified mode)
15 (R/W)	DIFS	Data-Independent Frame Sync. The `SPORT_CTL_B.DIFS` bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by `SPORT_DIV_B.FSDIV`. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the `SPORT_CTL_B.DIFS` bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The `SPORT_CTL_B.IFS` bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	FSR	Frame Sync Required. The `SPORT_CTL_B.FSR` selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I^{2}S / packed / left-justified mode (`SPORT_CTL_B.OPMODE` =1) or is in multichannel mode (`SPORT_MCTL_B.MCE` =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The `SPORT_CTL_B.CKRE` selects the rising or falling edge of the `SPORT_BCLK` clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the `SPORT_BCLK`. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for `SPORT_CTL_B.CKRE`. This programming drives the internally-generated signals on one edge of `SPORT_BCLK` and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge
11 (R/W)	OPMODE	Operation Mode. The `SPORT_CTL_B.OPMODE` bit selects whether the half SPORT operates in DSP standard / multichannel mode or operates in I^{2}S / packed / left-justified mode. The mode selection affects the operation of the `SPORT_CTL_B.LAFS` and `SPORT_CTL_B.LFS` bits. Also, the `SPORT_CTL_B.OPMODE` bit enables or disables operation of the `SPORT_CTL_B.GCLKEN`, `SPORT_CTL_B.FSED`, `SPORT_CTL_B.RJUST`, `SPORT_CTL_B.DIFS`, `SPORT_CTL_B.FSR`, and `SPORT_CTL_B.CKRE` bits.	
		0	DSP standard/multichannel mode
		1	I^{2}S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (`SPORT_CTL_B.OPMODE` =0), the `SPORT_CTL_B.ICLK` bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the `SPORT_BCLK` clock signal, and the `SPORT_BCLK` is an output. The `SPORT_DIV_B.CLKDIV` serial clock divisor value determines the clock frequency. For internal clock disabled, the `SPORT_BCLK` clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	PACK	Packing Enable. The `SPORT_CTL_B.PACK` bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable
8:4 (R/W)	SLEN	Serial Word Length. The `SPORT_CTL_B.SLEN` bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: `SPORT_CTL_B.SLEN` = (serial word length in bits) - 1 For DSP standard mode (`SPORT_CTL_B.OPMODE` =0), use `SPORT_CTL_B.SLEN` of 3 to 31 bits. For I^{2}S / packed / left-justified mode (`SPORT_CTL_B.OPMODE` =1), use `SPORT_CTL_B.SLEN` of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The `SPORT_CTL_B.LSBF` bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received (big endian)
		1	LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The `SPORT_CTL_B.DTYPE` bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (`SPORT_CTL_B.OPMODE` =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	u-law compand data
		3	A-law compand data

Table 34-26: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The `SPORT_CTL_B.SPENPRI` bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'A' Divisor Register

The SPORT_DIV_A register contains divisor values that determine frequencies of internally-generated clocks and frame syncs for half SPORT 'A'.

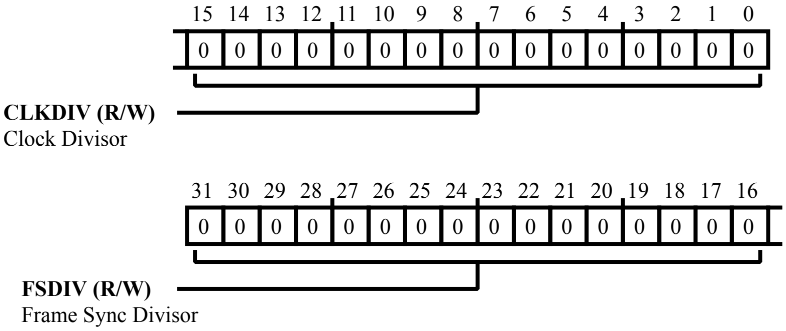


Figure 34-25: SPORT_DIV_A Register Diagram

Table 34-27: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The `SPORT_DIV_A.FSDIV` bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating `SPORT_DIV_A.FSDIV` to the number of cycles between frame sync pulses is: `SPORT_DIV_A.FSDIV` = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of `SPORT_DIV_A.FSDIV`, given the serial clock frequency and desired frame sync frequency: FSDIV = (`SPORT_ACLK` / `SPORT_AFS`) - 1 Note that the frame sync is continuously active when `SPORT_DIV_A.FSDIV` = 0. The value of `SPORT_DIV_A.FSDIV` should not be less than the serial word length (`SPORT_CTL_A.SLEN`), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The `SPORT_DIV_A.CLKDIV` bits select the divisor that the half SPORT uses to calculate the serial clock (`SPORT_ACLK`) from the processor system clock (SCLK0_0). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (`SPORT_CTL_A.ICLK` =1), legal `SPORT_DIV_A.CLKDIV` values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of `SPORT_DIV_A.CLKDIV`: CLKDIV = (SCLK0_0 / `SPORT_ACLK`) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'B' Divisor Register

The `SPORT_DIV_B` contains divisor values that determine frequencies of internally-generated clocks and frame syncs for SPORT half 'B'.

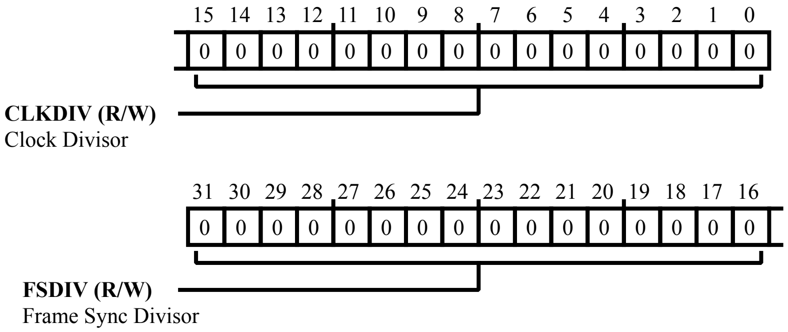


Figure 34-26: SPORT_DIV_B Register Diagram

Table 34-28: SPORT_DIV_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The `SPORT_DIV_B.FSDIV` bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating `SPORT_DIV_B.FSDIV` to the number of cycles between frame sync pulses is: `SPORT_DIV_B.FSDIV` = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of `SPORT_DIV_B.FSDIV`, given the serial clock frequency and desired frame sync frequency: FSDIV = (`SPORT_BCLK` / `SPORT_BFS`) - 1 Note that the frame sync is continuously active when `SPORT_DIV_B.FSDIV` = 0. The value of `SPORT_DIV_B.FSDIV` should not be less than the serial word length (`SPORT_CTL_B.SLEN`), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The `SPORT_DIV_B.CLKDIV` bits select the divisor that the half SPORT uses to calculate the serial clock (`SPORT_BCLK`) from the processor system clock (SCLK0_0). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (`SPORT_CTL_B.ICLK` =1), legal `SPORT_DIV_B.CLKDIV` values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of `SPORT_DIV_B.CLKDIV`: CLKDIV = (SCLK0_0 `SPORT_BCLK`) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'A' Error Register

The SPORT_ERR_A register contains error status and error interrupt mask bits for SPORT half 'A', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

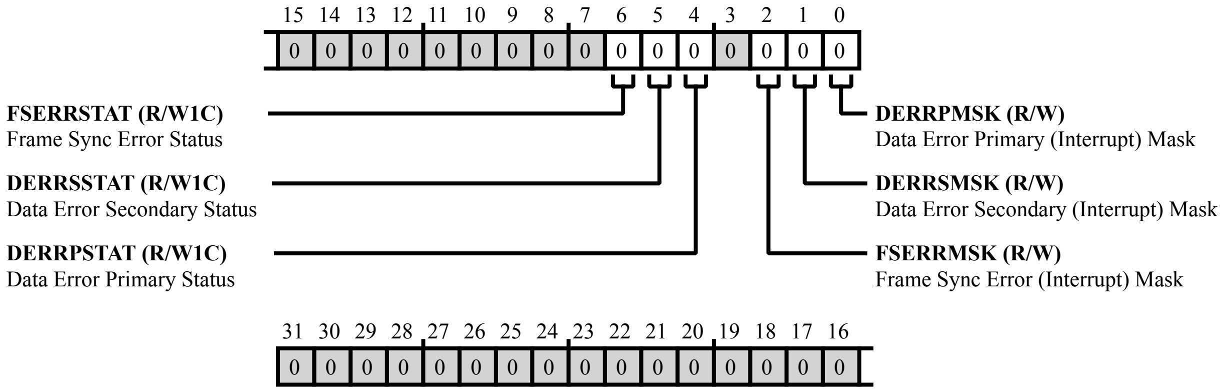

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSERRSTAT (R/W1C)
Frame Sync Error Status
DERRSSTAT (R/W1C)
Data Error Secondary Status
DERRPSTAT (R/W1C)
Data Error Primary Status
DERRPMSK (R/W)
Data Error Primary (Interrupt) Mask
DERRSMSK (R/W)
Data Error Secondary (Interrupt) Mask
FSERRMSK (R/W)
Frame Sync Error (Interrupt) Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-27: SPORT_ERR_A Register Diagram

Table 34-29: SPORT_ERR_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_A.FSERRSTAT bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, SPORT_CTL_A.SLEN = 31). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 34-29: SPORT_ERR_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The `SPORT_ERR_A.DERRSSTAT` bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (`SPORT_CTL_A.SPTRAN` =1), the `SPORT_ERR_A.DERRSSTAT` bit indicates the transmit underflow status. During receive (`SPORT_CTL_A.SPTRAN` =0), the `SPORT_ERR_A.DERRSSTAT` bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only `SPORT_CTL_A.DERRSEC` status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The `SPORT_ERR_A.DERRPSTAT` bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (`SPORT_CTL_A.SPTRAN` =1), the `SPORT_ERR_A.DERRPSTAT` bit indicates the transmit underflow status. During receive (`SPORT_CTL_A.SPTRAN` =0), the `SPORT_ERR_A.DERRPSTAT` bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only `SPORT_CTL_A.DERRPRI` status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The `SPORT_ERR_A.FSERRMSK` unmasks (enables) the half SPORT to generate the frame sync error interrupt request.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The `SPORT_ERR_A.DERRSMSK` unmasks (enables) the half SPORT to generate the data error interrupt request for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The `SPORT_ERR_A.DERRPMSK` unmasks (enables) the half SPORT to generate the data error interrupt request for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'B' Error Register

The `SPORT_ERR_B` register contains error status and error interrupt mask bits for SPORT half 'B', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

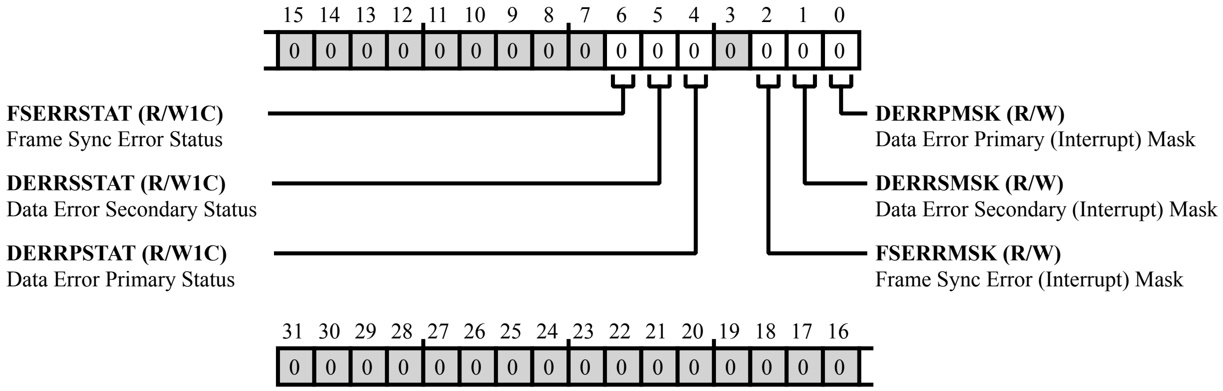

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSERRSTAT (R/W1C)
Frame Sync Error Status
DERRSSTAT (R/W1C)
Data Error Secondary Status
DERRPSTAT (R/W1C)
Data Error Primary Status
DERRPMSK (R/W)
Data Error Primary (Interrupt) Mask
DERRSMSK (R/W)
Data Error Secondary (Interrupt) Mask
FSERRMSK (R/W)
Frame Sync Error (Interrupt) Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-28: SPORT_ERR_B Register Diagram

Table 34-30: SPORT_ERR_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	FSERRSTAT	Frame Sync Error Status. The `SPORT_ERR_B.FSERRSTAT` bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, `SPORT_CTL_B.SLEN` = 31). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 34-30: SPORT_ERR_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The `SPORT_ERR_B.DERRSSTAT` bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (`SPORT_CTL_B.SPTRAN` =1), `SPORT_ERR_B.DERRSSTAT` indicates the transmit underflow status. During receive (`SPORT_CTL_B.SPTRAN` =0), `SPORT_ERR_B.DERRSSTAT` indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only `SPORT_CTL_B.DERRSEC` status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The `SPORT_ERR_B.DERRPSTAT` bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (`SPORT_CTL_B.SPTRAN` =1), the `SPORT_ERR_B.DERRPSTAT` bit indicates the transmit underflow status. During receive (`SPORT_CTL_B.SPTRAN` =0), the `SPORT_ERR_B.DERRPSTAT` bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only `SPORT_CTL_B.DERRPRI` status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The `SPORT_ERR_B.FSERRMSK` unmasks (enables) the half SPORT to generate the frame sync error interrupt request.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The `SPORT_ERR_B.DERRSMSK` unmasks (enables) the half SPORT to generate the data error interrupt request for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The `SPORT_ERR_B.DERRPMSK` unmasks (enables) the half SPORT to generate the data error interrupt request for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'A' Multichannel Control Register

The `SPORT_MCTL_A` register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

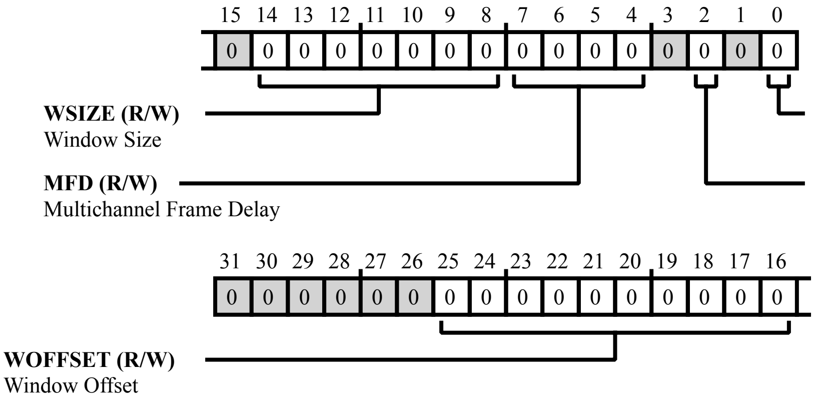

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WSIZE (R/W)
Window Size
MFD (R/W)
Multichannel Frame Delay
MCE (R/W)
Multichannel enable
MCPDE (R/W)
Multichannel Packing DMA Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WOFFSET (R/W)
Window Offset

Figure 34-29: SPORT_MCTL_A Register Diagram

Table 34-31: SPORT_MCTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The `SPORT_MCTL_A.WOFFSET` bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (`SPORT_MCTL_A.MCE` =0) and the right-justified mode is enabled (`SPORT_CTL_A.RJUST` =1), the least significant 6 bits of `SPORT_MCTL_A.WOFFSET` serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The `SPORT_MCTL_A.WSIZE` bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: `SPORT_MCTL_A.WSIZE` = (number of channel slots) -1
7:4 (R/W)	MFD	Multichannel Frame Delay. The `SPORT_MCTL_A.MFD` bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting multichannel frame delay of 0-15 serial clocks.

Table 34-31: SPORT_MCTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The `SPORT_MCTL_A.MCPDE` bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel enable. The `SPORT_MCTL_A.MCE` bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if `SPORT_CTL_A.OPMODE`=0; while it is configured in packed mode if `SPORT_CTL_A.OPMODE`=1. When configuring in these modes, the multichannel enable bit (`SPORT_MCTL_A.MCE`) should be set before enabling the SPORT data channel enable bits (`SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC`). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'B' Multichannel Control Register

The `SPORT_MCTL_B` register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

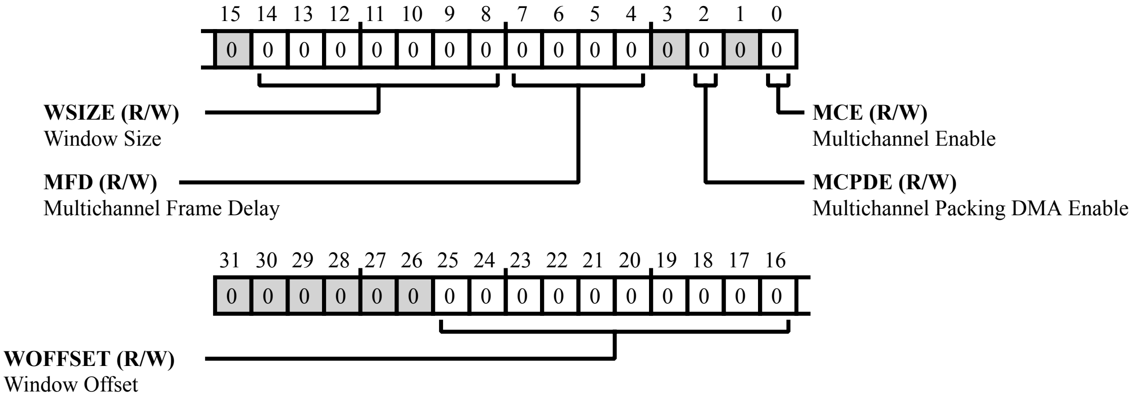

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WSIZE (R/W)
Window Size
MFD (R/W)
Multichannel Frame Delay
MCE (R/W)
Multichannel Enable
MCPDE (R/W)
Multichannel Packing DMA Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WOFFSET (R/W)
Window Offset

Figure 34-30: SPORT_MCTL_B Register Diagram

Table 34-32: SPORT_MCTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The `SPORT_MCTL_B.WOFFSET` bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (`SPORT_MCTL_B.MCE` =0) and right-justified mode is enabled (`SPORT_CTL_B.RJUST` =1), the least significant 6 bits of `SPORT_MCTL_B.WOFFSET` serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The `SPORT_MCTL_B.WSIZE` bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: `SPORT_MCTL_B.WSIZE` = (number of channel slots) -1
7:4 (R/W)	MFD	Multichannel Frame Delay. The `SPORT_MCTL_B.MFD` bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting a multichannel frame delay of 0-15 serial clocks.

Table 34-32: SPORT_MCTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The `SPORT_MCTL_B.MCPDE` bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel Enable. The `SPORT_MCTL_B.MCE` bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if `SPORT_CTL_B.OPMODE`=0; while it is configured in packed mode if `SPORT_CTL_B.OPMODE`=1. When configuring in these modes, the multichannel enable bit (`SPORT_MCTL_B.MCE`) should be set before enabling SPORT data channel enable bits (`SPORT_CTL_B.SPENPRI` and/or `SPORT_CTL_B.SPENSEC`). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the `SPORT_CTL_B.DERRPRI` and `SPORT_CTL_B.DERRSEC` bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'A' Multichannel Status Register

The `SPORT_MSTAT_A` register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the `SPORT_MSTAT_A` register restarts at 0 at each frame sync.

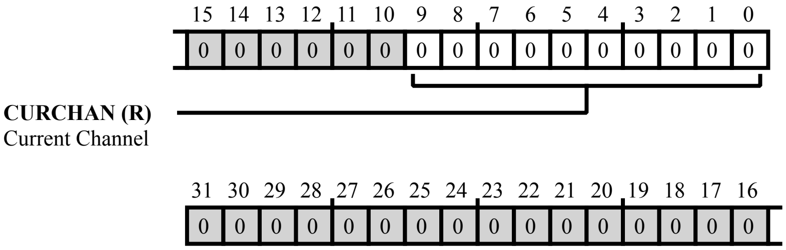

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CURCHAN (R)
Current Channel
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-31: SPORT_MSTAT_A Register Diagram

Table 34-33: SPORT_MSTAT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The `SPORT_MSTAT_A.CURCHAN` bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'B' Multichannel Status Register

The SPORT_MSTAT_B register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the SPORT_MSTAT_B register restarts at 0 at each frame sync.

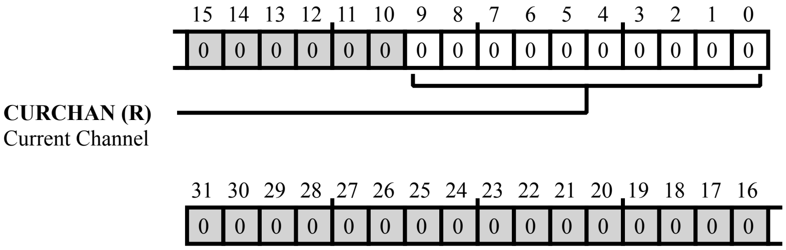

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CURCHAN (R)
Current Channel
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-32: SPORT_MSTAT_B Register Diagram

Table 34-34: SPORT_MSTAT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_B.CURCHAN bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'A' Rx Buffer (Primary) Register

The `SPORT_RXPRI_A` register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXPRI_A` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXPRI_A` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

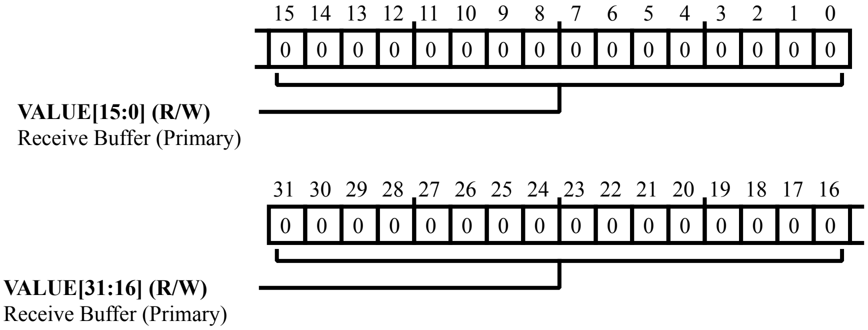

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Receive Buffer (Primary)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Receive Buffer (Primary)

Figure 34-33: SPORT_RXPRI_A Register Diagram

Table 34-35: SPORT_RXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The `SPORT_RXPRI_A.VALUE` bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_A.MCE`) empty the contents of this data buffer. For more information, see the `SPORT_CTL_A` and `SPORT_MCTL_A` register descriptions.

Half SPORT 'B' Rx Buffer (Primary) Register

The SPORT_RXPRI_B register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXPRI_B register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXPRI_B register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

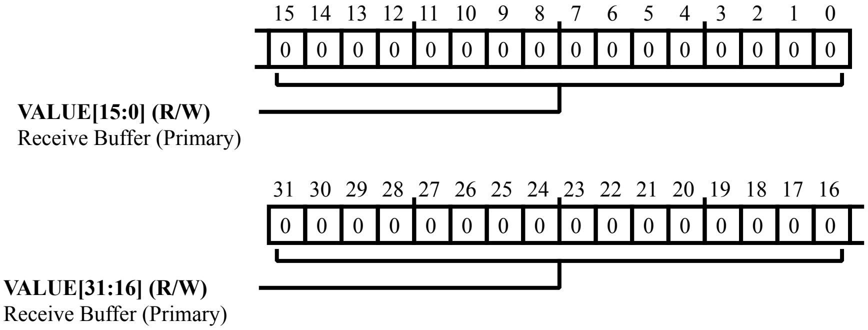

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Receive Buffer (Primary)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Receive Buffer (Primary)

Figure 34-34: SPORT_RXPRI_B Register Diagram

Table 34-36: SPORT_RXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_B.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Rx Buffer (Secondary) Register

The SPORT_RXSEC_A register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXSEC_A register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXSEC_A register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

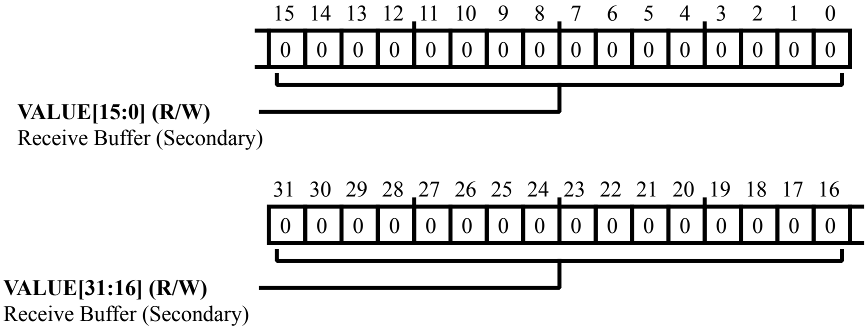

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Receive Buffer (Secondary)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Receive Buffer (Secondary)

Figure 34-35: SPORT_RXSEC_A Register Diagram

Table 34-37: SPORT_RXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_A.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Rx Buffer (Secondary) Register

The `SPORT_RXSEC_B` register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the `SPORT_RXSEC_B` register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the `SPORT_RXSEC_B` register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the `SPORT_CTL_A.PACK` bit setting).

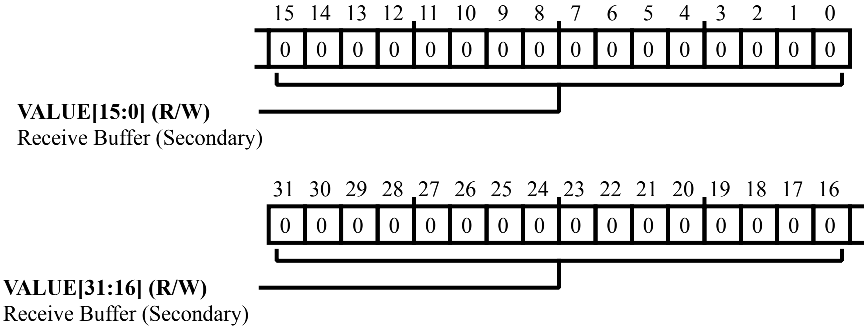

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Receive Buffer (Secondary)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Receive Buffer (Secondary)

Figure 34-36: SPORT_RXSEC_B Register Diagram

Table 34-38: SPORT_RXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The `SPORT_RXSEC_B.VALUE` bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_B.MCE`) empty the contents of this data buffer. For more information, see the `SPORT_CTL_B` and `SPORT_MCTL_B` register descriptions.

Half SPORT 'A' Tx Buffer (Primary) Register

The `SPORT_TXPRI_A` register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXPRI_A` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_A.PACK` =0); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_A.PACK` =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXPRI_A` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

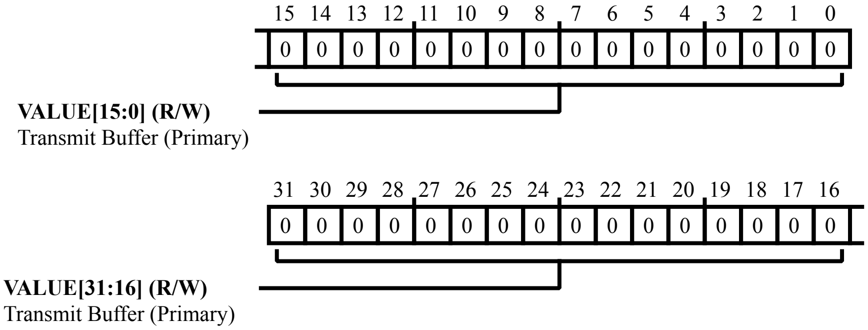


Figure 34-37: SPORT_TXPRI_A Register Diagram

Table 34-39: SPORT_TXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The `SPORT_TXPRI_A.VALUE` bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_A.MCE`) empty the contents of this data buffer. For more information, see the `SPORT_CTL_A` and `SPORT_MCTL_A` register descriptions.

Half SPORT 'B' Tx Buffer (Primary) Register

The `SPORT_TXPRI_B` register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXPRI_B` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_B.PACK` =0); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_B.PACK` =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXPRI_B` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

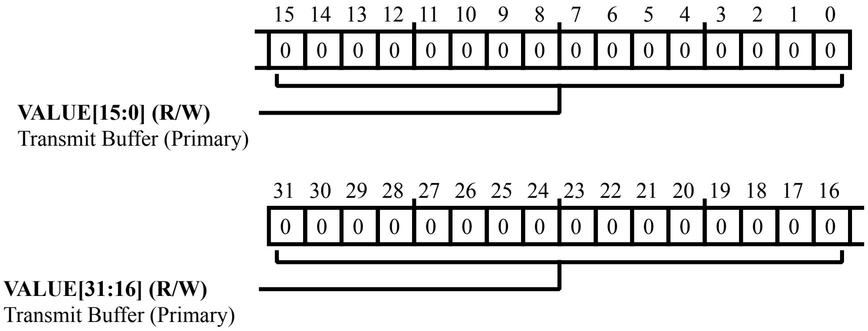


Figure 34-38: SPORT_TXPRI_B Register Diagram

Table 34-40: SPORT_TXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The `SPORT_TXPRI_B.VALUE` bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_B.MCE`) empty the contents of this data buffer. For more information, see the `SPORT_CTL_B` and `SPORT_MCTL_B` register descriptions.

Half SPORT 'A' Tx Buffer (Secondary) Register

The `SPORT_TXSEC_A` register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The `SPORT_TXSEC_A` register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_A.PACK` =0); while it acts as a two-location buffer when packing is enabled (`SPORT_CTL_A.PACK` =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the `SPORT_TXSEC_A` register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

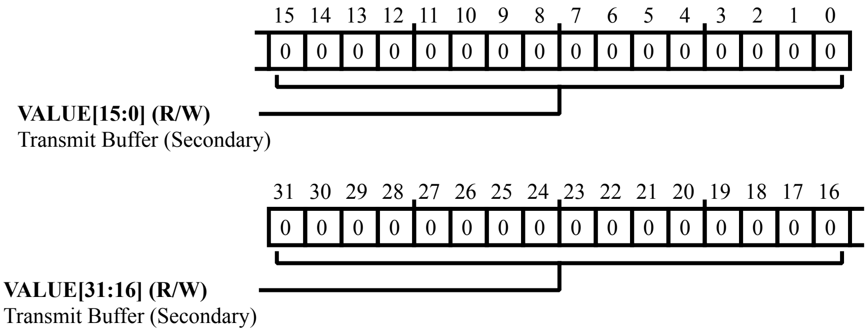


Figure 34-39: SPORT_TXSEC_A Register Diagram

Table 34-41: SPORT_TXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The `SPORT_TXSEC_A.VALUE` bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_A.MCE`) empty the contents of this data buffer. For more information, see the `SPORT_CTL_A` and `SPORT_MCTL_A` register descriptions.

Half SPORT 'B' Tx Buffer (Secondary) Register

The SPORT_TXSEC_B register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXSEC_B register acts as a three-location buffer if SPORT data packing is disabled (`SPORT_CTL_B.PACK` =0); while it acts as two-location buffer when packing is enabled (`SPORT_CTL_B.PACK` =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXSEC_B register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

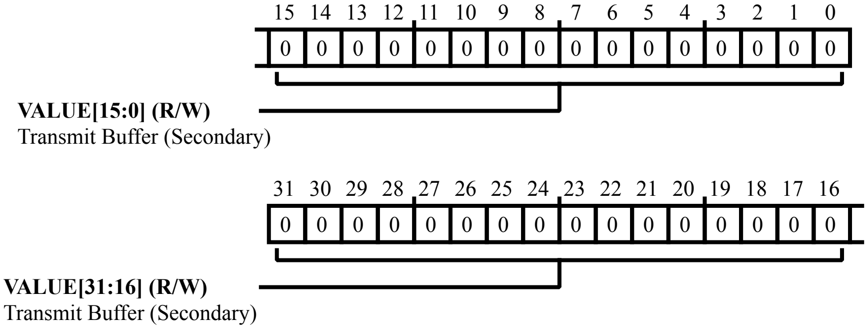


Figure 34-40: SPORT_TXSEC_B Register Diagram

Table 34-42: SPORT_TXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The `SPORT_TXSEC_B.VALUE` bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the `SPORT_MCTL_B.MCE`) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

35 Precision Clock Generators (PCG)

The precision clock generators are used to produce a pair of signals from a clock input signal. The two signals generated are normally used as a serial bit clock and frame sync pair. The PCG is part of the DAI. There are two PCG signals in each DAI: *PCGA* and *PCGB* belong to DAI0, *PCGC* and *PCGD* belong to DAI1.

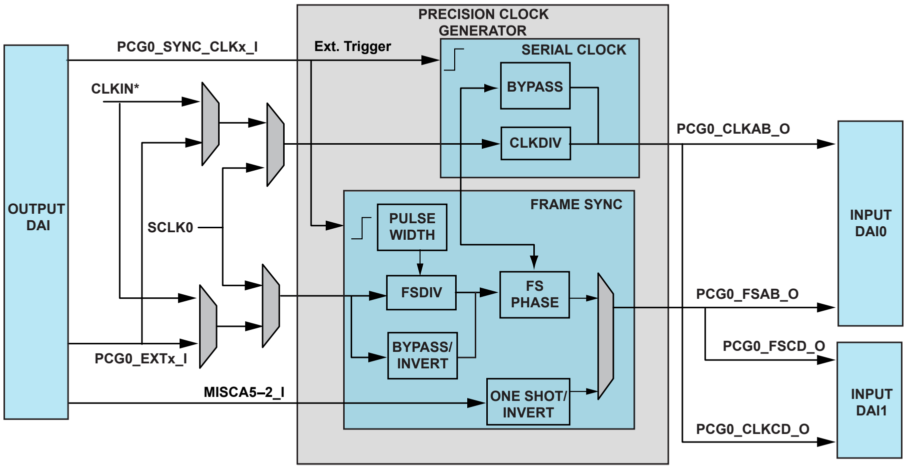
Features

The following list describes the features of the precision clock generators.

- SRU allows the routing of all of the PCG signals in one DAI (four PCG signals in DAI0 or DAI1)
- Input clock selection: SYS_CLKIN0 for PCG DAI0; SYS_CLKIN1 for PCG DAI1; SCLK0 for external DAI pins
- Provides four different clock dividers for serial clock, frame sync, phase (20-bit), and pulse width (16-bit)
- Phase shift allows adjustment of the frame sync relative to the serial clock and can be shifted the full period and wrapped around
- Provides pulse width control for arbitrary frame sync signal generation
- Bypass mode for external frame sync manipulation
- External trigger mode starts PCG operation
- No additional jitter is introduced when using off-chip clocks

Functional Description

The ***PCG Block Diagram*** shows the blocks within the module and its connection to the DAI. The following sections provide information on the function of these blocks.


PRECISION CLOCK GENERATOR
PCG0_SYNC_CLKx_I
Ext. Trigger
SERIAL CLOCK
CLKIN*
BYPASS
CLKDIV
PCG0_CLKAB_O
OUTPUT DAI
SCLK0
FRAME SYNC
PULSE WIDTH
INPUT DAI0
FSDIV
FS PHASE
PCG0_FSAB_O
BYPASS/ INVERT
PCG0_FSCD_O
PCG0_EXTx_I
MISCA5–2_I
ONE SHOT/ INVERT
INPUT DAI1
PCG0_CLKCD_O

* SYS_CLKIN0 is input to DAI0
SYS_CLKIN1 is input to DAI1

Figure 35-1: PCG Block Diagram

NOTE: In the *PCG Block Diagram*, SYS_CLKIN0 is input to DAI0 and SYS_CLKIN1 is input to DAI1.

ADSP-SC58x PCG Register List

Precision Clock Generator

Table 35-1: ADSP-SC58x PCG Register List

Name	Description
PCG_CTLA0	Precision Clock A Control 0 Register
PCG_CTLA1	Precision Clock A Control 1 Register
PCG_CTLB0	Precision Clock B Control 0 Register
PCG_CTLB1	Precision Clock B Control 1 Register
PCG_CTLC0	Precision Clock C Control 0 Register
PCG_CTLC1	Precision Clock C Control 1 Register
PCG_CTLD0	Precision Clock D Control 0 Register
PCG_CTLD1	Precision Clock D Control 1 Register
PCG_PW1	Precision Clock Pulse Width Control 1 Register
PCG_PW2	Precision Clock Pulse Width Control 2 Register
PCG_SYNC1	Precision Clock Frame Sync Synchronization 1 Register
PCG_SYNC2	Precision Clock Frame Sync Synchronization 2 Register

Internal Interface

The fundamental clock of the PCG is *SCLK0*. The clock to this module can be shut off for power savings.

Serial Clock

Each of the four units (A, B, C, and D) produces a clock output. Serial clock generation from a unit is independently enabled and controlled. Sources for the serial clock generation can be either from the *CLKIN*, *SCLK0*, or a DAI pin source. When CLKIN is chosen as input clock in PCG, the PCGs in DAI0 use `SYS_CLKIN0` and the PCGs in DAI1 use `SYS_CLKIN1`. The clock output is derived from the input to the PCG with a 20-bit divisor.

Note that the divider is working in normal mode for `PCG0_CTLx1.CLKDIV` > 1. For `PCG_CTLA1.CLKDIV`/`PCG_CTLB1.CLKDIV` = 0 or 1, the divider operates in bypass mode (input clock is fed directly to its output). In bypass mode, the clock at the output can theoretically run at up to the SCLK0 frequency. Check the data sheet for the specified maximum operation speed of the DAI pin buffers.

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.

NOTE: A PCG clock output cannot be fed to its own input.

Frame Sync

The following sections describe the use of frame syncs in the PCGs.

Frame Sync Output

Each of the four units (A through D) produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they must accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, SCLK0, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit.

If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, a 20-bit divisor controls frame sync generation.

Divider Mode Selection

If a frame sync divisor is greater than 1, the PCG frame sync output frequency is equal to the input clock frequency, divided by a 20-bit integer. This integer is specified in the `PCG_CTLA0.FSDIV`/`PCG_CTLB0.FSDIV` bit field (bits 19:0).

However, if the frame sync divisor is 0 or 1, the PCG's frame sync clock generation unit is bypassed, and the frame sync input is connected directly to the frame sync output. For `PCG_CTLB0.FSDIV` =0, 1 the `PCG_PW1`/`PCG_PW2` registers function differently than in normal mode.

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync relative to the input clock of the same unit. This feature allows the shifting of the frame sync signal in time relative to the clock input signal. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

For example, the I^2S protocol specifies that the frame sync transition from high-to-low occurs one clock cycle before the beginning of a frame. Since an I^2S frame is 64 clock cycles long, delaying the frame sync by 63 cycles produces the required framing.

Phase shifting is represented as a full 20-bit value. Even when the frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.

NOTE: Phase shifting is specified as a 2 x 10-bit divider value in the `PCG_CTLA0.FSPHASEHI` bit field (bits 29:20) and in the `PCG_CTLA1.FSPHASELO` bit field (bits 29:20).

A single 20-bit value spans these two-bit fields. The upper half of the word (bits 19:10) is in the `PCG_CTLA0` register, and the lower half (bits 9:0) is in the `PCG_CTLA1` register.

The phase shift between clock and frame sync outputs can be programmed using the `PCG_PW1`/`PCG_PW2` registers and all of the control registers under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- The clock and frame sync are enabled at the same time using a single atomic instruction.
- The frame sync divisor is an integral multiple of the clock divisor.

NOTE: When using a clock and frame sync as a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction.

NOTE: If the phase shift is 0 (see the ***Phase and Pulse Width Settings*** figure), the clock and frame sync outputs rise at the same time.

If the phase shift is 1, the frame sync output transitions one input clock period ahead of the clock transition.

If the phase shift is *divisor* – 1, the frame sync transitions *divisor* – 1 input clock periods ahead of the clock transitions.

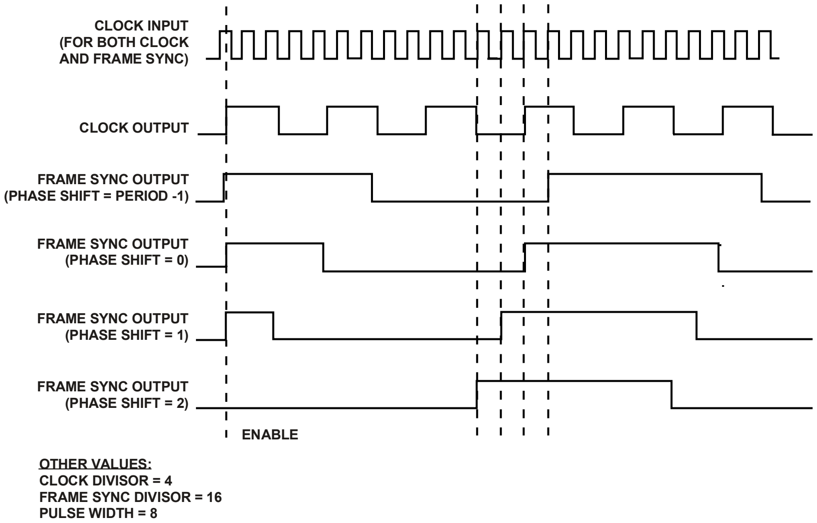

CLOCK INPUT
(FOR BOTH CLOCK
AND FRAME SYNC)
CLOCK OUTPUT
FRAME SYNC OUTPUT
(PHASE SHIFT = PERIOD -1)
FRAME SYNC OUTPUT
(PHASE SHIFT = 0)
FRAME SYNC OUTPUT
(PHASE SHIFT = 1)
FRAME SYNC OUTPUT
(PHASE SHIFT = 2)
ENABLE
OTHER VALUES:
CLOCK DIVISOR = 4
FRAME SYNC DIVISOR = 16
PULSE WIDTH = 8

Figure 35-2: Phase and Pulse Width Settings

NOTE: When generating single frame sync pulses (the length of one SPORT clock cycle), take care with respect to the drive and sampling edges. If the rules are violated, for example if the SPORT is not driving data, the module cannot detect a valid sample edge.

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high.

A 16-bit value determines the width of the framing pulse. Settings for pulse width can range from zero to `DIV` - 1. The pulse width should be less than the divisor of the frame sync. The pulse width of frame sync is specified in the `PCG_PW1.FSA`, `PCG_PW1.FSB`, `PCG_PW2.FSC`, and `PCG_PW2.FSD` bits (15-0) and (31-16).

Default Pulse Width

If the pulse width count is equal to 0 and if the `PCG_CTLA0.FSDIV`/`PCG_CTLB0.FSDIV` bit field is even, then the actual pulse width of the frame sync output is equal to:

For even divisors: frame sync divisor/2

If the pulse width count is equal to 0 and if the `PCG_CTLA0.FSDIV`/`PCG_CTLB0.FSDIV` bit field is odd, then the actual pulse width of the frame sync output is equal to:

For odd divisors: frame sync divisor – 1/2

Input Clock Source Considerations

The core Phase-Locked Loop (PLL) has been designed to provide clocking for the processor core. The performance specifications of this PLL are appropriate for the core. But, they have not been optimized or specified for precision data converters where jitter directly translates into time quantization errors and distortion.

Therefore, the PCG allows the routing of external clock sources which are independent of the core PLL.

Timing Example for I^2S Mode

For I^2S mode, the frame sync should be driven at the falling edge of SPORT clock. In other words, the frame sync edge must coincide with the falling edge of the SPORT clock. To satisfy this requirement, program the phase of the frame sync accordingly in the PCG control registers.

For example, assume that the input clock source for both clock and frame sync are the same and both the clock and frame sync are enabled at the same time. Also, assume that the clock divisor value for generating the required SPORT clock is `PCG_CTLA1.CLKDIV` = 4. Then, for a 32-bit word length, the frame sync divisor value is:

`PCG_CTLA0.FSDIV` = 64, `PCG_CTLA1.CLKDIV` = 256.

By default, for phase = 0, the rising edge of both SPORT clock and frame sync coincide. To make sure that the frame sync edges coincide with the falling edge of the SPORT clock, program the phase value as:

`PCG_CTLA1.CLKDIV`/2 = 2.

Cross Mode Connections

The symmetric dual DAI architecture allows cross connections between both PCGs (A,B) and (C,D) to the other DAI. Each PCG (A through D) supports an alternative input clock (PCG0_EXTx_I) (see Figure 35-1 PCG Block Diagram) which can be sourced via a DAI pin buffer from the other DAI. Note however if routing a source (clock or FS) only DAI pin buffer 2 to 20 can be used (DAI pin buffer 1 is no longer available and is replaced by the DAI CRS buffer for the other DAI). See DAI Routing Capabilities for more information.

Operating Modes

The following sections provide information on the operating modes of the precision clock generator.

Normal Mode

When the frame sync divisor is set to any value other than zero or one, the PCGs operate in normal mode. In normal mode, the divisor determines the frequency of the frame sync output where:

Frequency of Frame Sync Output = Input Frequency/Divisor

The value of the pulse width control determines the high period of the frame sync output. The value of the pulse width control must be less than the value of the divisor.

The value of the phase control determines the phase of the frame sync output. If the phase is zero, then the positive edges of the clock and frame sync coincide when:

- the clock and frame sync dividers are enabled at the same time using an atomic instruction
- the divisors of the clock and frame sync are the same
- the source for the clock and frame sync is the same

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the difference between the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame sync appears to lead the clock. If the phase is only slightly less than the frame sync divisor, then the frame sync appears to lag the clock. The frame sync phase must not be greater than the divisor.

Bypass Mode

When the frame sync divisor for the frame sync has a value of zero or one, the frame sync is in bypass mode, and the `PCG_PW1`/`PCG_PW2` registers have different functionality than in normal mode.

NOTE: In normal mode, bits 15:0 and 31:18 of the `PCG_PW1`/`PCG_PW2` registers are used to program the pulse width count. In bypass mode, bits 15:2 and 31:18 are ignored. Bits 1:0 and 17:16 are renamed to `PCG_PW1.STROBEA` and `PCG_PW1.INVFSA`, respectively. This functionality is described in more detail as follows.

If the `PCG_PW1.STROBEA` register is cleared, then the input is directly passed (see the ***Bypass and Inverted Bypass*** figure) to the frame sync output either inverted or not inverted, depending on the `PCG_PW1.INVFSA` bits.

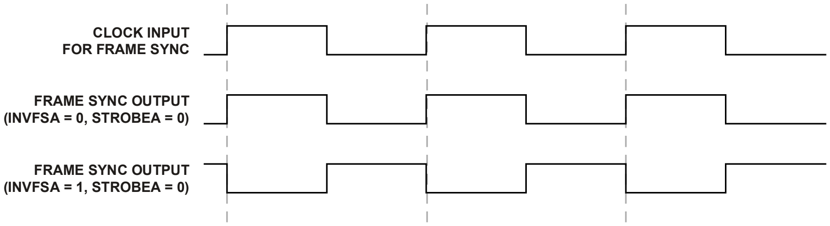

CLOCK INPUT
FOR FRAME SYNC
FRAME SYNC OUTPUT
(INVFSA = 0, STROBEA = 0)
FRAME SYNC OUTPUT
(INVFSA = 1, STROBEA = 0)

Figure 35-3: Bypass and Inverted Bypass

One-Shot Mode

In the one-shot mode operation shown in the ***One Shot Mode PCG A (MISCA2_I Input)*** figure, the PCG produces a series of periods but does not run continuously.

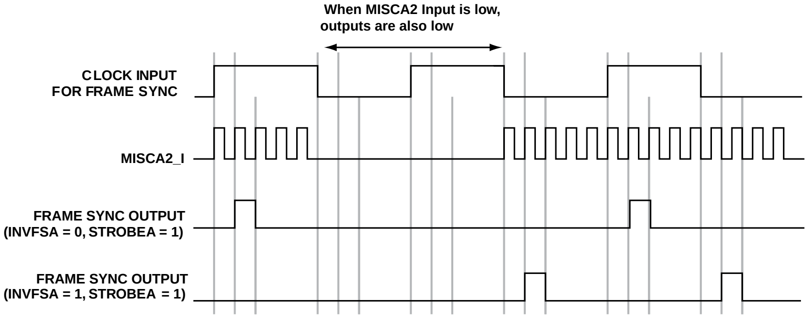

When MISCA2 Input is low,
outputs are also low
CLOCK INPUT
FOR FRAME SYNC
MISCA2_I
FRAME SYNC OUTPUT
(INVFSA = 0, STROBEA = 1)
FRAME SYNC OUTPUT
(INVFSA = 1, STROBEA = 1)

Figure 35-4: One Shot Mode PCG A (MISCA2_I Input)

Bypass mode also enables the generation of a strobe pulse (one-shot frame sync). Strobe usage ignores the divider counters and looks to the SRU to provide the input signal. Two-bit fields determine the operation in this mode.

In the bypass mode, if the `PCG_PW1.STROBEA` bit =1, then a one-shot pulse is generated. This one-shot pulse has the duration equal to the period of `DAI_MISCAx_I` for the PCGx unit. This pulse is generated either at the falling or rising edge of the input clock, depending on the value of the `PCG_PW1.INVFSA` bit. The output pulse width is equal to the period of the SRU source signal `DAI_MISCAx_I`. The pulse begins at the second rising edge of `MISCAx_I` following a rising edge of the clock input. When the `PCG_PW1.INVFSA` bit is set, the pulse begins at the second rising edge of `DAI_MISCAx_I` coinciding with or following a falling edge of the clock input.

NOTE: A strobe period is defined to be the period of the FS input clock signal as specified by the `PCG_CTLA1.FSSRC` bit.

PCG Event Control

The following sections describe the generation and control of PCG events.

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the `PCG_SYNC1`/`PCG_SYNC2` registers .

Refer to the ***FS Output Synchronization With External Trigger Input*** figure. Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

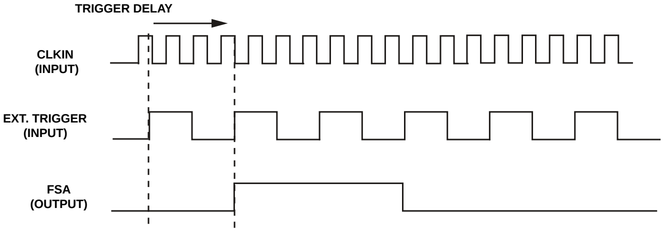


Figure 35-5: FS Output Synchronization With External Trigger Input

External Event Trigger Delay

The time delay between the rising trigger edge and the start of serial clock and frame sync varies between 2.5 to 3.5 input clock periods. If the input clock and the trigger signal are synchronous, the delay is 3 input clock periods. Consider the following cases:

- SCLK0 is the input source. In this case, if the given trigger event is synchronous to SCLK0, the delay is 3 SCLK0 periods. If the trigger signal is asynchronous with SCLK0, the delay varies from 2.5 SCLK0 periods to 3.5 SCLK0 periods. (It depends on whether the trigger edge occurs in the positive half cycle or negative half cycle of SCLK0.)
- The `SYS_CLKIN0` signal is the input source for PCG AB. The `SYS_CLKIN1` signal is the input source for PCG CD. In this case if the given trigger signal is synchronous to CLKIN, the delay is 3 CLKIN periods. But if they are asynchronous to CLKIN, the delay can vary between 2.5 CLKIN periods to 3.5 CLKIN periods.
- SRU DAI0 is the input source for PCG AB. SRU DAI1 is the input source for PCG CD. If the input clock and trigger signal are synchronous, the delay is exactly 3 input clock periods. If asynchronous, it varies between 2.5 to 3.5 input clock periods depending on the phase difference between the input clock and trigger signal.

Audio System Example

The *PCG Setup for I2S or Left-Justified DAI* figure shows an example of the interconnections between the S/PDIF receiver, ASRC, and the PCGs. The interconnections are made by programming the signal routing unit. It shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (ASRC) to interface to an external audio DAC. The PCG is configured to provide a fixed ASRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player, but can also be from other source at any nominal sample rates.

Similarly, the phase shift for frame syncs B, C, and D is specified in the corresponding control registers (`PCG_CTLA0` through `PCG_CTLB1`).

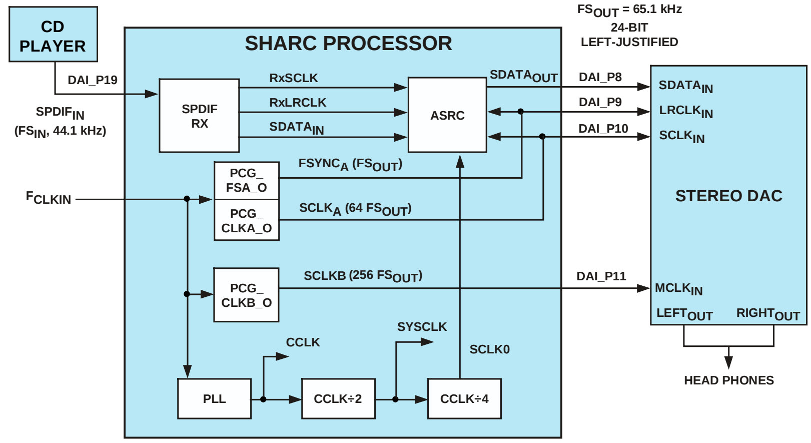

FS_{OUT} = 65.1 kHz
24-BIT
LEFT-JUSTIFIED
CD PLAYER
SHARC PROCESSOR
DAI_P19
$SPDIF_{IN}$
(FS_{IN}, 44.1 kHz)
SPDIF RX
RxSCLK
RxLRCLK
$SDATA_{IN}$
ASRC
$SDATA_{OUT}$
DAI_P8
DAI_P9
DAI_P10
$SDATA_{IN}$
$LRCLK_{IN}$
$SCLK_{IN}$
STEREO DAC
F_{CLKIN}
PCG_ FSA_O
PCG_ CLKA_O
$FSYNC_A$ (FS_{OUT})
$SCLK_A$ (64 FS_{OUT})
PCG_ CLKB_O
SCLKB (256 FS_{OUT})
DAI_P11
$MCLK_{IN}$
$LEFT_{OUT}$
$RIGHT_{OUT}$
HEAD PHONES
SYSCLK
CCLK
SCLK0
PLL
CCLK÷2
CCLK÷4

Figure 35-6: PCG Setup for I^2S or Left-Justified DAI

Three synchronous clocks are required in audio systems:

1. Frame sync (FS)
2. Serial bit clock (64 FS)
3. Master DAC clock (256 FS)

Since each PCG has only two outputs, this example requires two PCGs. Furthermore, because the digital audio interface requires a fixed-phase relation between serial clock and FS, these two outputs should come from one PCG (PCG A). The master clock comes from the second (PCG B).

The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the ASRCs and external DACs. However, the range of choices is limited by CLKIN and the ratio of `PCG_CLKx_O`: serial clock:*FS*. The ratio is normally fixed at 256:64:1 to support digital audio left-justified, I^2S and right-justified interface modes.

Many DACs also support 384, 512, and 786x FS for `PCG_CLKx_O`, which allows some additional flexibility in choosing serial clock.

Note that the falling edge of serial clock must always be synchronous with both edges of FS. This condition requires that the phase of the serial clock and FS signals for a common PCG (PCG A) be adjustable.

While the frequency of the master DAC clock (`PCG_CLKx_O`) must be synchronous with the sample rate supplied to the external DAC, there is no fixed-phase requirement.

Set the clock divisor and source and low-phase word first, followed by the control register enable bits, which are set together. When the `PCG_PW1`/`PCG_PW2` registers are set to zero (default), the FS pulse width is (divisor 2) for even divisors and (divisor - 1) 2 for odd divisors. Alternatively, the `PCG_PW1`/`PCG_PW2` registers can be set high for exactly one-half the period of CLKIN cycles for a 50% duty cycle, provided the FS divisor is an even number.

Clock Configuration Examples

For a CLKIN = 33.330 MHz, the two PCGs provide the three synchronous clocks PCGx_CLK, serial clock and *FS* for the SRCs and external DAC. These divisors are stored in the `PCG_CTLA1.CLKDIV`/`PCG_CTLB1.CLKDIV` bit fields.

The integer divisors for several possible sample rates based on 33.330 MHz *CLKIN* are shown in the ***Precision Clock Generator Division Ratios (33.330 CLKIN)*** table.

Table 35-2: Precision Clock Generator Division Ratios (33.330 CLKIN)

	PCG Divisors		FSDIV A[*1]
Sample Rate kHz)	CLKDIV B	CLKDIV A	
130.195	1	4	256
65.098	*2*	*8*	*512*
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280
21.699	6	24	1536
18.599	7	28	1792

[*1] The frame sync divisor should be an even integer in order to produce a 50% duty cycle waveform. See Frame Sync

Programming Model

This section describes the sequence of software steps required for successful PCG operation.

If the PCG is disabled to reprogram a parameter, use a delay after writing to the disable bit. This delay in core clock (*CCLK*) cycles = (PCG source clock period/*CCLK* period). In summary, use the following general procedure:

1. Clear the PCG enable bits without modifying any other settings.
2. Wait for N *CCLK* cycles (N = PCG source clock period/processor clock period).
3. Program all new parameters without setting the PCG enable bit.
4. Enable the PCG.

Frame Sync Phase Setting

The phase unit requires that the clock and FS are enabled simultaneously in an atomic instruction.

1. Write the clock divider/low 10-bit phase divider to the `PCG_CTLA1`/`PCG_CTLB1` registers.
2. Program the FS divider/high 10-bit phase divider, enabling both the `PCG_CTLA0.CLKEN`/`PCG_CTLA0.FSEN` and the `PCG_CTLB0.CLKEN`/`PCG_CTLB0.FSEN` bits.

Note that both units must be disabled in the same way.

External Event Trigger

The trigger with the external clock is is enabled by setting bits 0 and 16 of the `PCG_SYNC` register. Program the phase to 3, so that the rising edge of the external clock is in-sync with the frame sync (***FS Output Synchronization With External Trigger Input***).

Use the following steps.

1. Program the `PCG_SYNC1`/`PCG_SYNC2` registers and the `PCG_CTLA0` through `PCG_CTLB1` registers appropriately.
2. Enable the clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

Debug Features

Take care in cases where any input to the phase unit is modified. Any individual change of the `PCG_CTLA1.CLKDIV` or `PCG_CTLA0.FSDIV` dividers can cause a failure in the PCG sync operation between the serial clock and the frame sync. Only the programming model ensures a correct setup for phase settings.

ADSP-SC58x PCG Register Descriptions

Precision Clock Generator (PCG) contains the following registers.

Table 35-3: ADSP-SC58x PCG Register List

Name	Description
`PCG_CTLA0`	Precision Clock A Control 0 Register
`PCG_CTLA1`	Precision Clock A Control 1 Register
`PCG_CTLB0`	Precision Clock B Control 0 Register
`PCG_CTLB1`	Precision Clock B Control 1 Register
`PCG_CTLC0`	Precision Clock C Control 0 Register
`PCG_CTLC1`	Precision Clock C Control 1 Register

Table 35-3: ADSP-SC58x PCG Register List (Continued)

Name	Description
`PCG_CTLD0`	Precision Clock D Control 0 Register
`PCG_CTLD1`	Precision Clock D Control 1 Register
`PCG_PW1`	Precision Clock Pulse Width Control 1 Register
`PCG_PW2`	Precision Clock Pulse Width Control 2 Register
`PCG_SYNC1`	Precision Clock Frame Sync Synchronization 1 Register
`PCG_SYNC2`	Precision Clock Frame Sync Synchronization 2 Register

Precision Clock A Control 0 Register

The `PCG_CTLA0` register enables the clock, frame sync, and select divisor for the PCG0 clock A signal.

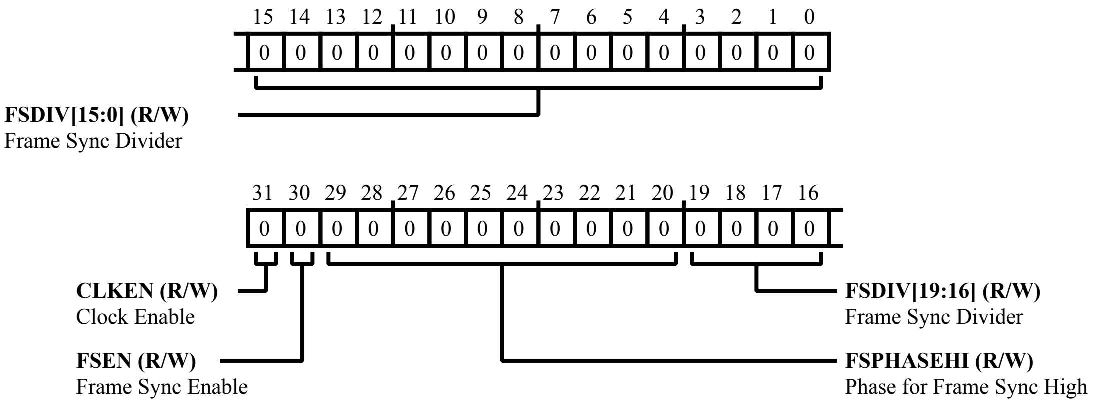

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSDIV[15:0] (R/W)
Frame Sync Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKEN (R/W)
Clock Enable
FSEN (R/W)
Frame Sync Enable
FSDIV[19:16] (R/W)
Frame Sync Divider
FSPHASEHI (R/W)
Phase for Frame Sync High

Figure 35-7: PCG_CTLA0 Register Diagram

Table 35-4: PCG_CTLA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKEN	Clock Enable. The `PCG_CTLA0.CLKEN` bit enables the clock.	
		0	Clock generation disabled
		1	Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The `PCG_CTLA0.FSEN` bit enables the frame sync.	
		0	Frame sync generation disabled
		1	Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The `PCG_CTLA0.FSPHASEHI` bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	FSDIV	Frame Sync Divider. The `PCG_CTLA0.FSDIV` bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: `PCG_CTLA0.FSDIV` >1 PCGx is in normal mode, `PCG_CTLA0.FSDIV` = 0, 1 PCGx is in bypass mode.	
		0	PCG is in bypass mode
		1	PCG is in bypass mode
		2-1048575	PCG is in normal mode

Precision Clock A Control 1 Register

The `PCG_CTLA1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock A signal.

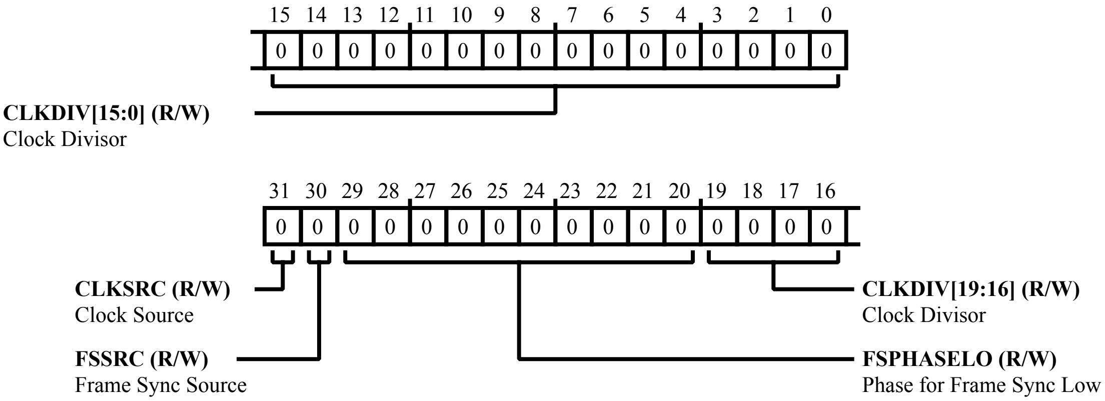

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKDIV[15:0] (R/W)
Clock Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKSRC (R/W)
Clock Source
FSSRC (R/W)
Frame Sync Source
CLKDIV[19:16] (R/W)
Clock Divisor
FSPHASELO (R/W)
Phase for Frame Sync Low

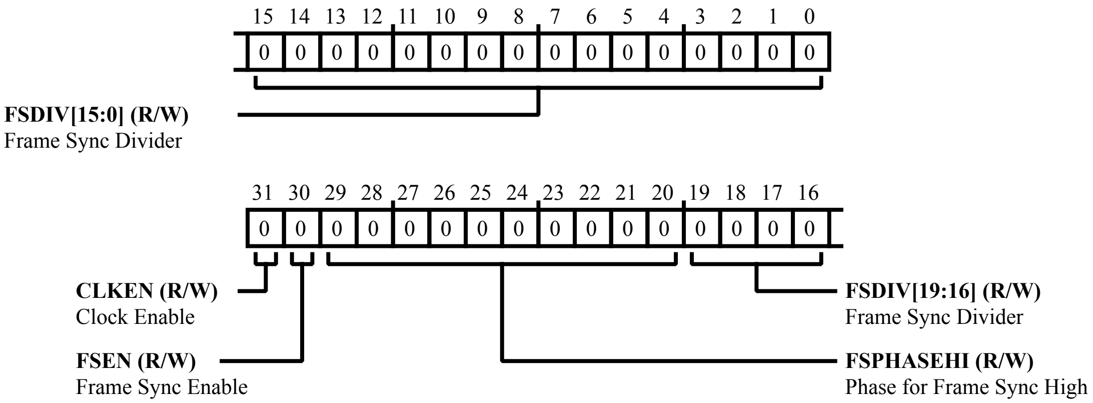
Figure 35-8: PCG_CTLA1 Register Diagram

Table 35-5: PCG_CTLA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKSRC	Clock Source. The `PCG_CTLA1.CLKSRC` bit specifies the clock source.	
		0	CLKIN0 pin selected for clock
		1	PCG_EXT_DAI0 selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The `PCG_CTLA1.FSSRC` bit specifies the frame sync source.	
		0	CLKIN0 pin selected for frame sync
		1	PCG_EXTX_DAI0 selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The `PCG_CTLA1.FSPHASELO` bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	CLKDIV	Clock Divisor. The `PCG_CTLA1.CLKDIV` bit field contains the clock divisor value.	

Precision Clock B Control 0 Register

The PCG_CTLB0 register enables the clock, frame sync, and select divisor for the PCG0 clock B signal.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSDIV[15:0] (R/W)
Frame Sync Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKEN (R/W)
Clock Enable
FSEN (R/W)
Frame Sync Enable
FSDIV[19:16] (R/W)
Frame Sync Divider
FSPHASEHI (R/W)
Phase for Frame Sync High

Figure 35-9: PCG_CTLB0 Register Diagram

Table 35-6: PCG_CTLB0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKEN	Clock Enable. The `PCG_CTLB0.CLKEN` bit enables the clock.	
		0	Clock generation disabled
		1	Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The `PCG_CTLB0.FSEN` bit enables the frame sync.	
		0	Frame sync generation disabled
		1	Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The `PCG_CTLB0.FSPHASEHI` bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	FSDIV	Frame Sync Divider. The `PCG_CTLB0.FSDIV` bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: `PCG_CTLB0.FSDIV` >1 PCGx is in normal mode, `PCG_CTLB0.FSDIV` = 0, 1 PCGx is in bypass mode.	
		0	PCG is in bypass mode
		1	PCG is in bypass mode
		2-1048575	FSDIV >1 PCG is in normal mode

Precision Clock B Control 1 Register

The PCG_CTLB1 register sets the clock divisor, frame sync source, and clock source for the PCG1 clock B signal.

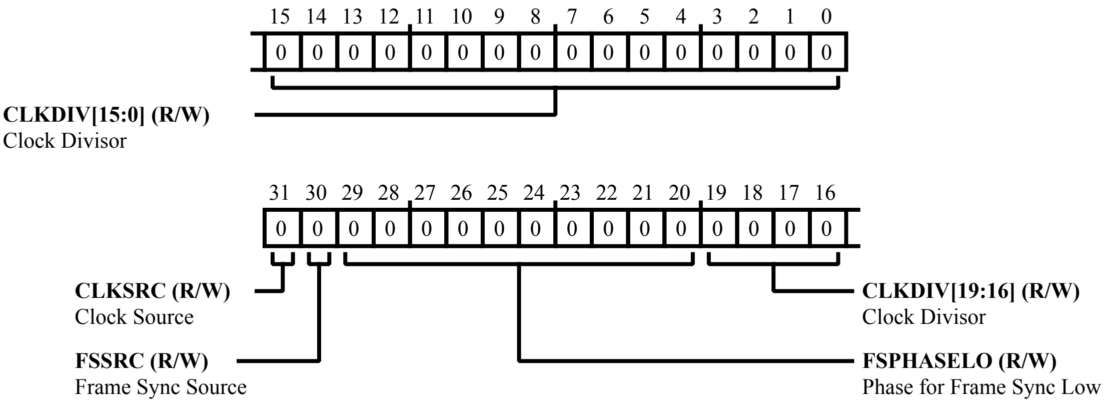

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKDIV[15:0] (R/W)
Clock Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKSRC (R/W)
Clock Source
FSSRC (R/W)
Frame Sync Source
CLKDIV[19:16] (R/W)
Clock Divisor
FSPHASELO (R/W)
Phase for Frame Sync Low

Figure 35-10: PCG_CTLB1 Register Diagram

Table 35-7: PCG_CTLB1 Register Fields

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">31
(R/W)</td><td rowspan="3">CLKSRC</td><td colspan="2">Clock Source.
The `PCG_CTLB1.CLKSRC` bit specifies the clock source.</td></tr>
<tr><td>0</td><td>CLKIN0 pin selected for clock</td></tr>
<tr><td>1</td><td>PCG_EXT_DAI0 selected for clock</td></tr>
<tr><td rowspan="3">30
(R/W)</td><td rowspan="3">FSSRC</td><td colspan="2">Frame Sync Source.
The `PCG_CTLB1.FSSRC` bit specifies the frame sync source.</td></tr>
<tr><td>0</td><td>CLKIN0 pin selected for frame sync</td></tr>
<tr><td>1</td><td>PCG_EXT_DAI0 selected for frame sync</td></tr>
<tr><td>29:20
(R/W)</td><td>FSPHASELO</td><td colspan="2">Phase for Frame Sync Low.
The `PCG_CTLB1.FSPHASELO` bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.</td></tr>
<tr><td>19:0
(R/W)</td><td>CLKDIV</td><td colspan="2">Clock Divisor.
The `PCG_CTLB1.CLKDIV` bit field contains the clock divisor value.</td></tr>
</table>

Precision Clock C Control 0 Register

The PCG_CTLC0 register enables the clock, frame sync, and select divisor for the PCG0 clock C signal.

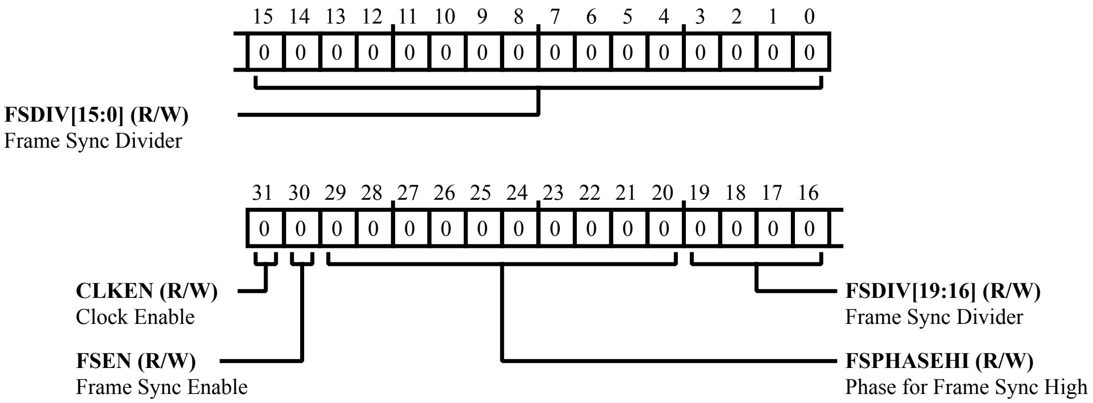

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSDIV[15:0] (R/W)
Frame Sync Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKEN (R/W)
Clock Enable
FSEN (R/W)
Frame Sync Enable
FSDIV[19:16] (R/W)
Frame Sync Divider
FSPHASEHI (R/W)
Phase for Frame Sync High

Figure 35-11: PCG_CTLC0 Register Diagram

Table 35-8: PCG_CTLC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKEN	Clock Enable. The PCG_CTLC0.CLKEN bit enables the clock.	
		0	Clock generation disabled
		1	Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The PCG_CTLC0.FSEN bit enables the frame sync.	
		0	Frame sync generation disabled
		1	Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The PCG_CTLC0.FSPHASEHI bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	FSDIV	Frame Sync Divider. The PCG_CTLC0.FSDIV bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: PCG_CTLC0.FSDIV >1 PCGx is in normal mode, PCG_CTLC0.FSDIV = 0, 1 PCGx is in bypass mode.	
		0	PCG is in bypass mode
		1	PCG is in bypass mode
		2-1048575	FSDIV >1 PCG is in normal mode

Precision Clock C Control 1 Register

The `PCG_CTLC1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock C signal.

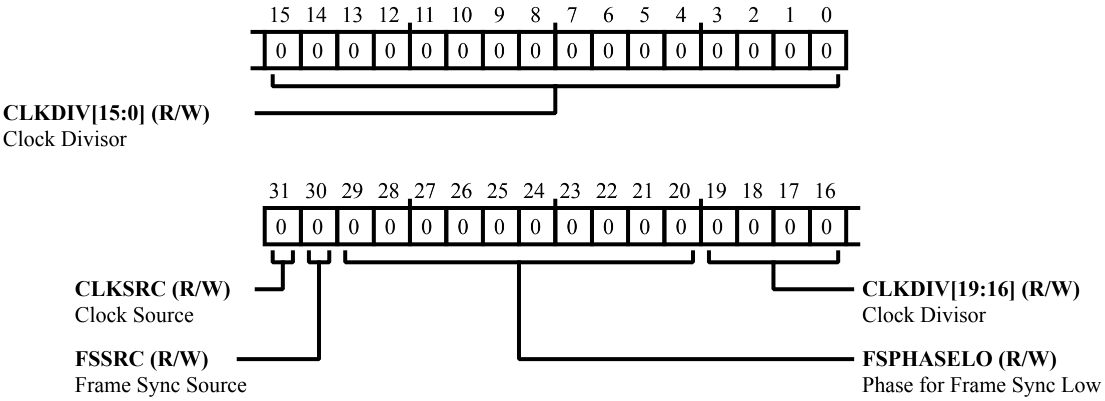

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKDIV[15:0] (R/W)
Clock Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKSRC (R/W)
Clock Source
FSSRC (R/W)
Frame Sync Source
CLKDIV[19:16] (R/W)
Clock Divisor
FSPHASELO (R/W)
Phase for Frame Sync Low

Figure 35-12: PCG_CTLC1 Register Diagram

Table 35-9: PCG_CTLC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKSRC	Clock Source. The `PCG_CTLC1.CLKSRC` bit specifies the clock source.	
		0	CLKIN1 pin selected for clock
		1	PCG_EXT_DAI1 selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The `PCG_CTLC1.FSSRC` bit specifies the frame sync source.	
		0	CLKIN1 pin selected for frame sync
		1	PCG_EXT_DAI1 selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The `PCG_CTLC1.FSPHASELO` bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	CLKDIV	Clock Divisor. The `PCG_CTLC1.CLKDIV` bit field contains the clock divisor value.	

Precision Clock D Control 0 Register

The `PCG_CTLD0` register enables the clock, frame sync, and select divisor for the PCG0 clock D signal.

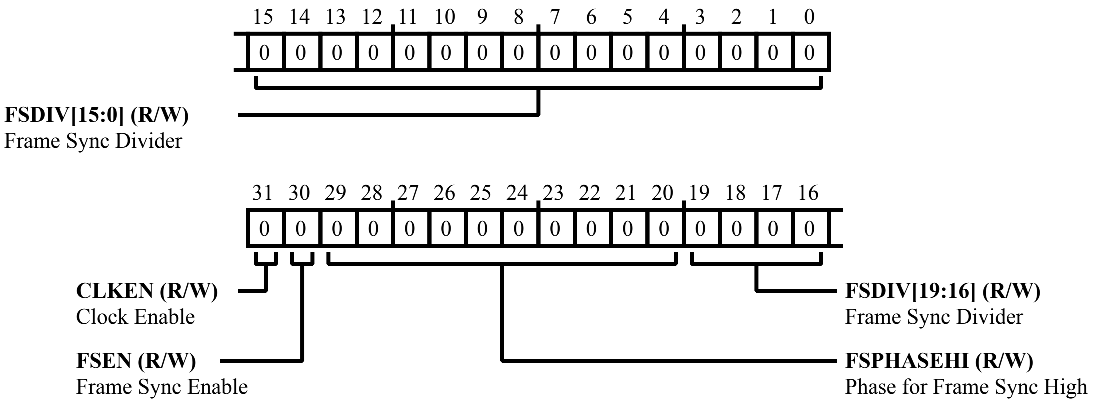

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSDIV[15:0] (R/W)
Frame Sync Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKEN (R/W)
Clock Enable
FSEN (R/W)
Frame Sync Enable
FSDIV[19:16] (R/W)
Frame Sync Divider
FSPHASEHI (R/W)
Phase for Frame Sync High

Figure 35-13: PCG_CTLD0 Register Diagram

Table 35-10: PCG_CTLD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKEN	Clock Enable. The `PCG_CTLD0.CLKEN` bit enables the clock.	
		0	Clock generation disabled
		1	Clock generation enabled
30 (R/W)	FSEN	Frame Sync Enable. The `PCG_CTLD0.FSEN` bit enables the frame sync.	
		0	Frame sync generation disabled
		1	Frame sync generation enabled
29:20 (R/W)	FSPHASEHI	Phase for Frame Sync High. The `PCG_CTLD0.FSPHASEHI` bit field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	FSDIV	Frame Sync Divider. The `PCG_CTLD0.FSDIV` bit field provides the frame sync divider value. This 20-bit field frame sync divider is multiplexed where: `PCG_CTLD0.FSDIV` >1 PCGx is in normal mode, `PCG_CTLD0.FSDIV` = 0, 1 PCGx is in bypass mode.	
		0	PCG is in bypass mode
		1	PCG is in bypass mode
		2-1048575	FSDIV >1 PCG is in normal mode

Precision Clock D Control 1 Register

The `PCG_CTLD1` register sets the clock divisor, frame sync source, and clock source for the PCG1 clock D signal.

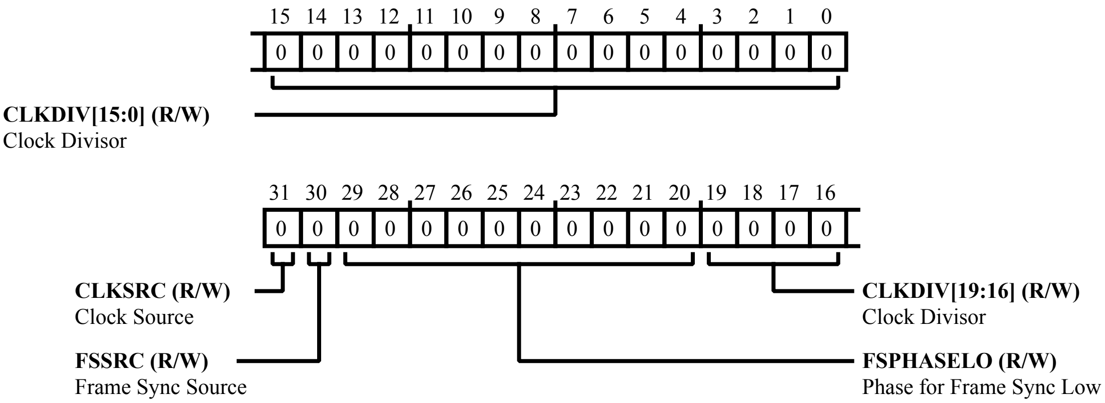

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKDIV[15:0] (R/W)
Clock Divisor
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLKSRC (R/W)
Clock Source
FSSRC (R/W)
Frame Sync Source
CLKDIV[19:16] (R/W)
Clock Divisor
FSPHASELO (R/W)
Phase for Frame Sync Low

Figure 35-14: PCG_CTLD1 Register Diagram

Table 35-11: PCG_CTLD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKSRC	Clock Source. The `PCG_CTLD1.CLKSRC` bit specifies the clock source.	
		0	CLKIN1 pin selected for clock
		1	PCG_EXT_DAI1 selected for clock
30 (R/W)	FSSRC	Frame Sync Source. The `PCG_CTLD1.FSSRC` bit specifies the frame sync source.	
		0	CLKIN1 pin selected for frame sync
		1	PCG_EXT_DAI1 selected for frame sync
29:20 (R/W)	FSPHASELO	Phase for Frame Sync Low. The `PCG_CTLD1.FSPHASELO` bit field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase.	
19:0 (R/W)	CLKDIV	Clock Divisor. The `PCG_CTLD1.CLKDIV` bit field contains the clock divisor value.	

Precision Clock Pulse Width Control 1 Register

The `PCG_PW1` register sets the one shot frame sync and the active low frame sync select for PCG A and PCG B.

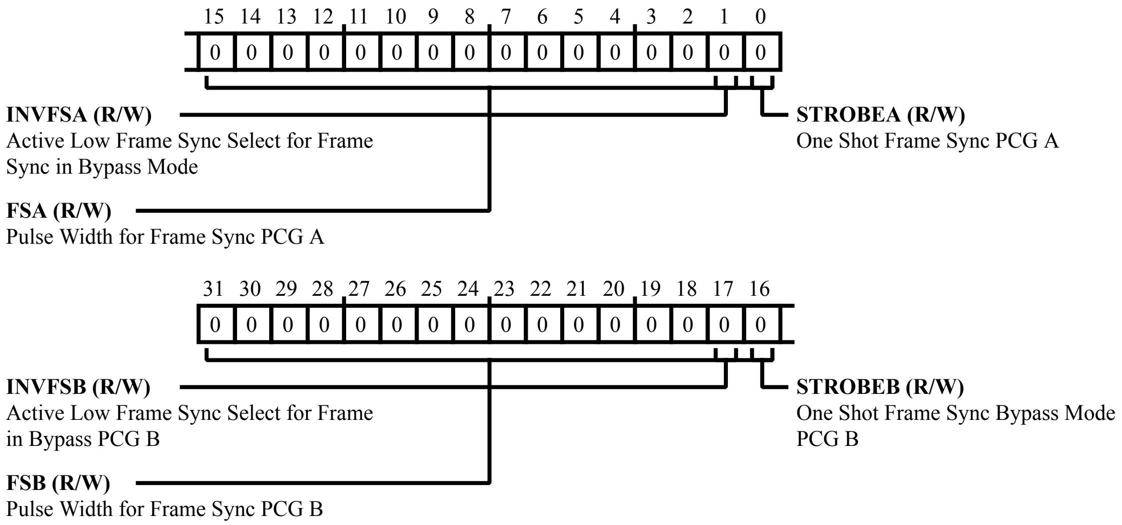

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INVFSA (R/W)
Active Low Frame Sync Select for Frame Sync in Bypass Mode
FSA (R/W)
Pulse Width for Frame Sync PCG A
STROBEA (R/W)
One Shot Frame Sync PCG A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INVFSB (R/W)
Active Low Frame Sync Select for Frame in Bypass PCG B
FSB (R/W)
Pulse Width for Frame Sync PCG B
STROBEB (R/W)
One Shot Frame Sync Bypass Mode PCG B

Figure 35-15: PCG_PW1 Register Diagram

Table 35-12: PCG_PW1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	INVFSB	Active Low Frame Sync Select for Frame in Bypass PCG B. The `PCG_PW1.INVFSB` bit selects active low or active high frame sync in bypass mode for PCG B.
16 (R/W)	STROBEB	One Shot Frame Sync Bypass Mode PCG B. The `PCG_PW1.STROBEB` bit sets the frame sync pulse in bypass mode for PCG B. This is the duration equal to one period of the DAI_MISCA2_I signal (PCG B) repeating at the beginning of every frame.
31:16 (R/W)	FSB	Pulse Width for Frame Sync PCG B. The `PCG_PW1.FSB` bit field sets the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync.
1 (R/W)	INVFSA	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The `PCG_PW1.INVFSA` bit selects active low or active high frame sync for PCG A in bypass mode.
15:0 (R/W)	FSA	Pulse Width for Frame Sync PCG A. The `PCG_PW1.FSA` bit field sets the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync.

Table 35-12: PCG_PW1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	STROBEA	One Shot Frame Sync PCG A. The `PCG_PW1.STROBEA` bit sets the frame sync pulse for PCG A in bypass mode. This is the duration equal to one period of the DAI_MISCA2_I signal (PCG A) repeating at the beginning of every frame.

Precision Clock Pulse Width Control 2 Register

The PCG_PW2 register sets the one shot frame sync and the active low frame sync select for PCG C and PCG D.

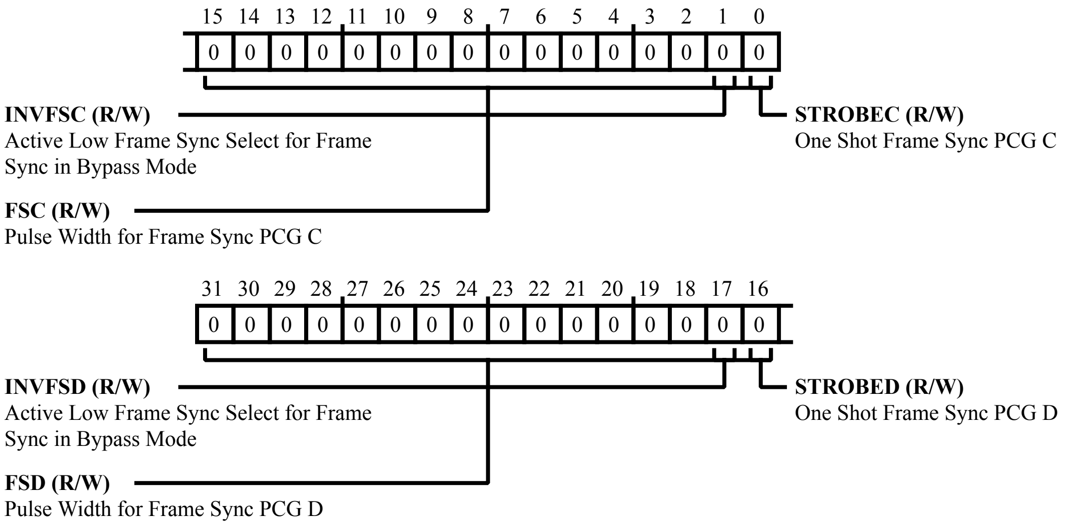

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INVFSC (R/W)
Active Low Frame Sync Select for Frame Sync in Bypass Mode
FSC (R/W)
Pulse Width for Frame Sync PCG C
STROBEC (R/W)
One Shot Frame Sync PCG C
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INVFSD (R/W)
Active Low Frame Sync Select for Frame Sync in Bypass Mode
FSD (R/W)
Pulse Width for Frame Sync PCG D
STROBED (R/W)
One Shot Frame Sync PCG D

Figure 35-16: PCG_PW2 Register Diagram

Table 35-13: PCG_PW2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	INVFSD	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The `PCG_PW2.INVFSD` bit selects active low or active high frame sync for PCG D in bypass mode.
16 (R/W)	STROBED	One Shot Frame Sync PCG D. The `PCG_PW2.STROBED` bit sets the frame sync pulse for PCG D in bypass mode. This is the duration equal to one period of the DAI_MISCA4_I signal (PCG D) repeating at the beginning of every frame.
31:16 (R/W)	FSD	Pulse Width for Frame Sync PCG D. The `PCG_PW2.FSD` bit field sets the number of input clock periods for which the frame sync output is high for PCG D. Pulse width should be less than the divisor of the frame sync.
1 (R/W)	INVFSC	Active Low Frame Sync Select for Frame Sync in Bypass Mode. The `PCG_PW2.INVFSC` bit selects active low or active high frame sync for PCG C in bypass mode.
15:0 (R/W)	FSC	Pulse Width for Frame Sync PCG C. The `PCG_PW2.FSC` bit field sets the number of input clock periods for which the frame sync output is high for PCG C. Pulse width should be less than the divisor of the frame sync.

Table 35-13: PCG_PW2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	STROBEC	One Shot Frame Sync PCG C. The `PCG_PW2.STROBEC` bit sets the frame sync pulse for PCG C in bypass mode. This is the duration equal to one period of the DAI_MISCA3_I signal (PCG C) repeating at the beginning of every frame.

Precision Clock Frame Sync Synchronization 1 Register

The `PCG_SYNC1` register allows programs to synchronize the clock frame sync units with external frame syncs. Note that the `PCG_CTLA1.CLKSRC` bit is overridden if `PCG_SYNC1.CLKASRC` bit in the `PCG_SYNC1` register is set.

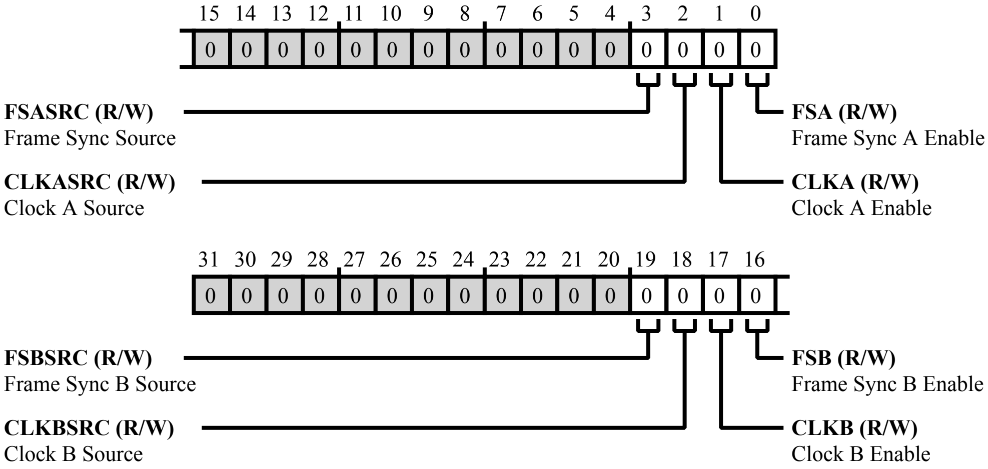

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSASRC (R/W)
Frame Sync Source
CLKASRC (R/W)
Clock A Source
FSA (R/W)
Frame Sync A Enable
CLKA (R/W)
Clock A Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSBSRC (R/W)
Frame Sync B Source
CLKBSRC (R/W)
Clock B Source
FSB (R/W)
Frame Sync B Enable
CLKB (R/W)
Clock B Enable

Figure 35-17: PCG_SYNC1 Register Diagram

Table 35-14: PCG_SYNC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	FSBSRC	Frame Sync B Source. The `PCG_SYNC1.FSBSRC` bit enables the frame sync B input source.	
		0	Output selected by FSBSOURCE bit
		1	Clock derived from core PLL selected for frame sync B
18 (R/W)	CLKBSRC	Clock B Source. The `PCG_SYNC1.CLKBSRC` bit enables the clock B input source.	
		0	Output selected by CLKBSOURCE bit
		1	Clock derived from core PLL selected for clock B
17 (R/W)	CLKB	Clock B Enable. The `PCG_SYNC1.CLKB` bit enables synchronization of clock B with the external frame sync.	
		0	Clock disabled
		1	Clock enabled

Table 35-14: PCG_SYNC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FSB	Frame Sync B Enable. The `PCG_SYNC1.FSB` bit enables synchronization of frame sync B with the external frame sync.	
		0	Frame sync disabled
		1	Frame sync enabled
3 (R/W)	FSASRC	Frame Sync Source. The `PCG_SYNC1.FSASRC` bit enables the frame sync A input source.	
		0	Output selected by FSASOURCE bit
		1	Clock derived from core PLL selected for frame sync A
2 (R/W)	CLKASRC	Clock A Source. The `PCG_SYNC1.CLKASRC` bit enables the clock A input source.	
		0	Output selected by CLKASOURCE bit
		1	Clock derived from core PLL selected for clock A
1 (R/W)	CLKA	Clock A Enable. The `PCG_SYNC1.CLKA` bit enables synchronization of clock A with the external frame sync.	
		0	Clock disabled
		1	Clock enabled
0 (R/W)	FSA	Frame Sync A Enable. The `PCG_SYNC1.FSA` bit enables synchronization of frame sync A with the external frame sync.	
		0	Frame Sync Disabled
		1	Frame Sync Enabled

Precision Clock Frame Sync Synchronization 2 Register

The `PCG_SYNC2` register allows programs to synchronize the clock frame sync units with external frame syncs. Note that the `PCG_CTLD1.CLKSRC` bit is overridden if `PCG_SYNC2.CLKDSRC` bit in the `PCG_SYNC2` register is set.

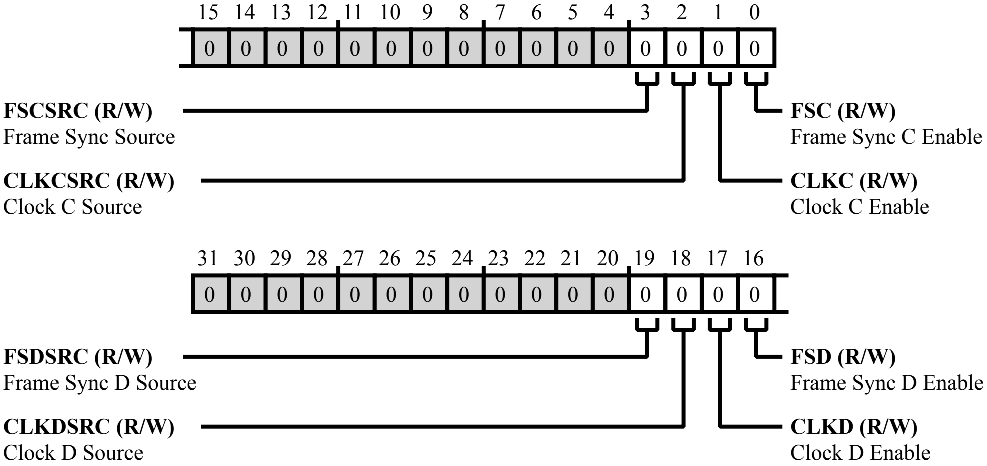

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSCSRC (R/W)
Frame Sync Source
CLKCSRC (R/W)
Clock C Source
FSC (R/W)
Frame Sync C Enable
CLKC (R/W)
Clock C Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSDSRC (R/W)
Frame Sync D Source
CLKDSRC (R/W)
Clock D Source
FSD (R/W)
Frame Sync D Enable
CLKD (R/W)
Clock D Enable

Figure 35-18: PCG_SYNC2 Register Diagram

Table 35-15: PCG_SYNC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	FSDSRC	Frame Sync D Source. The `PCG_SYNC2.FSDSRC` bit enables the frame sync D input source.	
		0	Output selected by FSDSOURCE bit
		1	Clock derived from core PLL selected for frame sync D
18 (R/W)	CLKDSRC	Clock D Source. The `PCG_SYNC2.CLKDSRC` bit enables the clock D input source.	
		0	Output selected by CLKDSOURCE bit
		1	Clock derived from core PLL selected for clock D
17 (R/W)	CLKD	Clock D Enable. The `PCG_SYNC2.CLKD` bit enables synchronization of clock D with the external frame sync.	
		0	Clock disabled
		1	Clock enabled

Table 35-15: PCG_SYNC2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FSD	Frame Sync D Enable. The `PCG_SYNC2.FSD` bit enables synchronization of frame sync D with the external frame sync.	
		0	Frame sync disabled
		1	Frame sync enabled
3 (R/W)	FSCSRC	Frame Sync Source. The `PCG_SYNC2.FSCSRC` bit enables the frame sync C input source.	
		0	Output selected by FSCSOURCE bit
		1	Clock derived from core PLL selected for frame sync C
2 (R/W)	CLKCSRC	Clock C Source. The `PCG_SYNC2.CLKCSRC` bit enables the clock C input source.	
		0	Output selected by CLKCSOURCE bit
		1	Clock derived from core PLL selected for clock C
1 (R/W)	CLKC	Clock C Enable. The `PCG_SYNC2.CLKC` bit enables synchronization of clock C with the external frame sync.	
		0	Clock disabled
		1	Clock enabled
0 (R/W)	FSC	Frame Sync C Enable. The `PCG_SYNC2.FSC` bit enables synchronization of frame sync C with the external frame sync.	
		0	Frame sync disabled
		1	Frame sync enabled

36 Asynchronous Sample Rate Converter (ASRC)

Sample rate converters (SRC) are frequently used in digital signal processing audio applications. The most frequently used sample rate conversions are off-loaded into hardware modules that are dedicated for filter processing and reduce the instruction processing load on the core, freeing it up for other tasks.

Features

The ASRC has these features and capabilities.

- 4 asynchronous stereo SRCs operating in slave mode are available in each DAI
- Simple programming model
- Controllable muting options (hardware, software and automatic)
- Automatically senses input and output sample frequencies
- Supports left-justified, I^2S, right-justified (16-,18-, 20-, 24-bits), and TDM serial port modes
- Daisy-chain configuration in TDM modes (including between DAI0 and DAI1) for input and output ports to create a serial frame
- Different protocols on input/output port allow format conversions
- De-emphasis filter for 32, 44.1 and 48 kHz sampling frequencies
- Up to 192 kHz sample rate input/output continuous sample ratios from 7.5:1 to 1:8
- Group delay (latency of interpolation filter) is 16 samples
- SNR from 128 to 140 dB (depending on processor model)
- Matched phase mode available to compensate for group delays
- Can be used to de-jitter clocks in systems

Functional Description

Conceptually, the sample rate converter interpolates the serial input data at a rate of 220 and samples the interpolated data stream by the output sample rate. In practice, a 64-tap FIR filter with 220 polyphases, a FIFO, a digital servo loop that measures the time difference between the input and output samples within 5 ps, and a digital circuit to track the sample rate ratio are used to perform the interpolation and output sampling.

ADSP-SC58x ASRC Register List

Sample Rate Converter Module

Table 36-1: ADSP-SC58x ASRC Register List

Name	Description
ASRC_CTL01	Control Register for ASRC 0 and 1
ASRC_CTL23	Control Register for ASRC 2 and 3
ASRC_MUTE	Mute Register
ASRC_RAT01	Ratio Register for ASRC 0 and 1
ASRC_RAT23	Ratio Register for ASRC 2 and 3

ASRC Interrupt List

The ASRC interrupts are controlled through the DAI.

Table 36-2: ASRC Interrupt List

Interrupt Name	Interrupt Condition	Return DAI Register	Return SEC Register	SEC ID
DAI0_IRQH	ASRC initialization	DAIx_IRPTL	SEC_ID	24
DAI1_IRQH	ASRC Sample rate change			25
DAI0_IRQL				145
DAI1_IRQL			146	

ASRC Block Diagram

The ***ASRC Block Diagram*** figure shows a top level block diagram of the ASRC module and the ***Core Architecture*** figure shows architecture details.

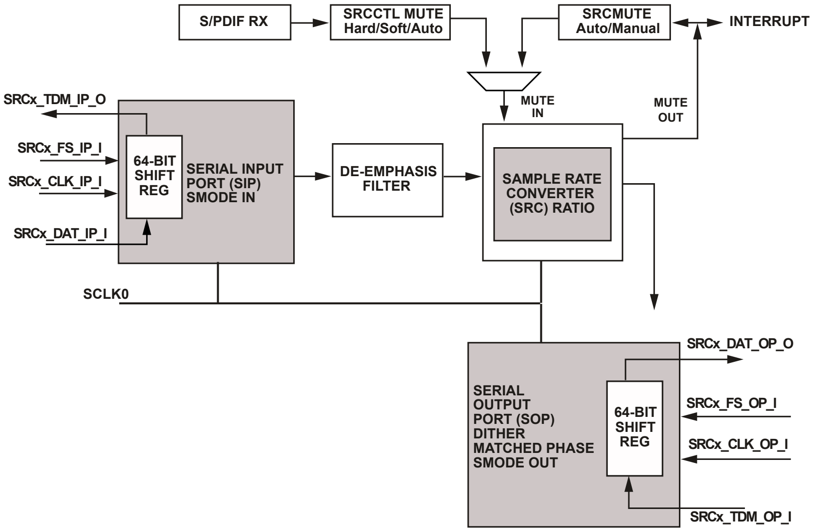

S/PDIF RX
SRCCTL MUTE Hard/Soft/Auto
SRCMUTE Auto/Manual
INTERRUPT
SRCx_TDM_IP_O
MUTE IN
MUTE OUT
SRCx_FS_IP_I
SRCx_CLK_IP_I
64-BIT SHIFT REG
SERIAL INPUT PORT (SIP) SMODE IN
DE-EMPHASIS FILTER
SAMPLE RATE CONVERTER (SRC) RATIO
SRCx_DAT_IP_I
SCLK0
SRCx_DAT_OP_O
SERIAL OUTPUT PORT (SOP) DITHER MATCHED PHASE SMODE OUT
64-BIT SHIFT REG
SRCx_FS_OP_I
SRCx_CLK_OP_I
SRCx_TDM_OP_I

Figure 36-1: ASRC Block Diagram

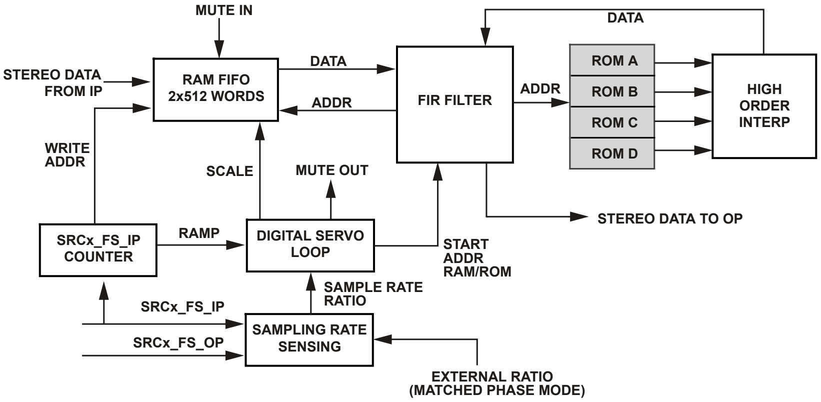

MUTE IN
DATA
STEREO DATA FROM IP
RAM FIFO 2x512 WORDS
ADDR
FIR FILTER
ADDR
ROM A
ROM B
ROM C
ROM D
HIGH ORDER INTERP
WRITE ADDR
SCALE
MUTE OUT
STEREO DATA TO OP
SRCx_FS_IP COUNTER
RAMP
DIGITAL SERVO LOOP
START ADDR RAM/ROM
SAMPLE RATE RATIO
SRCx_FS_IP
SRCx_FS_OP
SAMPLING RATE SENSING
EXTERNAL RATIO (MATCHED PHASE MODE)

Figure 36-2: ASRC Core Architecture

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the ASRCs to the output pins or any other peripherals. For more information, see the Digital Audio Interface (DAI) chapter.

Clocking

The ASRC module is in the SCLK0 clock domain. An internal divided version of the SCLK0 clock is generated and used as the fundamental clock for the ASRC module.

I/O Ports

The I/O ports provide the interface through which data is transferred asynchronously into and out of the SRC modules. The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I^2S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support TDM mode for daisy-chaining multiple SRCs to form a frame. The serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected.

NOTE: The SRC converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

De-Emphasis Filter

The de-emphasis filter is used to de-emphasize audio data that has been emphasized.

Mute Control

When either the SRC starts up (or there is a change in sample ratio), the mute out signal (`SRCx_MUTEOUT`) is asserted (=1). The mute out signal stays high until the SRC settles on the new sample rates. While mute out is asserted high, the mute in signal should be asserted high as well. The mute in signal performs a soft mute of the audio input data when asserted and un-mutes the input audio data softly when deasserted.

Note that it takes 4096 input port FS samples until the audio input data is completely muted and 4096 FS samples until the audio input data is completely unmuted.

SRC Core

As shown in the ***ASRC Core Architecture*** figure, the sample rate converter's RAM FIFO block adjusts the left and right input samples and stores them for the FIR filter's convolution cycle. The `ASRCx_FS_IP` counter provides the write address (for scaling) to the FIFO block and the ramp input to the digital-servo loop. The ROM stores the coefficients for the FIR filter convolution and performs a high-order interpolation between the stored coefficients. The sample rate ratio block measures the sample rate by dynamically altering the ROM coefficients and scaling the FIR filter length and input data. The digital-servo loop automatically tracks the `SRCx_FS_IP` and `SRCx_FS_OP` sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.

NOTE: Unlike other peripherals, the sample rate converters own local memories (RAM and ROM) which are dedicated for the purpose of sample rate conversion only.

The sample rate converter only operates asynchronously and is always a slave to the input and output ports.

RAM FIFO

The RAM FIFO receives the left and right input data and adjusts the amplitude of the data for both the soft muting of the SRC and the scaling of the input data by the sample rate ratio before storing the samples in RAM. The input data is scaled by the sample rate ratio because as the FIR filter length of the convolution increases, so does the amplitude of the convolution output. To keep the output of the FIR filter from saturating, the input data is scaled down by multiplying it by (`SRCx_FS_OP`)/(`SRCx_FS_IP`) when `SRCx_FS_OP<SRCx_FS_IP`. The FIFO also scales the input data to mute and stop muting the SRC.

Digital Servo Loop

The digital-servo loop is essentially a ramp filter that provides the initial pointer to the address in RAM and ROM for the start of the FIR convolution. The RAM pointer is the integer output of the ramp filter while the ROM pointer is the fractional part. The digital-servo loop must be able to provide excellent rejection of jitter on the `SRCx_FS_IP` and `SRCx_FS_OP` clocks as well as measure the arrival of the `SRCx_FS_OP` clock within 5 ps. The digital-servo loop also divides the fractional part of the ramp output by the ratio of (`SRCx_FS_IP`)/(`SRCx_FS_OP`) for the case when `SRCx_FS_IP`> `SRCx_FS_OP`, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample rate, a fast mode has been added to the filter. When the digital-servo loop starts up or the sample rate is changed, the digital-servo loop kicks into fast mode to adjust and settle on the new sample rate. Upon sensing the digital-servo loop settling down to some reasonable value, the digital-servo loop kicks into normal or slow mode. During fast mode, the `SRCx_MUTE_OUT` bit of the ASRC is asserted to mute the ASRC input which avoids clicks and pops.

FIR Filter

The FIR filter is a 64-tap filter in the case of `SRCx_FS_OP<SRCx_FS_IP` and is (`SRCx_FS_IP`)/(`SRCx_FS_OP`) × 64 taps for the case when `SRCx_FS_IP>SRCx_FS_OP`. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the `SRCx_FS_OP` period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the (`SRCx_FS_OP/SRCx_FS_IP`) × 2^{20} ratio for `SRCx_FS_IP>SRCx_FS_OP` or 2^{20} for `SRCx_FS_OP <SRCx_FS_IP`. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

Sample Rate Sensing

The (`SRCx_FS_IP`)/(`SRCx_FS_OP`) sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when `SRCx_FS_IP>SRCx_FS_OP`. The ratio is calculated by comparing the output of an `SRCx_FS_OP` counter to the output of an `SRCx_FS_IP` counter. If `ASRCx_FS_OP>SRCx_FS_IP`, the ratio is held at one. If `SRCx_FS_IP>SRCx_FS_OP`, the sample rate ratio is updated if it is different by more than two `SRCx_FS_OP` periods from the previous `SRCx_FS_OP` to `SRCx_FS_IP` comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

Digital Filter Group Delay

The RAM in the FIFO is 512 words deep for both left and right channels. An offset of 16 samples to the write address, provided by the `SRCx_FS_IP` counter, is added to prevent the RAM read pointer from overlapping the write address. The maximum decimation rate can be calculated from the RAM word: depth = (512 – 16) ÷ 64 taps = 7.5:1.

The 64 samples effect latency in the interpolation filter. This latency (group delay) depends on interpolation or decimation ratio and is determined as follows:

Interpolation or Decimation Ratio (1): GDL = $16/f_{S\_IN} + 32/f_{S\_IN}$ seconds for `SRC_FS_OP` > `SRC_FS_IP`

Interpolation or Decimation Ratio (2): GDL = $16/f_{S\_IN} + 32/f_{S\_IN} \times f_{S\_IN}/f_{S\_OUT}$ seconds for `SRC_FS_OP` < `SRC_FS_IP`

Data Format

The ***ASRC Data Frame Format by Protocol*** figure shows the data input format for a frame (stereo data). The frame format is valid for all protocols. For models which do not support matched phase mode the 8-bit data field is ignored.

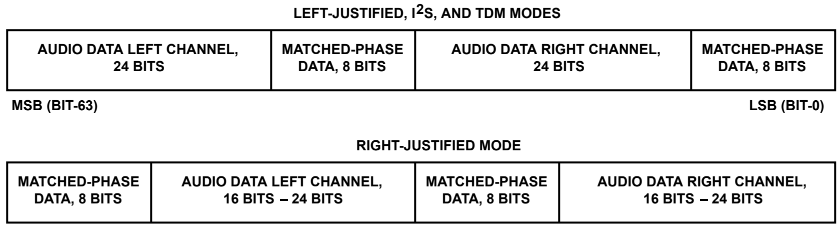

LEFT-JUSTIFIED, I2S, AND TDM MODES
AUDIO DATA LEFT CHANNEL, 24 BITS
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA RIGHT CHANNEL, 24 BITS
MATCHED-PHASE DATA, 8 BITS
MSB (BIT-63)
LSB (BIT-0)
RIGHT-JUSTIFIED MODE
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA LEFT CHANNEL, 16 BITS – 24 BITS
MATCHED-PHASE DATA, 8 BITS
AUDIO DATA RIGHT CHANNEL, 16 BITS – 24 BITS

Figure 36-3: ASRC Data Frame Format by Protocol

Operating Modes

The ASRC can operate in TDM, I^2S, left-justified, right-justified, and bypass modes. The serial ports of the processor can be used for moving the ASRC data to/from the internal memory.

In I^2S, left-justified and right-justified modes, the ASRCs operate individually. The serial data provided in the input port is converted to the sample rate of the output port.

TDM Input Mode

In TDM input port, several ASRCs can be daisy-chained together and connected to the serial input port of a SHARC processor or other processor (see the ***TDM Input/Output Modes*** figure). The ASRC IP contains a 64-bit parallel load shift register. When the `SRCx_FS_IP_I` pulse arrives, each ASRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to `SRCx_DATA_IP_I`, while the output is connected to `SRCx_TDM_IP_O`. By connecting the `SRCx_TDM_IP_O` to the `SRCx_DATA_IP_I` of the next ASRC, a large shift register is created, which is clocked by `SRCx_CLK_IP_I`.

NOTE: The number of ASRCs that can be daisy-chained together is limited by the maximum frequency of `SRCx_CLK_xx_I`, refer to the data sheet for exact values. For example, if the maximum frequency of `SRCx_CLK_xx_I` is *x* MHz, and the output sample rate is *fS*, then number of ASRCs (n) that can be connected in daisy chained fashion is: n 64 FS $\leq x$ MHz.

TDM Output Mode

As shown in the ***TDM Input/Output Modes*** figure, using the TDM output port several ASRCs can be daisy-chained together and connected to the SPORT of this or another processor. The ASRC OP contains a 64-bit parallel load shift register. When the `SRCx_FS_OP_I` pulse arrives, each ASRC loads its left and right data into the 64-bit shift register. The input to the shift register is connected to `SRCx_TDM_OP_I`, and the output is connected to

`SRCx_DAT_OP_O`. By connecting the `SRCx_DAT_OP_O` to the `SRCx_TDM_OP_I` of the next ASRC, a large shift register is created, which is clocked by `SRCx_CLK_OP_I`.

As shown in ***TDM Input/Output Modes***, with three ASRCs in a daisy-chain connection, the serial clock for input/or output port is defined as: SCLK = 3 × 64 FS = 192 FS.

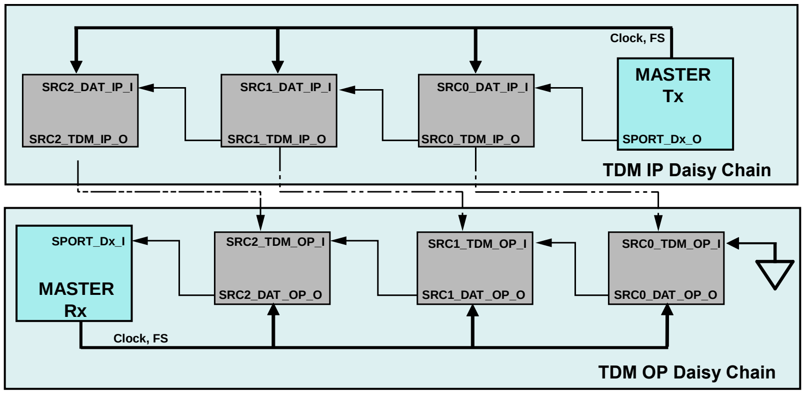

Clock, FS
MASTER Tx
SRC2_DAT_IP_I
SRC2_TDM_IP_O
SRC1_DAT_IP_I
SRC1_TDM_IP_O
SRC0_DAT_IP_I
SRC0_TDM_IP_O
SPORT_Dx_O
TDM IP Daisy Chain
SPORT_Dx_I
MASTER Rx
SRC2_TDM_OP_I
SRC2_DAT_OP_O
SRC1_TDM_OP_I
SRC1_DAT_OP_O
SRC0_TDM_OP_I
SRC0_DAT_OP_O
Clock, FS
TDM OP Daisy Chain

Figure 36-4: TDM Input/Output Modes

Matched-Phase Mode

The matched-phase mode of the sample rate converter, shown in ***Typical Configuration for Matched-Phase Mode Operation***, is enabled by the `ASRC_CTL01.MPHASE0`, `ASRC_CTL01.MPHASE1`, `ASRC_CTL23.MPHASE2` and `ASRC_CTL23.MPHASE3` bits. This mode is used to match the phase (group delay) between two or more adjacent sample rate converters that are operating with the same input and output clocks.

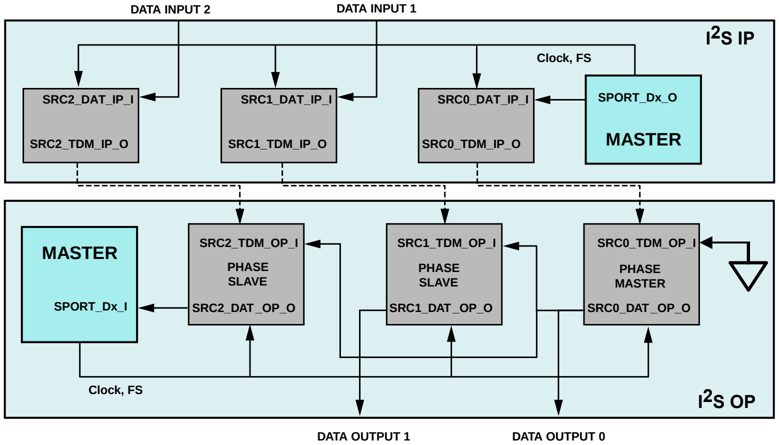

DATA INPUT 2
DATA INPUT 1
I²S IP
Clock, FS
SRC2_DAT_IP_I
SRC2_TDM_IP_O
SRC1_DAT_IP_I
SRC1_TDM_IP_O
SRC0_DAT_IP_I
SRC0_TDM_IP_O
SPORT_Dx_O
MASTER
MASTER
SPORT_Dx_I
SRC2_TDM_OP_I
PHASE SLAVE
SRC2_DAT_OP_O
SRC1_TDM_OP_I
PHASE SLAVE
SRC1_DAT_OP_O
SRC0_TDM_OP_I
PHASE MASTER
SRC0_DAT_OP_O
Clock, FS
I²S OP
DATA OUTPUT 1
DATA OUTPUT 0

Figure 36-5: Typical Configuration for Matched-Phase Mode Operation

Hysteresis of the (`SRCx_FS_OP`)/(`SRCx_FS_IP`) ratio circuit can cause phase mismatching between two ASRCs operating with the same input and output clocks. Since the hysteresis requires a difference of more than two `SRCx_FS_OP` periods to update the `SRCx_FS_OP` and `SRCx_FS_IP` ratios, two ASRCs may have differences in their ratios from 0 to 4 `SRCx_FS_OP` period counts. The (`SRCx_FS_OP`)/(`SRCx_FS_IP`) ratio adjusts the filter length of the ASRC, which corresponds directly with the group delay. Thus, the magnitude in the phase difference depends upon the resolution of the `SRCx_FS_OP` and `SRCx_FS_IP` counters. The greater the resolution of the counters, the smaller the phase difference error.

When the slave SRC matched-phase mode bit is set (=1), it accepts the sample rate ratio transmitted by another SRC, (the matched-phase master) which has its matched-phase mode bit cleared (=0), through its serial output.

The phase master ASRC device transmits its `SRCx_FS_OP`/`SRCx_FS_IP` ratio through the data output pin (`SRCx_DAT_OP_O`) to the slave's ASRC's data input pins (`SRCx_TDM_OP_I`). The transmitted data (32-bit subframe) contains 24-bit data and 8-bits matched phase (see the ***ASRC Data Frame Format by Protocol*** figure).

The slave SRCs receive the 8-bit matched phase bits (instead of their own internally-derived ratio) if their `SRCx_MPHASE` bits are set to 1, respectively. The `SRCx_FS_IP` and `SRCx_FS_OP` signals may be asynchronous with respect to each other in this mode. Note that there must be 64 `SRCx_CLK_OP` cycles per frame in matched-phase mode (two 24-bits data and two 8-bits phase match).

NOTE: By default, matched phased data is sent on the `SRCx_DAT_OP_O` pin, but only if the `SRCx_TDM_OP_I` pin is tied low. The slaves simply ignore the matched phased data if their `ASRC_CTL01.MPHASE1` through `ASRC_CTL23.MPHASE3` bits are cleared (= 0).

Bypass Mode

When the `ASRC_CTL01.BYP0`, `ASRC_CTL01.BYP1`, `ASRC_CTL23.BYP2` and `ASRC_CTL23.BYP3` bits are set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. Dithering is disabled. This mode is ideal when the input and output sample rates are the same and the `SRCx_FS_IP_I` and `SRCx_FS_OP_I` signals are synchronous with respect to each other. In matched phase bypass mode, the `SRCx_FS_OP_I` signal should come at least one `SRCx_CLK_xx_I` period before `SRCx_FS_IP_I`. Cases where this is not met could result in data loss. For example, if internal SPORTS are used then the `SRCx_FS_OP_I` and `SRCx_FS_IP_I` signals could be driven by different SPORTs so that the timing of these signals can be controlled by enabling them at different times. This mode can also be used for passing through non-audio data since no processing is performed on the input data.

De-Emphasis Mode

The `ASRC_CTL01.DEEMPHASIS0`, `ASRC_CTL01.DEEMPHASIS1`, `ASRC_CTL23.DEEMPHASIS2` and `ASRC_CTL23.DEEMPHASIS3` bits choose the type of de-emphasis filter based on the input sample rate for 32, 44.1 or 48 kHz sampling rates.

Dithering Mode

The `ASRC_CTL01.DITHER0`, `ASRC_CTL01.DITHER1`, `ASRC_CTL23.DITHER2`, and `ASRC_CTL23.DITHER3` control this mode of operation. Serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected. In the case of 20-, 18- and 16-bit word lengths, the least significant bits of the 24-bit word coming from the SRC into the serial output port are truncated. The `DITHER_EN` signal (not user configurable) automatically adds dithering to the 24-bit word before truncating to the appropriate output word length. The `21BIT_DITHER` signal is used for the consumer version of the SRC to reduce the dynamic range performance to approximately 128 dB.

NOTE: The ASRC can be programmed to add the triangular Probability Distribution Function (PDF) dither to the digital audio samples. It is advisable to add dither when the input word width exceeds the output word width, for example the input word is 20 bits and the output word is 16 bits. Triangular PDF is generally considered to create the most favorable noise shaping of the residual quantization noise.

Muting Modes

The mute feature of the ASRC can be controlled automatically in hardware using the `MUTE_IN` signal by connecting it to the `MUTE_OUT` signal. Automatic muting can be disabled by setting (=1) the `ASRC_MUTE.MUTE0` through `ASRC_MUTE.MUTE3` bits.

NOTE: Note that by default, the `ASRC_MUTE` register connects the `MUTE_IN` signal to the `MUTE_OUT` signal, but not conversely.

Soft Mute

When the `ASRC_CTL01.SOFTMUTE0`, `ASRC_CTL01.SOFTMUTE1`, `ASRC_CTL23.SOFTMUTE2` and `ASRC_CTL23.SOFTMUTE3` bits are set, the `MUTE_IN` signal is asserted, and the ASRC performs a soft mute by linearly decreasing the input data to the ASRC FIFO to zero, (-144 dB) attenuation as described for automatic hardware muting.

A 12-bit counter, clocked by `SRCx_FS_IP_I`, is used to control the mute attenuation. Therefore, the time it takes from the assertion of the `MUTE_IN` signal to -144 dB, full mute attenuation is 4096 FS cycles. Likewise, the time it takes to reach 0 dB mute attenuation from the deassertion of the `MUTE_IN` signal is 4096 FS cycles.

Hard Mute

When the `ASRC_CTL01.HARDMUTE0`, `ASRC_CTL01.HARDMUTE1`, `ASRC_CTL23.HARDMUTE2` and `ASRC_CTL23.HARDMUTE3` bits are set, the ASRC immediately mutes the input data to the ASRC FIFO to zero, (-144 dB) attenuation.

Auto Mute

When the `ASRC_CTL01.AUTOMUTE0`, `ASRC_CTL01.AUTOMUTE1`, `ASRC_CTL23.AUTOMUTE2` and `ASRC_CTL23.AUTOMUTE3` bits are set, the ASRC communicates with the S/PDIF receiver peripheral to determine when the input should mute.

This mode is useful for automatic detection of non-PCM audio data received from the S/PDIF receiver.

Interrupts

The following sections provide information about interrupt sources, masking and servicing.

Sources

Each ASRC module drives one interrupt signal (mute out asserted). All these signals are connected into the DAI_IRPTL_H or DAI_IRPTL_L latch registers. The ASRC ports generate interrupts as described below.

SRC Mute Out

The SRC mute out signal can be used to generate interrupts on their rising edge, falling edge, or both, depending on how the DAI interrupt mask registers (DAI_IMSK_RE/DAI_IMSK_FE) are programmed. This programming allows the generation of DAI_IRPTL_H/DAI_IRPTL_L interrupts either entering mute, exiting muting or both. The SRCx_MUTE_OUT interrupt is generated only once when the SRC is locked (after 4096 FS input samples) and after changes to the sample ratio. Hard mute, soft mute, and auto mute only control the muting of the input data to the SRC.

Masking

The DAI_IMSK_FE, DAI_IMSK_RE, and DAI_IMSK_PRI registers must be unmasked accordingly. The DAI_IRQH and DAI_IRQL signals are routed to the system event controller (SEC) and general interrupt controller (GIC).

Service

The ISR reads the DAI_IRPTL_H and DAI_IRPTL_L registers to clear the interrupt request.

Programming Model

The following is basic information on programming the ASRC module.

1. Program the ASRC_CTL01 and ASRC_CTL23 registers and keep the ASRC_CTL01.EN0 through ASRC_CTL23.EN3 bits cleared.
2. Set the ASRC_CTL01.EN0 through ASRC_CTL23.EN3 bits. After 4096 input port FS cycles the ASRC has un-muted.

Debug Features

The asynchronous sample rate converter allow the bypass mode. When the ASRC_CTL01.BYP0 through ASRC_CTL23.BYP3 bits are set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. This mode can be used for testing both ports when the input and output sample rates are at the same frequency, therefore both input and output ports can be routed to the same serial clock and frame sync.

ADSP-SC58x ASRC Register Descriptions

Sample Rate Converter Module (ASRC) contains the following registers.

Table 36-3: ADSP-SC58x ASRC Register List

Name	Description
ASRC_CTL01	Control Register for ASRC 0 and 1
ASRC_CTL23	Control Register for ASRC 2 and 3
ASRC_MUTE	Mute Register
ASRC_RAT01	Ratio Register for ASRC 0 and 1
ASRC_RAT23	Ratio Register for ASRC 2 and 3

Control Register for ASRC 0 and 1

The `ASRC_CTL01` register (read/write) controls the operating modes, filters, and data formats used in the ASRC modules 0 and 1.

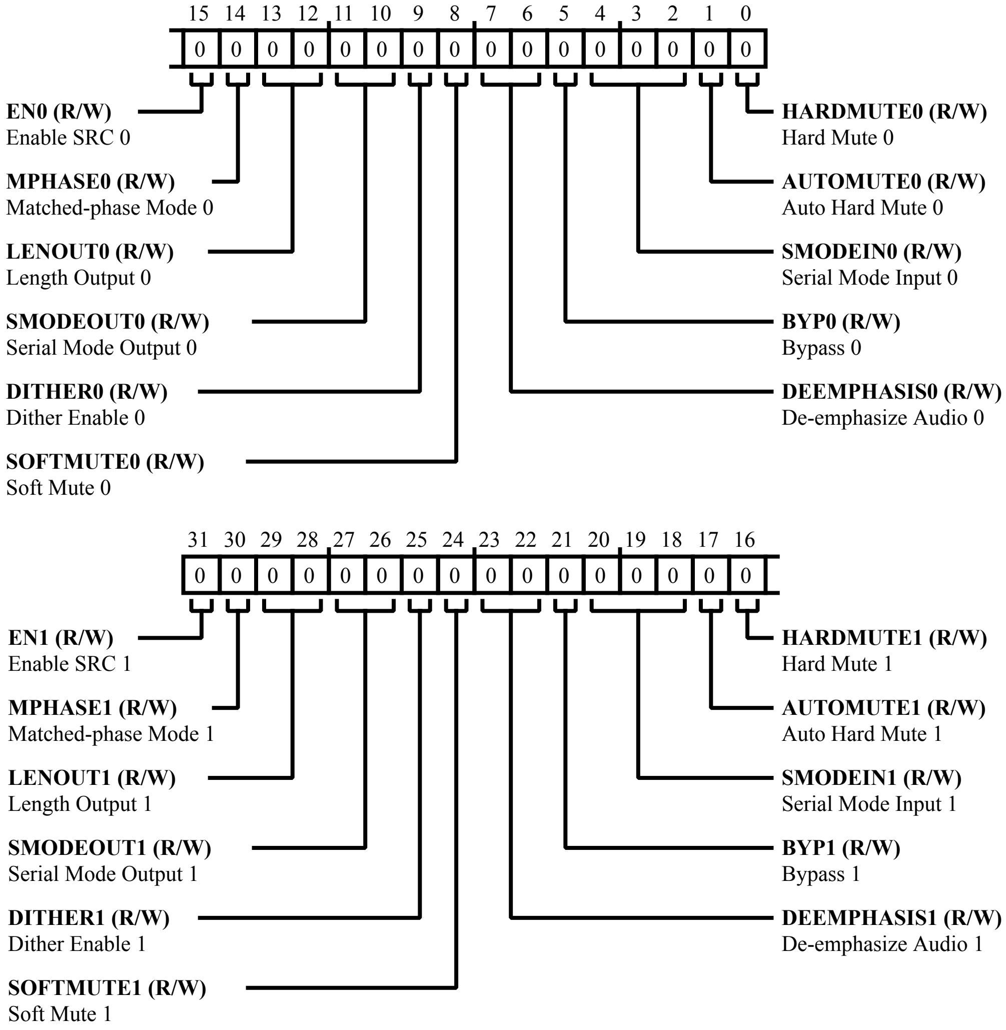

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN0 (R/W)
Enable SRC 0
MPHASE0 (R/W)
Matched-phase Mode 0
LENOUT0 (R/W)
Length Output 0
SMODEOUT0 (R/W)
Serial Mode Output 0
DITHER0 (R/W)
Dither Enable 0
SOFTMUTE0 (R/W)
Soft Mute 0
HARDMUTE0 (R/W)
Hard Mute 0
AUTOMUTE0 (R/W)
Auto Hard Mute 0
SMODEIN0 (R/W)
Serial Mode Input 0
BYP0 (R/W)
Bypass 0
DEEMPHASIS0 (R/W)
De-emphasize Audio 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN1 (R/W)
Enable SRC 1
MPHASE1 (R/W)
Matched-phase Mode 1
LENOUT1 (R/W)
Length Output 1
SMODEOUT1 (R/W)
Serial Mode Output 1
DITHER1 (R/W)
Dither Enable 1
SOFTMUTE1 (R/W)
Soft Mute 1
HARDMUTE1 (R/W)
Hard Mute 1
AUTOMUTE1 (R/W)
Auto Hard Mute 1
SMODEIN1 (R/W)
Serial Mode Input 1
BYP1 (R/W)
Bypass 1
DEEMPHASIS1 (R/W)
De-emphasize Audio 1

Figure 36-6: ASRC_CTL01 Register Diagram

Table 36-4: ASRC_CTL01 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EN1	Enable SRC 1. The `ASRC_CTL01.EN1` bit enables SRC 1. When (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB). Note that SRC power-up completion is finished by clearing the `ASRC_RAT01.MUTEOUT1` bit. Writes to the `ASRC_CTL01` register should be at least one cycle before setting the `ASRC_CTL01.EN1` bit. When setting and clearing this bit, it should be held low for a minimum of 5 SCLK0_0 cycles.	
30 (R/W)	MPHASE1	Matched-phase Mode 1. The `ASRC_CTL01.MPHASE1` bit configures SRC1 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.	
		0	Matched phase slave disabled
		1	Matched phase slave enabled
29:28 (R/W)	LENOUT1	Length Output 1. The `ASRC_CTL01.LENOUT1` bit field selects the serial output word length on SRC1.	
		0	24 bits
		1	20 bits
		2	18 bits
		3	16 bits
27:26 (R/W)	SMODEOUT1	Serial Mode Output 1. The `ASRC_CTL01.SMODEOUT1` bit field selects the serial output format on SRC1.	
		0	Left-justified
		1	I2S
		2	TDM
		3	Right-justified
25 (R/W)	DITHER1	Dither Enable 1. The `ASRC_CTL01.DITHER1` bit enables dithering before truncation on SRC1 when a word length less than 24 bits is selected.	
		0	Truncation only
		1	Dithering before truncation

Table 36-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	SOFTMUTE1	Soft Mute 1. The `ASRC_CTL01.SOFTMUTE1` bit enables soft mute on SRC1.	
		0	Unmute
		1	Mute
23:22 (R/W)	DEEMPHASIS1	De-emphasize Audio 1. The `ASRC_CTL01.DEEMPHASIS1` bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).	
		0	No de-emphasis
		1	32 kHz
		2	44.1 kHz
		3	48 kHz
21 (R/W)	BYP1	Bypass 1. The `ASRC_CTL01.BYP1` bit makes the output of SRC1 the same as the input.	
20:18 (R/W)	SMODEIN1	Serial Mode Input 1. The `ASRC_CTL01.SMODEIN1` bit field selects the serial input format for SRC1.	
		0	left-justified
		1	I2S
		2	TDM
		4	24-bit right-justified
		5	20-bit right-justified
		6	18-bit right-justified
		7	16-bit right-justified
17 (R/W)	AUTOMUTE1	Auto Hard Mute 1. The `ASRC_CTL01.AUTOMUTE1` bit auto hard mutes SRC1 when non audio is asserted by the SPDIF receiver.	
		0	Unmute
		1	Mute
16 (R/W)	HARDMUTE1	Hard Mute 1. The `ASRC_CTL01.HARDMUTE1` bit hard mutes SRC1.	
		0	Unmute
		1	Mute

Table 36-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EN0	Enable SRC 0. The `ASRC_CTL01.EN0` bit enables SRC 0. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB). Note that SRC power-up completion is finished by clearing the `ASRC_RAT01.MUTEOUT0` bit. Writes to the `ASRC_CTL01` register should be at least one cycle before setting the `ASRC_CTL01.EN0` bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.	
14 (R/W)	MPHASE0	Matched-phase Mode 0. The `ASRC_CTL01.MPHASE0` bit configures SRC0 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.	
		0	Matched phase slave disabled
		1	Matched phase slave enabled
13:12 (R/W)	LENOUT0	Length Output 0. The `ASRC_CTL01.LENOUT0` bit field selects the serial output word length on SRC0.	
		0	24 bits
		1	20 bits
		2	18 bits
		3	16 bits
11:10 (R/W)	SMODEOUT0	Serial Mode Output 0. The `ASRC_CTL01.SMODEOUT0` bit field selects the serial output format on SRC0.	
		0	Left-justified
		1	I2S
		2	TDM
		3	Right-justified
9 (R/W)	DITHER0	Dither Enable 0. The `ASRC_CTL01.DITHER0` bit enables dithering before truncation on SRC0 when a word length less than 24 bits is selected.	
		0	Truncation only
		1	Dithering before truncation

Table 36-4: ASRC_CTL01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	SOFTMUTE0	Soft Mute 0. The `ASRC_CTL01.SOFTMUTE0` bit enables soft mute on SRC0.	
		0	Unmute
		1	Mute
7:6 (R/W)	DEEMPHASIS0	De-emphasize Audio 0. The `ASRC_CTL01.DEEMPHASIS0` bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).	
		0	No de-emphasis
		1	32 kHz
		2	44.1 kHz
		3	48 kHz
5 (R/W)	BYP0	Bypass 0. The `ASRC_CTL01.BYP0` bit makes the output of SRC0 the same as the input.	
4:2 (R/W)	SMODEIN0	Serial Mode Input 0. The `ASRC_CTL01.SMODEIN0` bit field selects the serial input format for SRC0.	
		0	left-justified
		1	I2S
		2	TDM
		4	24-bit right-justified
		5	20-bit right-justified
		6	18-bit right-justified
		7	16-bit right-justified
1 (R/W)	AUTOMUTE0	Auto Hard Mute 0. The `ASRC_CTL01.AUTOMUTE0` bit auto hard mutes SRC0 when non audio is asserted by the SPDIF receiver.	
		0	Unmute
		1	Mute
0 (R/W)	HARDMUTE0	Hard Mute 0. The `ASRC_CTL01.HARDMUTE0` bit hard mutes SRC0.	
		0	Unmute
		1	Mute (default)

Control Register for ASRC 2 and 3

The ASRC_CTL23 register (read/write) controls the operating modes, filters, and data formats used in the sample rate converter modules 2 and 3.

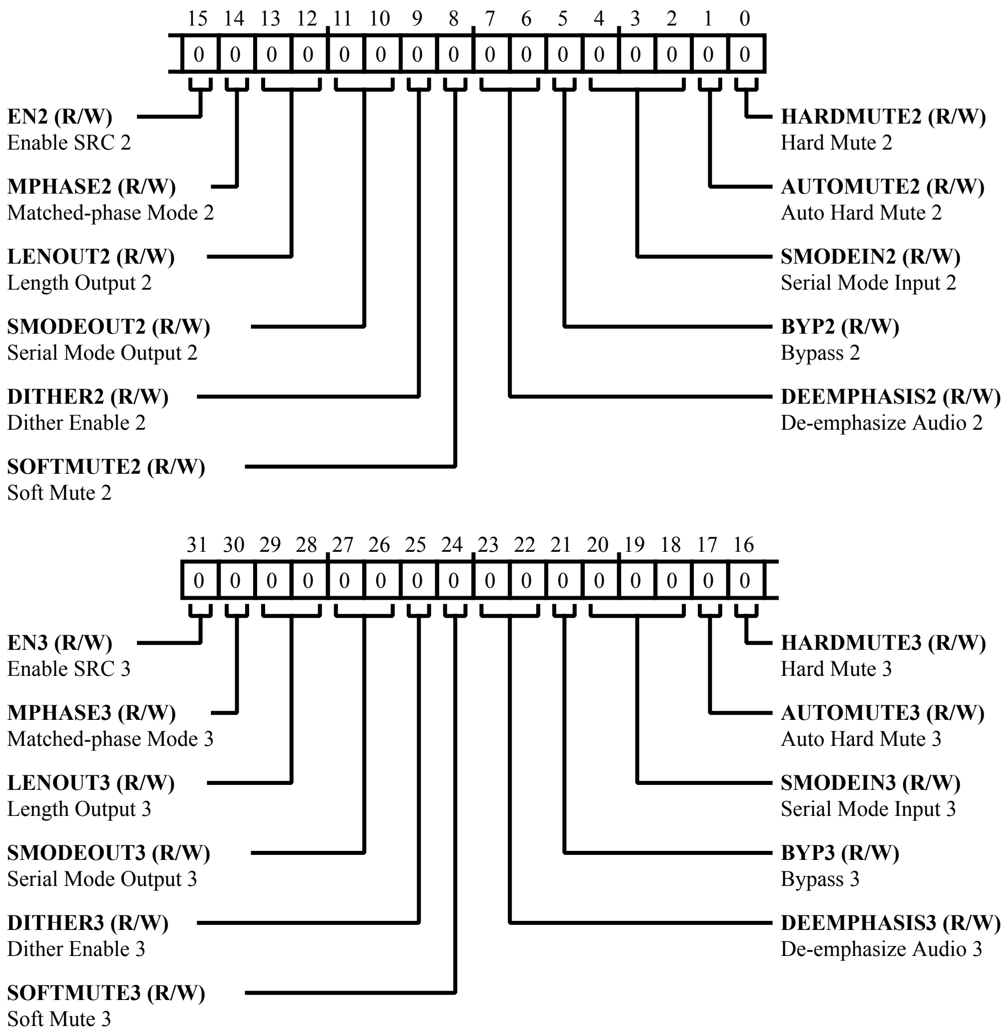

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN2 (R/W)
Enable SRC 2
MPHASE2 (R/W)
Matched-phase Mode 2
LENOUT2 (R/W)
Length Output 2
SMODEOUT2 (R/W)
Serial Mode Output 2
DITHER2 (R/W)
Dither Enable 2
SOFTMUTE2 (R/W)
Soft Mute 2
HARDMUTE2 (R/W)
Hard Mute 2
AUTOMUTE2 (R/W)
Auto Hard Mute 2
SMODEIN2 (R/W)
Serial Mode Input 2
BYP2 (R/W)
Bypass 2
DEEMPHASIS2 (R/W)
De-emphasize Audio 2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN3 (R/W)
Enable SRC 3
MPHASE3 (R/W)
Matched-phase Mode 3
LENOUT3 (R/W)
Length Output 3
SMODEOUT3 (R/W)
Serial Mode Output 3
DITHER3 (R/W)
Dither Enable 3
SOFTMUTE3 (R/W)
Soft Mute 3
HARDMUTE3 (R/W)
Hard Mute 3
AUTOMUTE3 (R/W)
Auto Hard Mute 3
SMODEIN3 (R/W)
Serial Mode Input 3
BYP3 (R/W)
Bypass 3
DEEMPHASIS3 (R/W)
De-emphasize Audio 3

Figure 36-7: ASRC_CTL23 Register Diagram

Table 36-5: ASRC_CTL23 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EN3	Enable SRC 3. The `ASRC_CTL23.EN3` bit enables SRC 3. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB). Note that SRC power-up completion is finished by clearing the `ASRC_RAT23.MUTEOUT3` bit. Writes to the `ASRC_CTL23` register should be at least one cycle before setting the `ASRC_CTL23.EN3` bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.	
30 (R/W)	MPHASE3	Matched-phase Mode 3. The `ASRC_CTL23.MPHASE3` bit configures SRC3 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.	
		0	Matched phase slave disabled
		1	Matched phase slave enabled
29:28 (R/W)	LENOUT3	Length Output 3. The `ASRC_CTL23.LENOUT3` bit field selects the serial output word length on SRC3.	
		0	24 bits
		1	20 bits
		2	18 bits
		3	16 bits
27:26 (R/W)	SMODEOUT3	Serial Mode Output 3. The `ASRC_CTL23.SMODEOUT3` bit field selects the serial output format on SRC3.	
		0	Left-justified
		1	I2S
		2	TDM
		3	Right-justified
25 (R/W)	DITHER3	Dither Enable 3. The `ASRC_CTL23.DITHER3` bit enables dithering before truncation on SRC3 when a word length less than 24 bits is selected.	
		0	Truncation only
		1	Dithering before truncation

Table 36-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	SOFTMUTE3	Soft Mute 3. The `ASRC_CTL23.SOFTMUTE3` bit enables soft mute on SRC3.	
		0	Unmute
		1	Mute
23:22 (R/W)	DEEMPHASIS3	De-emphasize Audio 3. The `ASRC_CTL23.DEEMPHASIS3` bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).	
		0	No de-emphasis
		1	32 kHz
		2	44.1 kHz
		3	48 kHz
21 (R/W)	BYP3	Bypass 3. The `ASRC_CTL23.BYP3` bit makes the output of SRC3 the same as the input.	
20:18 (R/W)	SMODEIN3	Serial Mode Input 3. The `ASRC_CTL23.SMODEIN3` bit field selects the serial input format for SRC3.	
		0	left-justified
		1	I2S
		2	TDM
		4	24-bit right-justified
		5	20-bit right-justified
		6	18-bit right-justified
		7	16-bit right-justified
17 (R/W)	AUTOMUTE3	Auto Hard Mute 3. The `ASRC_CTL23.AUTOMUTE3` bit auto hard mutes SRC3 when non audio is asserted by the SPDIF receiver.	
		0	Unmute
		1	Mute
16 (R/W)	HARDMUTE3	Hard Mute 3. The `ASRC_CTL23.HARDMUTE3` bit hard mutes SRC3.	
		0	Unmute
		1	Mute

Table 36-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EN2	Enable SRC 2. The `ASRC_CTL23.EN2` bit enables SRC 2. When (set =1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (144 dB). Note that SRC power-up completion is finished by clearing the `ASRC_RAT23.MUTEOUT2` bit. Writes to the `ASRC_CTL23` register should be at least one cycle before setting the `ASRC_CTL23.EN2` bit. When setting and clearing this bit, it should be held low for a minimum of 5 CLK cycles.	
14 (R/W)	MPHASE2	Matched-phase Mode 2. The `ASRC_CTL23.MPHASE2` bit configures SRC2 to not use its own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data.	
		0	Matched phase slave disabled
		1	Matched phase slave enabled
13:12 (R/W)	LENOUT2	Length Output 2. The `ASRC_CTL23.LENOUT2` bit field selects the serial output word length on SRC2.	
		0	24 bits
		1	20 bits
		2	18 bits
		3	16 bits
11:10 (R/W)	SMODEOUT2	Serial Mode Output 2. The `ASRC_CTL23.SMODEOUT2` bit field selects the serial output format on SRC2.	
		0	Left-justified
		1	I2S
		2	TDM
		3	Right-justified
9 (R/W)	DITHER2	Dither Enable 2. The `ASRC_CTL23.DITHER2` bit enables dithering before truncation on SRC2 when a word length less than 24 bits is selected.	
		0	Truncation only
		1	Dithering before truncation

Table 36-5: ASRC_CTL23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	SOFTMUTE2	Soft Mute 2. The `ASRC_CTL23.SOFTMUTE2` bit enables soft mute on SRC2.	
		0	Unmute
		1	Mute
7:6 (R/W)	DEEMPHASIS2	De-emphasize Audio 2. The `ASRC_CTL23.DEEMPHASIS2` bits are used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is based on the input sample rate (SRCx_FS_IP_I signal).	
		0	No de-emphasis
		1	32 kHz
		2	44.1 kHz
		3	48 kHz
5 (R/W)	BYP2	Bypass 2. The `ASRC_CTL23.BYP2` bit makes the output of SRC2 the same as the input.	
4:2 (R/W)	SMODEIN2	Serial Mode Input 2. The `ASRC_CTL23.SMODEIN2` bit field selects the serial input format for SRC2.	
		0	left-justified
		1	I2S
		2	TDM
		4	24-bit right-justified
		5	20-bit right-justified
		6	18-bit right-justified
		7	16-bit right-justified
1 (R/W)	AUTOMUTE2	Auto Hard Mute 2. The `ASRC_CTL23.AUTOMUTE2` bit auto hard mutes SRC2 when non audio is asserted by the SPDIF receiver.	
		0	Unmute
		1	Mute
0 (R/W)	HARDMUTE2	Hard Mute 2. The `ASRC_CTL23.HARDMUTE2` bit hard mutes SRC2.	
		0	Unmute
		1	Mute

Mute Register

This register connects an ASRCx mute input and output when the mute bit is cleared (=0). This allows ASRCx to automatically mute input while the ASRC is initializing (0 = automatic muting and 1 = manual muting). Bit 0 controls ASRC0, bit 1 controls ASRC1, bit 2 controls ASRC2, and bit 3 controls ASRC3.

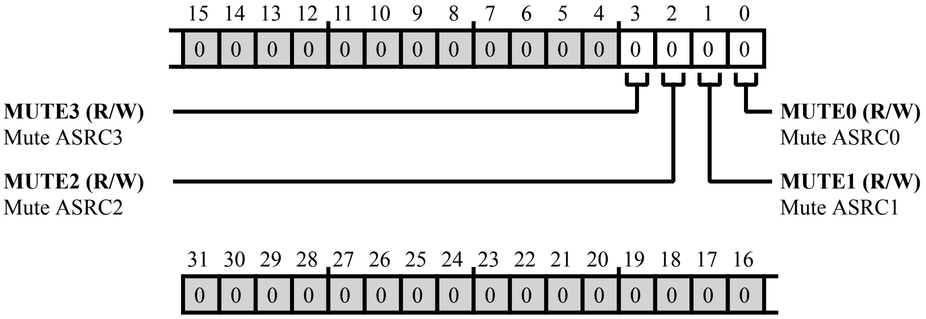

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUTE3 (R/W)
Mute ASRC3
MUTE2 (R/W)
Mute ASRC2
MUTE0 (R/W)
Mute ASRC0
MUTE1 (R/W)
Mute ASRC1
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 36-8: ASRC_MUTE Register Diagram

Table 36-6: ASRC_MUTE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	MUTE3	Mute ASRC3. The `ASRC_MUTE.MUTE3` bit automatically mutes ASCR3 output when cleared (=0).
2 (R/W)	MUTE2	Mute ASRC2. The `ASRC_MUTE.MUTE2` bit automatically mutes ASCR2 output when cleared (=0).
1 (R/W)	MUTE1	Mute ASRC1. The `ASRC_MUTE.MUTE1` bit automatically mutes ASCR1 output when cleared (=0).
0 (R/W)	MUTE0	Mute ASRC0. The `ASRC_MUTE.MUTE0` bit automatically mutes ASCR0 output when cleared (=0).

Ratio Register for ASRC 0 and 1

The `ASRC_RAT01` register report the mute and I/O sample ratio for ASRC0 and ASRC1.

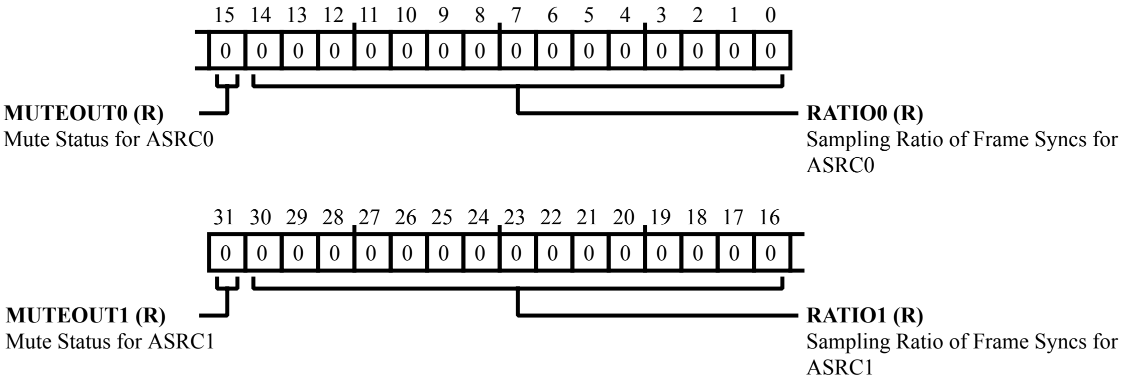

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUTEOUT0 (R)
Mute Status for ASRC0
RATIO0 (R)
Sampling Ratio of Frame Syncs for ASRC0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUTEOUT1 (R)
Mute Status for ASRC1
RATIO1 (R)
Sampling Ratio of Frame Syncs for ASRC1

Figure 36-9: ASRC_RAT01 Register Diagram

Table 36-7: ASRC_RAT01 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MUTEOUT1	Mute Status for ASRC1. The `ASRC_RAT01.MUTEOUT1` bit field reports the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared, the ratio can be read. When ASRC1 is enabled or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted.
30:16 (R/NW)	RATIO1	Sampling Ratio of Frame Syncs for ASRC1. The `ASRC_RAT01.RATIO1` bit field is read to find the ratio of output to input sampling frequency for ASRC1 (SRCx_FS_OP_I/SRCx_FS_IP_I). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15 (R/NW)	MUTEOUT0	Mute Status for ASRC0. The `ASRC_RAT01.MUTEOUT0` bit field reports the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared, the ratio can be read. When ASRC0 is enabled or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted.

Table 36-7: ASRC_RAT01 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	RATIO0	Sampling Ratio of Frame Syncs for ASRC0. The `ASRC_RAT01.RATIO0` bit field is read to find the ratio of output to input sampling frequency for ASRC0 (SRCx_FS_OP_I/SRCx_FS_IP_I). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.

Ratio Register for ASRC 2 and 3

The `ASRC_RAT23` register report the mute and I/O sample ratio for ASRC0 and ASRC1.

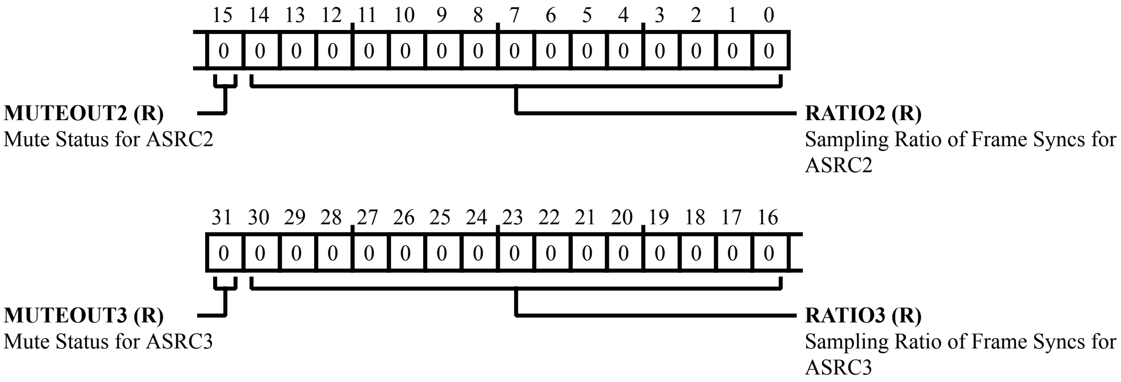

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUTEOUT2 (R)
Mute Status for ASRC2
RATIO2 (R)
Sampling Ratio of Frame Syncs for ASRC2
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MUTEOUT3 (R)
Mute Status for ASRC3
RATIO3 (R)
Sampling Ratio of Frame Syncs for ASRC3

Figure 36-10: ASRC_RAT23 Register Diagram

Table 36-8: ASRC_RAT23 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	MUTEOUT3	Mute Status for ASRC3. The `ASRC_RAT23.MUTEOUT3` bit field reports the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared, the ratio can be read. When ASRC3 is enabled or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted.
30:16 (R/NW)	RATIO3	Sampling Ratio of Frame Syncs for ASRC3. The `ASRC_RAT23.RATIO3` bit field is read to find the ratio of output to input sampling frequency for ASRC3 (SRCx_FS_OP_I/SRCx_FS_IP_I). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15 (R/NW)	MUTEOUT2	Mute Status for ASRC2. The `ASRC_RAT23.MUTEOUT2` bit field reports the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared, the ratio can be read. When ASRC2 is enabled or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loops internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted.

Table 36-8: ASRC_RAT23 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/NW)	RATIO2	Sampling Ratio of Frame Syncs for ASRC2. The `ASRC_RAT23.RATIO2` bit field is read to find the ratio of output to input sampling frequency for ASRC2 (SRCx_FS_OP_I/SRCx_FS_IP_I). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.

37 Sony/Philips Digital Interface (S/PDIF)

The Sony/Philips Digital Interface (S/PDIF) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. The digital audio interface carries three types of information; audio data, non audio data (compressed data) and timing information.

Features

The S/PDIF interface has the following features.

- Supports one stereo channel or compressed audio streams.
- AES3-compatible S/PDIF transmitter and receiver.
- Transmitting a biphase mark encoded signal that may contain any number of audio channels (compressed or linear pulse code modulation) or non-audio data.
- S/PDIF receiver managing clock recovery with separate S/PDIF on-chip PLL.
- S/PDIF receiver supports the detection of DTS frames of 256, 512, 1024, 2048, and 4096.
- Manage user status information and provide error-handling capabilities in both the transmitter and receiver.
- DAI allows interactions over DAI by serial ports and the external DAI pins to interface to other S/PDIF devices. This includes using the receiver to decode incoming biphase encoded audio streams and passing them via the SPORTs to internal memory for processing-or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system.

It is important to be familiar with serial digital audio interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

ADSP-SC58x SPDIF Register List

The S/PDIF module is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. A set of registers govern S/PDIF operations. For more information on S/PDIF functionality, see the S/PDIF register descriptions.

Table 37-1: ADSP-SC58x SPDIF Register List

Name	Description
`SPDIF_RX_CTL`	Receive Control
`SPDIF_RX_STAT`	Receive Status Register
`SPDIF_RX_STAT0_A`	Receive Status A0 Register
`SPDIF_RX_STAT0_B`	Receive Status B0 Register
`SPDIF_RX_STAT1_A`	Receive Status A1 Register
`SPDIF_RX_STAT1_B`	Receive Status B1 Register
`SPDIF_TX_CTL`	Transmit Control Register
`SPDIF_TX_STAT_A0`	Transmit Status A0 Register
`SPDIF_TX_STAT_A1`	Transmit Status A1 Register
`SPDIF_TX_STAT_A2`	Transmit Status A2 Register
`SPDIF_TX_STAT_A3`	Transmit Status A3 Register
`SPDIF_TX_STAT_A4`	Transmit Status A4 Register
`SPDIF_TX_STAT_A5`	Transmit Status A5 Register
`SPDIF_TX_STAT_B0`	Transmit Status B0 Register
`SPDIF_TX_STAT_B1`	Transmit Status B1 Register
`SPDIF_TX_STAT_B2`	Transmit Status B2 Register
`SPDIF_TX_STAT_B3`	Transmit Status B3 Register
`SPDIF_TX_STAT_B4`	Transmit Status B4 Register
`SPDIF_TX_STAT_B5`	Transmit Status B5 Register
`SPDIF_TX_UBUFF_A0`	Transmit User Buffer A0 Register
`SPDIF_TX_UBUFF_A1`	Transmit User Buffer A1 Register
`SPDIF_TX_UBUFF_A2`	Transmit User Buffer A2 Register
`SPDIF_TX_UBUFF_A3`	Transmit User Buffer A3 Register
`SPDIF_TX_UBUFF_A4`	Transmit User Buffer A4 Register
`SPDIF_TX_UBUFF_A5`	Transmit User Buffer A5 Register
`SPDIF_TX_UBUFF_B0`	Transmit User Buffer B0 Register
`SPDIF_TX_UBUFF_B1`	Transmit User Buffer B1 Register
`SPDIF_TX_UBUFF_B2`	Transmit User Buffer B2 Register
`SPDIF_TX_UBUFF_B3`	Transmit User Buffer B3 Register
`SPDIF_TX_UBUFF_B4`	Transmit User Buffer B4 Register
`SPDIF_TX_UBUFF_B5`	Transmit User Buffer B5 Register

Table 37-1: ADSP-SC58x SPDIF Register List (Continued)

Name	Description
SPDIF_TX_USRUPDT	User Bit Update Register

SRU Programming

The SRU (signal routing unit) is used to connect the S/PDIF transmitter biphase data out to the output pins or to the S/PDIF receiver. The serial clock, frame sync, data, and external sync (if external synchronization is required) inputs also need to be routed through the SRU. For details of the routing, see the DAI Routing Capabilities section in the *Digital Audio Interface (DAI)* chapter.

The SRU needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the connection to the input biphase stream.

Program the corresponding SRU registers to connect the outputs to the required destinations (see the DAI Routing Capabilities section). The biphase encoded data and the external PLL clock inputs to the receiver are routed through the SRU. The extracted clock, frame sync, and data are also routed through the SRU.

S/PDIF Interrupt List

Table 37-2: S/PDIF Interrupt List

Interrupt Name	Interrupt Condition	Return DAI Register	Return SEC Register	SEC ID
DAI0_IRQH	Block Start	DAIx_IRPTL	SEC_ID	24
DAI1_IRQH	Validity			25
DAI0_IRQL	Non Audio			145
DAI1_IRQL	Locked/Loss of Lock			146

Clocking

The module clock of the S/PDIF transceiver is SCLK0_0. The clock source for the S/PDIF receiver reference clock is CLKO5 from the CDU. When CLKO5 is configured, it supports sampling frequencies of 24 kHz to 192 kHz. The clock to this module may be shut off for power savings.

Sample rates of 24 kHz to 96 kHz are supported using a 170 MHz to 180 MHz setting on CLKO5.

Sample rates of 32 kHz to 192 kHz are supported using a 225.0 MHz setting on CLKO5.

For information on clock programming, see CDU Programming Model.

S/PDIF Transmitter

The following sections provide information on the S/PDIF transmitter.

Functional Description

The S/PDIF transmitter, shown in the ***S/PDIF Transmitter Block Diagram***, resides within the DAI, and its inputs and outputs can be routed via the SRU. It receives audio data in serial format, encloses the specified user status information, and converts it into the biphase encoded signal. The serial data input to the transmitter can be formatted as left-justified, I^{2}S, or right-justified with word widths of 16, 18, 20 or 24 bits. ***AES3 Output Block*** shows the detail of the AES block.

The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the signal routing unit (SRU).

The S/PDIF transmitter output may be routed to an output pin via the SRU and then routed to another S/PDIF receiver or to components for off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU.

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status/user registers for a frame (192-bits/24 bytes) in the transmitter that correspond to each channel or subframe.

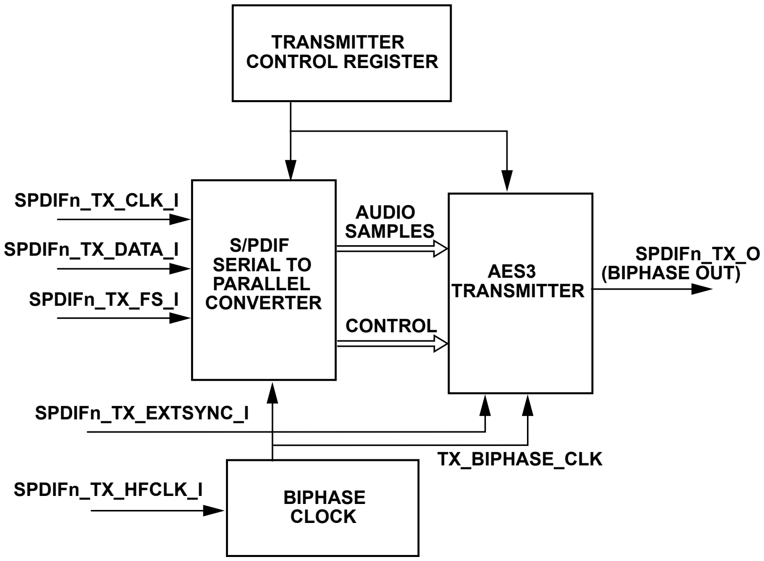

TRANSMITTER CONTROL REGISTER
SPDIFn_TX_CLK_I
SPDIFn_TX_DATA_I
SPDIFn_TX_FS_I
S/PDIF SERIAL TO PARALLEL CONVERTER
AUDIO SAMPLES
CONTROL
AES3 TRANSMITTER
SPDIFn_TX_O (BIPHASE OUT)
SPDIFn_TX_EXTSYNC_I
TX_BIPHASE_CLK
SPDIFn_TX_HFCLK_I
BIPHASE CLOCK

Figure 37-1: S/PDIF Transmitter Block Diagram

Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data.

A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I^{2}S, or right-justified with 16-, 18-, 20- or 24-bit word widths. The over sampling clock is also selected by the transmitter control register.

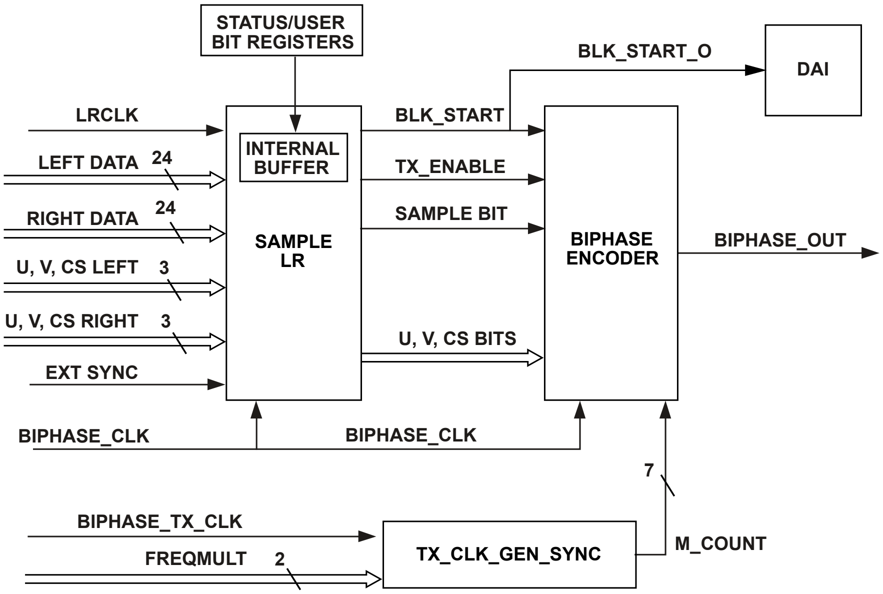

STATUS/USER
BIT REGISTERS
BLK_START_O
DAI
LRCLK
LEFT DATA
24
RIGHT DATA
24
U, V, CS LEFT
3
U, V, CS RIGHT
3
EXT SYNC
INTERNAL
BUFFER
SAMPLE
LR
BLK_START
TX_ENABLE
SAMPLE BIT
U, V, CS BITS
BIPHASE
ENCODER
BIPHASE_OUT
BIPHASE_CLK
BIPHASE_CLK
7
BIPHASE_TX_CLK
FREQMULT
2
TX_CLK_GEN_SYNC
M_COUNT

Figure 37-2: AES3 Output Block

Input Data Formats

The ***I2S and Left-Justified Formats*** and ***Right-Justified Formats*** figures show the format of data that is sent to the S/PDIF transmitter using a variety of protocol standards.

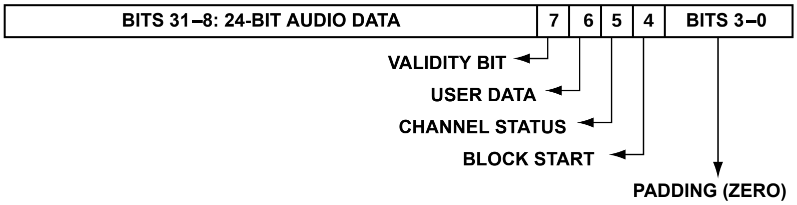

BITS 31–8: 24-BIT AUDIO DATA
7
6
5
4
BITS 3–0
VALIDITY BIT
USER DATA
CHANNEL STATUS
BLOCK START
PADDING (ZERO)

Figure 37-3: I^{2}S and Left-Justified Formats

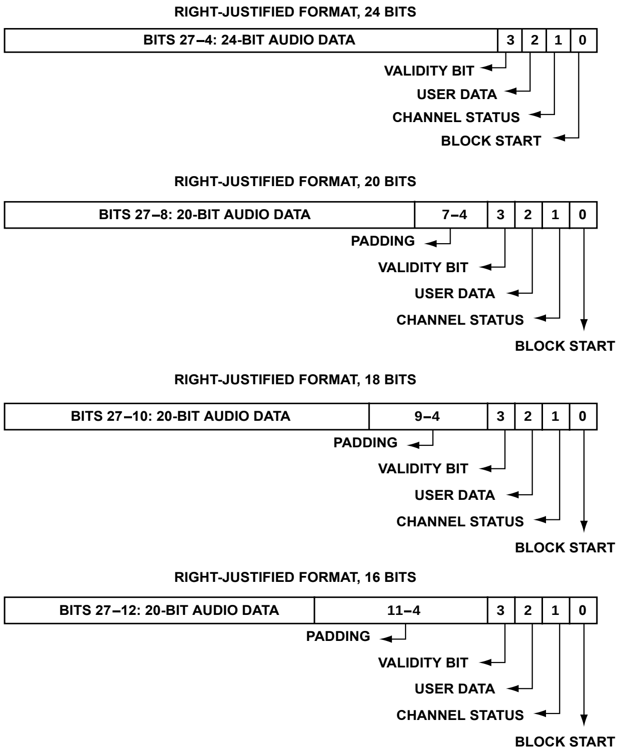

RIGHT-JUSTIFIED FORMAT, 24 BITS
BITS 27–4: 24-BIT AUDIO DATA
3
2
1
0
VALIDITY BIT
USER DATA
CHANNEL STATUS
BLOCK START
RIGHT-JUSTIFIED FORMAT, 20 BITS
BITS 27–8: 20-BIT AUDIO DATA
7–4
PADDING
VALIDITY BIT
USER DATA
CHANNEL STATUS
BLOCK START
RIGHT-JUSTIFIED FORMAT, 18 BITS
BITS 27–10: 20-BIT AUDIO DATA
9–4
PADDING
VALIDITY BIT
USER DATA
CHANNEL STATUS
BLOCK START
RIGHT-JUSTIFIED FORMAT, 16 BITS
BITS 27–12: 20-BIT AUDIO DATA
11–4
PADDING
VALIDITY BIT
USER DATA
CHANNEL STATUS
BLOCK START

Figure 37-4: Right-Justified Formats

Operating Modes

The S/PDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Full Serial Mode

This mode is selected by clearing the `SPDIF_TX_CTL.AUTO` bit. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the serial data stream (`SPDIF_TX_DATA_I`) pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Standalone Mode

This mode is selected by setting the `SPDIF_TX_CTL.AUTO` bit. In this mode, the block start bit (indicating the start of a frame) is generated internally. The channel status bits come from the channel status buffer registers. The user status bits come from the user bits buffers as shown in the ***AES3 Output Block*** figure.

The validity bits are the `SPDIF_TX_CTL.VALIDR` and `SPDIF_TX_CTL.VALIDL`. In this mode only audio data comes from the `SPDIF_TX_DATA_I` pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Once the user bits buffer registers (`SPDIF_TX_UBUFF_A0` - `SPDIF_TX_UBUFF_B5`) are programmed, they are used only for the next block of data. This allows programs to change the user bit information in every block of data.

To allow user bit updates, write a 0x1 to the `SPDIF_TX_USRUPDT` register that is used for further processing. If the `SPDIF_TX_CTL.AUTO` bit is set:

- and if `SPDIF_TX_USRUPDT` =1, at every 192nd frame end the user status bits are taken from user bits buffers and transmitted. Simultaneously, the `SPDIF_TX_USRUPDT` register is cleared automatically by hardware.
- and if `SPDIF_TX_USRUPDT` =0, at every 192nd frame end the user status bits are updated as zeros and transmitted. The `SPDIF_TX_USRUPDT` register remains low.

In general, for the next block, programs can update user bits buffers at any time during the transfer of the current block (1 block = 192 frames). There are internal buffers to store the user status bits of the current block of transfer. In other words, at the beginning of every new block, the user status bit (`SPDIF_TX_CTL.USRPEND` bit) from user bits buffers are copied to internal buffers and transmitted in each frame during the transfer.

Note that since a frame contains 192 bits/8 = 24 bytes, six status/user registers are required to store each four bytes.

Data Output Mode

Two output data formats are supported by the transmitter; *two channel mode* and *single-channel double-frequency* (SCDF) mode. The output format is determined by the transmitter control register (`SPDIF_TX_CTL`).

In two channel mode, the left channel (channel A) is transmitted when the `SPDIF_TX_FS_I` is high and the right channel (channel B) is transmitted when the `SPDIF_TX_FS_I` is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both sub frames, instead of channel A and B. The transmitter will transmit at half the sample rate of the input bit stream. The `SPDIF_TX_CTL.SCDF` bit selects SCDF mode. When in SCDF mode, the `SPDIF_TX_CTL.SCDFLR` bit determines whether left or right channel data is transmitted.

S/PDIF Receiver

The S/PDIF receiver (***S/PDIF Receiver Block Diagram***) is compliant with all common serial digital audio interface standards including IEC-60958, IEC-61937, AES3, and AES11. For the IEC-60958 standard, all the user-data and channel-status bits (as outlined in this document) are not decoded by the S/PDIF receiver. The interface does make all 192 user-data and channel-status pairs available as an output of the block, for post-decoding.

For the IEC-61937 standard, the S/PDIF only detects compressed AC-3 and DTS formats. No decompression is performed.

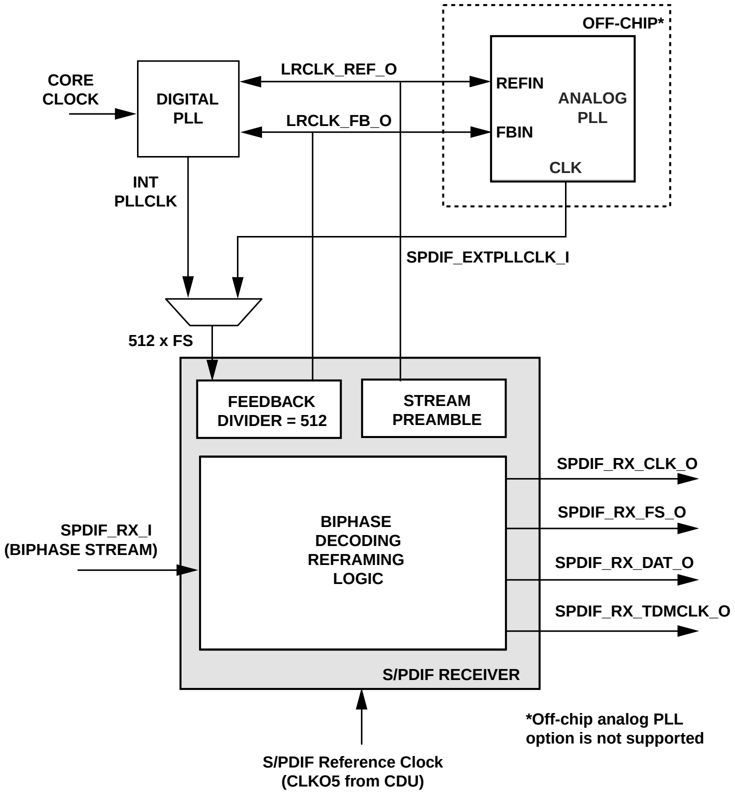

OFF-CHIP*
CORE CLOCK
DIGITAL PLL
LRCLK_REF_O
LRCLK_FB_O
REFIN
FBIN
ANALOG PLL
CLK
INT PLLCLK
SPDIF_EXTPLLCLK_I
512 x FS
FEEDBACK DIVIDER = 512
STREAM PREAMBLE
SPDIF_RX_CLK_O
SPDIF_RX_FS_O
SPDIF_RX_DAT_O
SPDIF_RX_TDMCLK_O
SPDIF_RX_I (BIPHASE STREAM)
BIPHASE DECODING REFRAMING LOGIC
S/PDIF RECEIVER
*Off-chip analog PLL option is not supported
S/PDIF Reference Clock (CLKO5 from CDU)

Figure 37-5: S/PDIF Receiver Block Diagram

Functional Description

If the receiver is used, programs need to enable it using the `SPDIF_RX_CTL` register. After the SRU programming is complete, write to the register with control values. At this point, the receiver attempts to lock.

NOTE: The S/PDIF receiver is disabled at default. If the receiver is used in an application, programs should enable the receiver.

The input to the receiver (SPDIFn_RX_I) is a biphase encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single biphase encoded stream, producing an I^{2}S compatible serial data output that consists of a serial clock, a left-right frame sync, and data (channel A/B). It provides the programmer with several methods of managing the incoming status bit information.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 24 kHz to 192 kHz. Refer to Clocking for more details.

The channel status bits are collected into memory-mapped registers, while other channel status and user bytes must be handled manually. The block start bit, which replaces the parity bit in the serial I^{2}S stream, indicates the reception of the Z preamble and the start of a new block of channel status and data bits.

Clock Recovery

The S/PDIF receiver recovers the clock that generated the AES3/SPDIF biphase encoded stream from the incoming S/PDIF stream.

This clock is used by the receiver to clock in the biphase encoded data stream and also to provide clocks for either the SPORTs, sample rate converter, or the AES3 and S/PDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

In order to maintain performance, jitter on the clock is sourced to several peripherals.

To be AES11 compliant, the recovered left/right clock must be aligned with the preambles within a + or - 5% of the frame period. Since the PLL clock generates a clock 512 times the frame rate clock (512 f_{SCLK}), this clock can be used and divided down to create the phase aligned jitter-free left/right clock.

Output Data Format

The extracted 24-bit audio data, V, U, C and block start bits are sent on the `SPDIF_RX_DAT` pin in 32-bit I^2S format as shown in ***I2S and Left-Justified Formats***. The frame sync is transmitted on the `SPDIF_RX_FS` pin and serial clock is transmitted on the `SPDIF_RX_CLK` pin. All three pins are routed through the SRU.

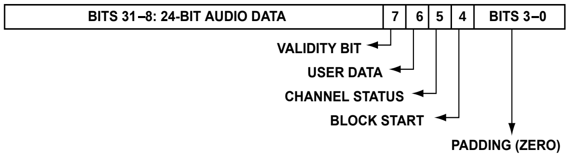


Figure 37-6: I2S Format

Channel Status

The channel status for the first bytes 4-0 (consumer mode) are collected into memory-mapped registers (`SPDIF_RX_STAT0_A`, `SPDIF_RX_STAT0_B`, `SPDIF_RX_STAT1_A` and `SPDIF_RX_STAT1_B`). All other channel status bytes 23-5 (professional mode) must be manually extracted from the receiver data stream.

NOTE: Only the first 5 channel status bytes (40-bit) for consumer mode of a frame are stored into the S/PDIF receiver status registers.

Operating Modes

This section describes the receiver channel status for the different modes.

Compressed or Non-linear Audio Data

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how this peripheral handles different data.

MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the `SPDIF_RX_STAT.VALID` bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of 0xF872 and another word consisting of 0x4E1F. When this sync code is detected, the `SPDIF_RX_STAT.COMPMODE` bits hold the information regarding type of compression.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.

Emphasized Audio Data

Determination as to whether the received audio data is emphasized or not is done in software using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2-4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.
- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3-5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

Single-Channel Double-Frequency Mode

Unlike previous processors, support for single-channel, double-frequency mode (SCDF) is not supported through specific bits within the `SPDIF_RX_CTL` register, but rather have to be implemented in software using the information provided by the CS (channel status) bits.

- 0111 = single channel double frequency mode
- 1000 = single channel double frequency mode-stereo left
- 1001 = single channel double frequency mode-stereo right

Clock Recovery Modes

The S/PDIF receiver extracts audio data, channel status, and user bits from the biphase encoded AES3 and S/PDIF stream. In addition, a 50% duty cycle reference clock running at the sampling rate of the audio input data is generated for the receiver to recover the oversampling clock.

Number Controlled Oscillator

The receiver can recover the clock from the biphase encoded stream using an on-chip NCO shown in the following figure. Note the dedicated NCO is separate from the PLL that supplies the clock to the processor core and which is the default operation of the receiver.

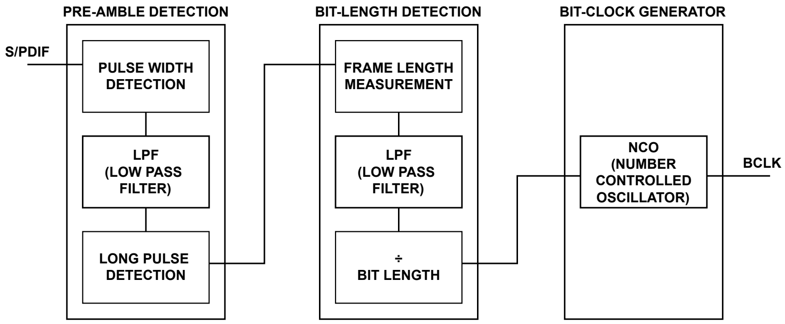

PRE-AMBLE DETECTION
BIT-LENGTH DETECTION
BIT-CLOCK GENERATOR
S/PDIF
PULSE WIDTH DETECTION
LPF (LOW PASS FILTER)
LONG PULSE DETECTION
FRAME LENGTH MEASUREMENT
LPF (LOW PASS FILTER)
÷ BIT LENGTH
NCO (NUMBER CONTROLLED OSCILLATOR)
BCLK

Figure 37-7: S/PDIF Clock Recovery Mechanism

The left/right frame reference clock for the NCO is generated using the preambles. The recovered low jitter left/right frame clock from the NCO attempts to align with the reference clock. However, this recovered left/right clock, like the reference clock, is not phase aligned with the preambles.

Interrupts

The following sections provide information about interrupt sources, masking and servicing.

Sources

The S/PDIF module of each DAI drives five interrupt signals. Four are status signals driven from SPDIFn_RX and one signal is driven from SPDIFn_TX (block start). These signals are connected into the `DAI_IRPTL_L`/`DAI_IRPTL_H` latch register.

Transmit Block Start

The SPDIFn_TX_BLKSTART output signal, if routed to any miscellaneous interrupt bits (DAIn_INT_31-22 in the `DAI_MISC0`/`DAI_MISC1` registers), triggers a block start interrupt during the last frame of current block.

Receiver Status

The following receiver status bits generate an interrupt.

- Validity (`SPDIF_RX_STAT.VALID`)
- Receiver locked (`SPDIF_RX_STAT.LOCK`)
- No audio (`SPDIF_RX_STAT.AUDIOTYPE`)

Receiver Error

The loss of lock (`SPDIF_RX_STAT.LOCKLOSS`) bit generates an interrupt.

Masking

For the S/PDIF receiver the `DAI_IMSK_RE` register must be unmasked accordingly. For the S/PDIF transmit the DAIn_IMASK_x register must be unmasked accordingly.

The `INTR_DAI_IRQH` and `INTR_DAI_IRQL` signals are routed to the SEC and GIC.

Service

The ISR reads the `DAI_IRPTL_H` and `DAI_IRPTL_L` registers to clear the interrupt request.

Programming Model

The following sections provide information on programming the transmitter and receiver.

Programming the Transmitter

Since the S/PDIF transmitter data input is not available to the core, programming the transmitter is as simple as: 1) connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be encoded, and 2) selecting the desired mode in the transmitter control register. This setup can be accomplished in three steps.

1. Connect the transmitter's four required input signals and one biphase encoded output in the SRU. The four input signals are the serial clock (`SPDIF_TX_CLK_I`), the serial frame sync (`SPDIF_TX_FS_I`), the serial data (`SPDIF_TX_DAT_I`), and the high frequency clock (`SPDIF_TX_HFCLK_I`) used for the encoding. The only output of the transmitter is `SPDIF_TX_O`.
2. If user bits are required, write 0x1 to the `SPDIF_TX_USRUPDT` register for the first block of transfer. Also route the `SPDIF_TX_BLK_START_O` signal to the `DAI_INT_31-22` (`DAI_IRPTLx` register). This generates interrupts during the last frame of the block (192), allowing changes of user bits for the next block.
3. Initialize the `SPDIF_TX_CTL` register to enable the data encoding.
4. Manually set the block start bit in the data stream once per block (every 384 words). This is necessary if automatic generation of block start information is not enabled using the `SPDIF_TX_CTL.AUTO` bit = 0.

NOTE: For more information, see the "DAI Routing Capabilities" section of the *Digital Audio Interface (DAI)* chapter.

Programming the Receiver

Since the S/PDIF receiver data output is not available to the core, programming the peripheral is as simple as connecting the SRU to the on-chip (serial ports) or off-chip (DAI pins) serial devices that provide the clock and data to be decoded, and selecting the desired mode in the receiver control register. This setup can be accomplished in two steps.

1. Connect the input signal and three output signals in the SRU. The only input of the receiver is the biphase encoded stream, `SPDIFn_RX_I`. The three required output signals are the serial clock (`SPDIFn_RX_CLK`), the serial frame sync (`SPDIFn_RX_FS`), and the serial data (`SPDIFn_RX_DAT`). The high frequency clock (`SPDIFn_RX_TDMCLK`) derived from the encoded stream is also available if the system requires it.

2. Initialize the `SPDIF_RX_CTL` register to enable the data decoding. Note that this peripheral is disabled by default.

NOTE: For more information, see the "DAI Routing Capabilities" section of the *Digital Audio Interface (DAI)* chapter.

Interrupted Data Streams on the Receiver

When using the S/PDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the S/PDIF receiver's digital PLL will relock to the stream. The steps to accomplish this are described below.

1. Set up interrupts within the DAI so that the S/PDIF receiver can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the NCO. This is accomplished by setting and then clearing the `SPDIF_RX_CTL.RST` bit.
3. Return from the ISR and continue normal operation.

This method of resetting the NCO has been shown to provide extremely reliable performance when the S/PDIF inputs are interrupted or unplugged momentarily.

Debug Features

The following feature supports S/PDIF debugging.

Loopback Routing

The S/PDIF supports an internal loopback mode by using the SRU. For more information about loopback, see "Loopback Routing" in the the *Digital Audio Interface (DAI)* chapter.

ADSP-SC58x SPDIF Register Descriptions

The S/PDIF module (SPDIF) contains the following registers.

Table 37-3: ADSP-SC58x SPDIF Register List

Name	Description
`SPDIF_RX_CTL`	Receive Control
`SPDIF_RX_STAT`	Receive Status Register
`SPDIF_RX_STAT0_A`	Receive Status A0 Register
`SPDIF_RX_STAT0_B`	Receive Status B0 Register
`SPDIF_RX_STAT1_A`	Receive Status A1 Register
`SPDIF_RX_STAT1_B`	Receive Status B1 Register

Table 37-3: ADSP-SC58x SPDIF Register List (Continued)

Name	Description
SPDIF_TX_CTL	Transmit Control Register
SPDIF_TX_STAT_A0	Transmit Status A0 Register
SPDIF_TX_STAT_A1	Transmit Status A1 Register
SPDIF_TX_STAT_A2	Transmit Status A2 Register
SPDIF_TX_STAT_A3	Transmit Status A3 Register
SPDIF_TX_STAT_A4	Transmit Status A4 Register
SPDIF_TX_STAT_A5	Transmit Status A5 Register
SPDIF_TX_STAT_B0	Transmit Status B0 Register
SPDIF_TX_STAT_B1	Transmit Status B1 Register
SPDIF_TX_STAT_B2	Transmit Status B2 Register
SPDIF_TX_STAT_B3	Transmit Status B3 Register
SPDIF_TX_STAT_B4	Transmit Status B4 Register
SPDIF_TX_STAT_B5	Transmit Status B5 Register
SPDIF_TX_UBUFF_A0	Transmit User Buffer A0 Register
SPDIF_TX_UBUFF_A1	Transmit User Buffer A1 Register
SPDIF_TX_UBUFF_A2	Transmit User Buffer A2 Register
SPDIF_TX_UBUFF_A3	Transmit User Buffer A3 Register
SPDIF_TX_UBUFF_A4	Transmit User Buffer A4 Register
SPDIF_TX_UBUFF_A5	Transmit User Buffer A5 Register
SPDIF_TX_UBUFF_B0	Transmit User Buffer B0 Register
SPDIF_TX_UBUFF_B1	Transmit User Buffer B1 Register
SPDIF_TX_UBUFF_B2	Transmit User Buffer B2 Register
SPDIF_TX_UBUFF_B3	Transmit User Buffer B3 Register
SPDIF_TX_UBUFF_B4	Transmit User Buffer B4 Register
SPDIF_TX_UBUFF_B5	Transmit User Buffer B5 Register
SPDIF_TX_USRUPDT	User Bit Update Register

Receive Control

The `SPDIF_RX_CTL` register is used to enable and control the S/PDIF receiver.

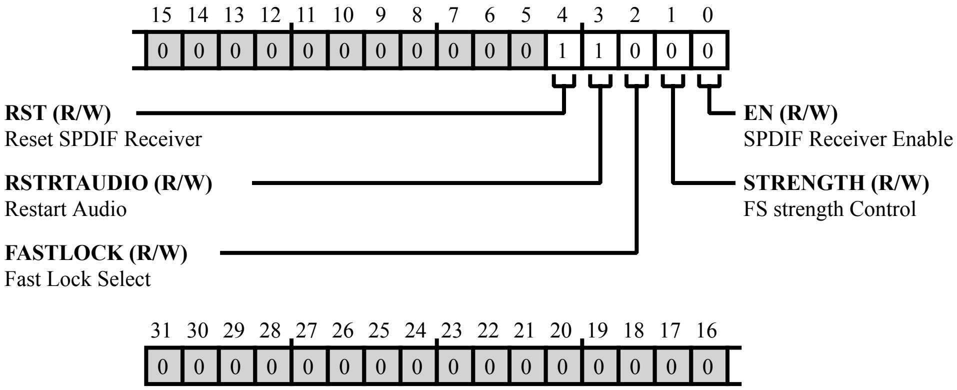

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
RST (R/W)
Reset SPDIF Receiver
RSTRTAUDIO (R/W)
Restart Audio
FASTLOCK (R/W)
Fast Lock Select
EN (R/W)
SPDIF Receiver Enable
STRENGTH (R/W)
FS strength Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 37-8: SPDIF_RX_CTL Register Diagram

Table 37-4: SPDIF_RX_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	RST	Reset SPDIF Receiver. The `SPDIF_RX_CTL.RST` bit resets the receiver.	
		0	Takes the receiver out of reset
		1	Resets the SPDIF receiver
3 (R/W)	RSTRTAUDIO	Restart Audio. The `SPDIF_RX_CTL.RSTRTAUDIO` bit restarts the audio once a re-lock has occurred. When the S/PDIF receiver loses lock the audio output is set to 0. This bit determines the behavior of the audio once lock is re-established. Audio can be manually restarted by toggling this bit high and then low.	
		0	Manually restart audio
		1	Automatically restart audio
2 (R/W)	FASTLOCK	Fast Lock Select. The `SPDIF_RX_CTL.FASTLOCK` bit allows the lock mechanism to lock at normal speed or at faster speed. This has the advantage of recovering very quickly whenever the S/PDIF receiver looses lock due to glitches in the signal. In normal mode the S/PDIF receiver locks after 64 consecutive valid samples, in fast mode the S/PDIF receiver locks after 8 consecutive valid samples	
		0	Enable normal mode
		1	Enable fast mode

Table 37-4: SPDIF_RX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	STRENGTH	FS strength Control. The `SPDIF_RX_CTL.STRENGTH` bit controls the strength of the bit clock and Frame sync outputs from the SPDIF receiver. In strong mode these output signals are continued (as best possible) when the receiver notices a loss-of-lock condition. Note that 'as best possible' refers to the fact that this recovered signal may not be accurate, given the loss-of-lock condition. In weak mode these output signals are interrupted as soon as the receiver notices a loss-of-lock condition.	
		0	Enable strong mode
		1	Enable weak mode
0 (R/W)	EN	SPDIF Receiver Enable. When the `SPDIF_RX_CTL.EN` bit =0 the clock to SPDIF is switched off for power savings.	
		0	Disable receiver
		1	Enable receiver

Receive Status Register

The `SPDIF_RX_STAT` register consists of bits that indicate the status of various functions supported by S/PDIF receiver.

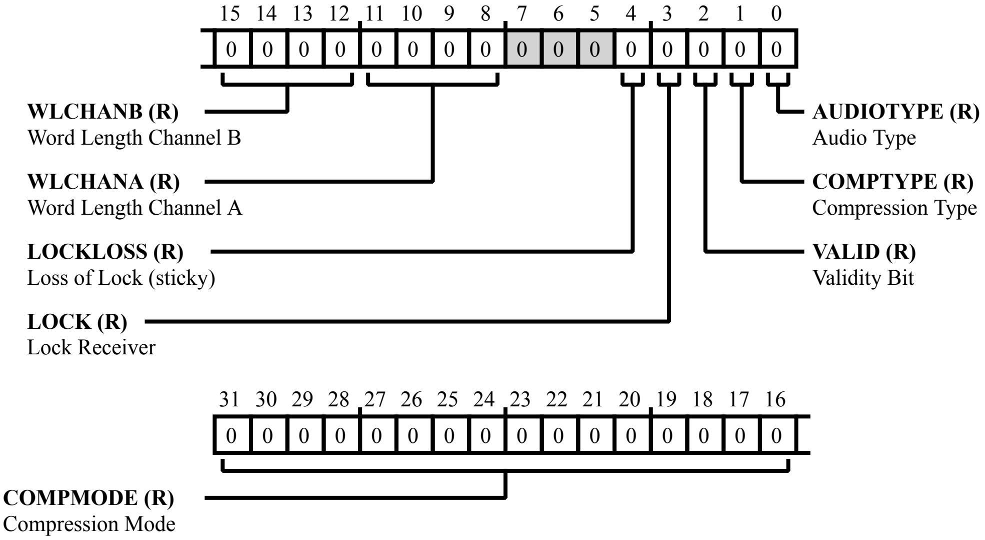

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WLCHANB (R)
Word Length Channel B
WLCHANA (R)
Word Length Channel A
LOCKLOSS (R)
Loss of Lock (sticky)
LOCK (R)
Lock Receiver
AUDIOTYPE (R)
Audio Type
COMPTYPE (R)
Compression Type
VALID (R)
Validity Bit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COMPMODE (R)
Compression Mode

Figure 37-9: SPDIF_RX_STAT Register Diagram

Table 37-5: SPDIF_RX_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16 (R/NW)	COMPMODE	Compression Mode. The `SPDIF_RX_STAT.COMPMODE` bit field indicates the type of compression mode used in the Digital audio stream. The value in this field indicates the 16-bit burst information as specified by the IEC 62937-2 standard. Use this document to decode the value in this bit field.	
15:12 (R/NW)	WLCHANB	Word Length Channel B. The `SPDIF_RX_STAT.WLCHANB` bit field indicates the S/PDIF word length for channel B. May be decoded as follows (from the S/PDIF standard).	
		0	Reserved
		1	Reserved
		2	SPDIF_LENGTH_16
		3	Reserved
		4	SPDIF_LENGTH_18
		5	SPDIF_LENGTH_22
		6	Reserved

Table 37-5: SPDIF_RX_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		7	Reserved
		8	SPDIF_LENGTH_19
		9	SPDIF_LENGTH_23
		10	SPDIF_LENGTH_20
		11	SPDIF_LENGTH_24
		12	SPDIF_LENGTH_17
		13	SPDIF_LENGTH_21
		14	Reserved
		15	Reserved
11:8 (R/NW)	WLCHANA	Word Length Channel A. The `SPDIF_RX_STAT.WLCHANA` bit indicates the S/PDIF word length for channel A. May be decoded as follows (from the S/PDIF standard).	
		0	Reserved
		1	Reserved
		2	SPDIF_LENGTH_16
		3	Reserved
		4	SPDIF_LENGTH_18
		5	SPDIF_LENGTH_22
		6	Reserved
		7	Reserved
		8	SPDIF_LENGTH_19
		9	SPDIF_LENGTH_23
		10	SPDIF_LENGTH_20
		11	SPDIF_LENGTH_24
		12	SPDIF_LENGTH_17
		13	SPDIF_LENGTH_21
		14	Reserved
		15	Reserved

Table 37-5: SPDIF_RX_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/NW)	LOCKLOSS	Loss of Lock (sticky). The `SPDIF_RX_STAT.LOCKLOSS` bit indicates that the system has lost lock. This is different to the lock register, as it is sticky it goes high when system looses lock, but returns to low once `SPDIF_RX_CTL.RSTRTAUDIO` bit is toggled. This is to allow the programs to poll the lock status.	
		0	SPDIF receiver locked
		1	SPDIF Receiver lost lock
3 (R/NW)	LOCK	Lock Receiver. The `SPDIF_RX_STAT.LOCK` bit indicates the S/PDIF receiver has successfully locked to the S/PDIF stream and is outputting valid data.	
		0	Receiver not locked
		1	Receiver locked
2 (R/NW)	VALID	Validity Bit. The `SPDIF_RX_STAT.VALID` bit indicates the ORed bits of channel A and B.	
		0	Linear PCM data
		1	Non-linear audio data
1 (R/NW)	COMPTYPE	Compression Type. The `SPDIF_RX_STAT.COMPTYPE` bit indicates AC3 or DTS compression. Valid only if `SPDIF_RX_STAT.AUDIOTYPE` indicates compressed data.	
		0	AC3 compressed data
		1	DTS compressed data
0 (R/NW)	AUDIOTYPE	Audio Type. The `SPDIF_RX_STAT.AUDIOTYPE` bit indicates PCM or compressed data.	
		0	PCM data
		1	Compressed Data

Receive Status A0 Register

The `SPDIF_RX_STAT0_A` register holds the receive channel 0 status for bytes 0-3 for sub frame A.

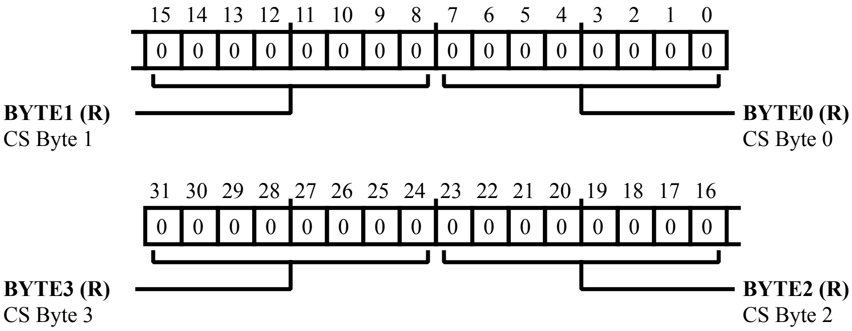

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE1 (R)
CS Byte 1
BYTE0 (R)
CS Byte 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE3 (R)
CS Byte 3
BYTE2 (R)
CS Byte 2

Figure 37-10: SPDIF_RX_STAT0_A Register Diagram

Table 37-6: SPDIF_RX_STAT0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	BYTE3	CS Byte 3. The `SPDIF_RX_STAT0_A.BYTE3` bit field contains byte 3 of received channel A status.
23:16 (R/NW)	BYTE2	CS Byte 2. The `SPDIF_RX_STAT0_A.BYTE2` bit field contains byte 2 of received channel A status.
15:8 (R/NW)	BYTE1	CS Byte 1. The `SPDIF_RX_STAT0_A.BYTE1` bit field contains byte 1 of received channel A status.
7:0 (R/NW)	BYTE0	CS Byte 0. The `SPDIF_RX_STAT0_A.BYTE0` bit field contains the status of byte 0-3 of received channel A.

Receive Status B0 Register

The SPDIF_RX_STAT0_B register holds the receive channel 0 status for bytes 0-3 for sub frame B.

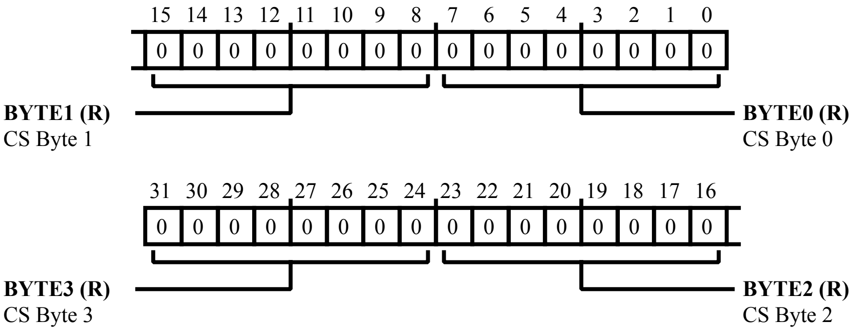

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE1 (R)
CS Byte 1
BYTE0 (R)
CS Byte 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE3 (R)
CS Byte 3
BYTE2 (R)
CS Byte 2

Figure 37-11: SPDIF_RX_STAT0_B Register Diagram

Table 37-7: SPDIF_RX_STAT0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	BYTE3	CS Byte 3. The `SPDIF_RX_STAT0_B.BYTE3` bit field contains byte 3 of received channel B status.
23:16 (R/NW)	BYTE2	CS Byte 2. The `SPDIF_RX_STAT0_B.BYTE2` bit field contains byte 2 of received channel B status.
15:8 (R/NW)	BYTE1	CS Byte 1. The `SPDIF_RX_STAT0_B.BYTE1` bit field contains byte 1 of received channel B status.
7:0 (R/NW)	BYTE0	CS Byte 0. The `SPDIF_RX_STAT0_B.BYTE0` bit field contains byte 0 of received channel B status.

Receive Status A1 Register

The `SPDIF_RX_STAT1_A` register holds the receive channel 1 status for byte 4 for sub frame A.

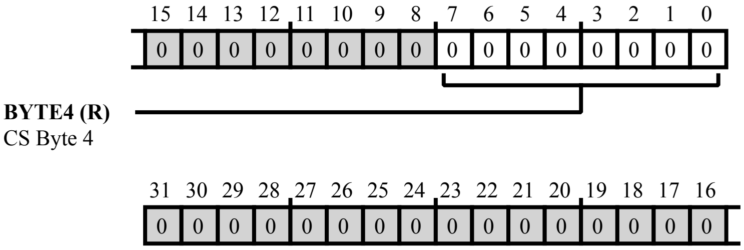

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE4 (R)
CS Byte 4
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 37-12: SPDIF_RX_STAT1_A Register Diagram

Table 37-8: SPDIF_RX_STAT1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	BYTE4	CS Byte 4. The `SPDIF_RX_STAT1_A.BYTE4` bit field contains byte 4 of received channel A status.

Receive Status B1 Register

The SPDIF_RX_STAT1_B register holds the receive channel 1 status for byte 4 for sub frame B.

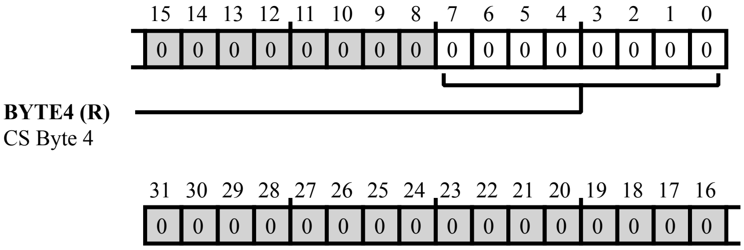

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE4 (R)
CS Byte 4
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 37-13: SPDIF_RX_STAT1_B Register Diagram

Table 37-9: SPDIF_RX_STAT1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	BYTE4	CS Byte 4. The `SPDIF_RX_STAT1_B.BYTE4` bit field contains byte 4 of received channel B status.

Transmit Control Register

The `SPDIF_TX_CTL` register provides bits the enable/disable the transmitter and configure several options related to data transmission.

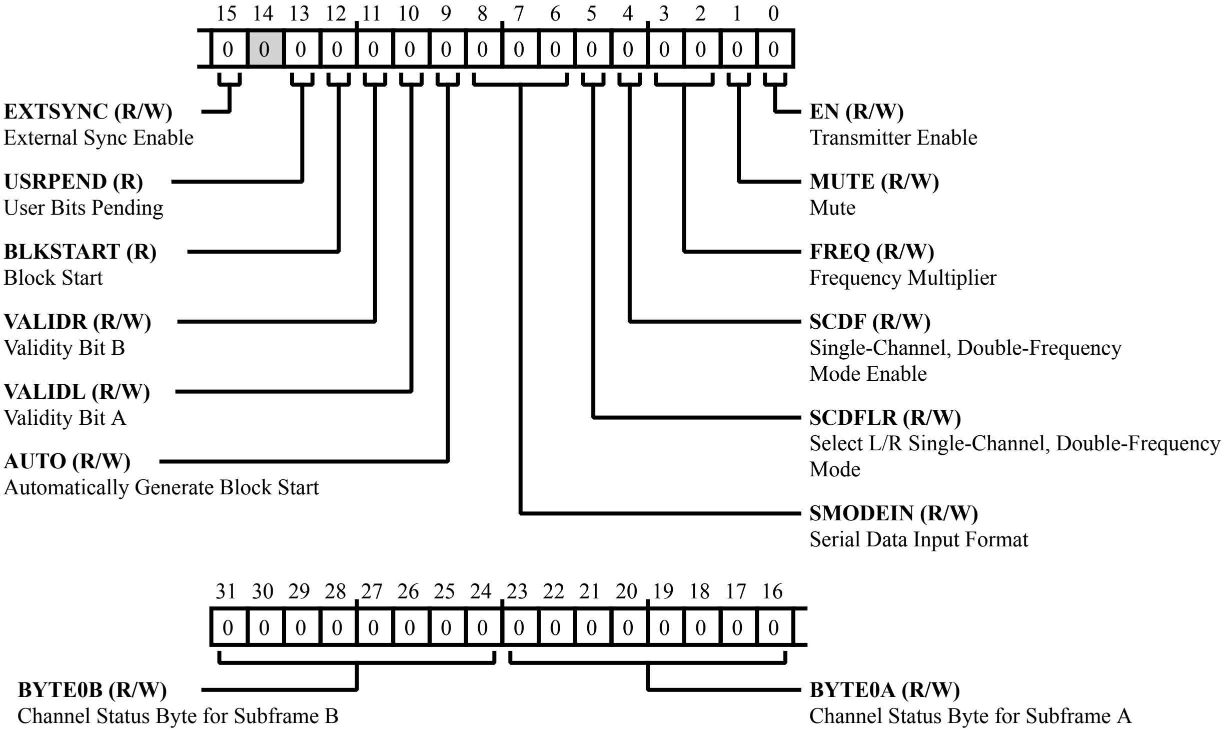

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EXTSYNC (R/W)
External Sync Enable
USRPEND (R)
User Bits Pending
BLKSTART (R)
Block Start
VALIDR (R/W)
Validity Bit B
VALIDL (R/W)
Validity Bit A
AUTO (R/W)
Automatically Generate Block Start
EN (R/W)
Transmitter Enable
MUTE (R/W)
Mute
FREQ (R/W)
Frequency Multiplier
SCDF (R/W)
Single-Channel, Double-Frequency Mode Enable
SCDFLR (R/W)
Select L/R Single-Channel, Double-Frequency Mode
SMODEIN (R/W)
Serial Data Input Format
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE0B (R/W)
Channel Status Byte for Subframe B
BYTE0A (R/W)
Channel Status Byte for Subframe A

Figure 37-14: SPDIF_TX_CTL Register Diagram

Table 37-10: SPDIF_TX_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE0B	Channel Status Byte for Subframe B. The `SPDIF_TX_CTL.BYTE0B` bit field contains the channel status for the second bytes 95.
23:16 (R/W)	BYTE0A	Channel Status Byte for Subframe A. The `SPDIF_TX_CTL.BYTE0A` bit field contains the channel status for the first bytes 40.
15 (R/W)	EXTSYNC	External Sync Enable. When the `SPDIF_TX_CTL.EXTSYNC` bit is set (regardless of the `SPDIF_TX_CTL.AUTO` bit setting) the internal frame counter is set to zero at an internal LRCLK rising edge followed by an EXTSYNC_I rising edge.

Table 37-10: SPDIF_TX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	USRPEND	User Bits Pending. The `SPDIF_TX_CTL.USRPEND` bit is set if the update of the internal buffer from the Transmit User Bits Buffer registers has not yet completed.	
12 (R/NW)	BLKSTART	Block Start. The `SPDIF_TX_CTL.BLKSTART` bit is a status bit that indicates block start (when the `SPDIF_TX_CTL.AUTO` bit = 1).	
		0	Current word is not block start
		1	Current word is block start
11 (R/W)	VALIDR	Validity Bit B. Use the `SPDIF_TX_CTL.VALIDR` bit with the channel status buffer.	
10 (R/W)	VALIDL	Validity Bit A. The `SPDIF_TX_CTL.VALIDL` bit either manually start block transfer according to input stream status bits or automatically start block transfer. Use the `SPDIF_TX_CTL.VALIDL` bit with the channel status buffer.	
		0	Manually start block transfer
		1	Automatically start block transfer
9 (R/W)	AUTO	Automatically Generate Block Start. When enabled, the transmitter is in standalone mode where it inserts block start, channel status, and validity bits on its own. If the channel status or validity buffer needs to be enabled (after the DAI programming is complete), first write to the buffers with the required data and then enable the buffers by setting the `SPDIF_TX_CTL.AUTO` bit.	
8:6 (R/W)	SMODEIN	Serial Data Input Format. The `SPDIF_TX_CTL.SMODEIN` bit selects the data input format.	
		0	Left-justified
		1	I2S
		2	Reserved
		3	Reserved
		4	Right-justified, 24 bits
		5	Right-justified, 20 bits
		6	Right-justified, 18 bits
		7	Right-justified, 16 bits

Table 37-10: SPDIF_TX_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SCDFLR	Select L/R Single-Channel, Double-Frequency Mode. The `SPDIF_TX_CTL.SCDFLR` bit selects the left or right channel in SCDF mode.	
		0	Left channel
		1	Right channel
4 (R/W)	SCDF	Single-Channel, Double-Frequency Mode Enable. The `SPDIF_TX_CTL.SCDF` bit enables single-channel, double-frequency mode.	
		0	Two-channel mode
		1	SCDF mode
3:2 (R/W)	FREQ	Frequency Multiplier. The `SPDIF_TX_CTL.FREQ` bit field sets the oversampling ratio.	
		0	256 x frame sync oversampling
		1	384 x frame sync oversampling
		2-3	Reserved
1 (R/W)	MUTE	Mute. The `SPDIF_TX_CTL.MUTE` bit mutes the serial data output.	
		0	Disable Mute
		1	Enable Mute
0 (R/W)	EN	Transmitter Enable. The `SPDIF_TX_CTL.EN` bit enables the transmitter and resets the control registers to their defaults.	
		0	Transmitter disabled
		1	Transmitter enabled

Transmit Status A0 Register

The `SPDIF_TX_STAT_A0` register holds the transmit channel 0 status for bytes 1-4 for sub frame A.

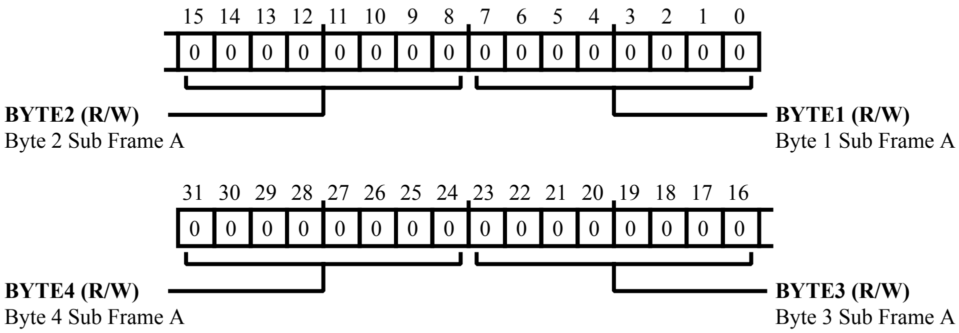

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE2 (R/W)
Byte 2 Sub Frame A
BYTE1 (R/W)
Byte 1 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE4 (R/W)
Byte 4 Sub Frame A
BYTE3 (R/W)
Byte 3 Sub Frame A

Figure 37-15: SPDIF_TX_STAT_A0 Register Diagram

Table 37-11: SPDIF_TX_STAT_A0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE4	Byte 4 Sub Frame A. The `SPDIF_TX_STAT_A0.BYTE4` bit field holds the transmit channel 0 status for byte 4 for sub frame A.
23:16 (R/W)	BYTE3	Byte 3 Sub Frame A. The `SPDIF_TX_STAT_A0.BYTE3` bit field holds the transmit channel 0 status for byte 3 for sub frame A.
15:8 (R/W)	BYTE2	Byte 2 Sub Frame A. The `SPDIF_TX_STAT_A0.BYTE2` bit field holds the transmit channel 0 status for byte 2 for sub frame A.
7:0 (R/W)	BYTE1	Byte 1 Sub Frame A. The `SPDIF_TX_STAT_A0.BYTE1` bit field holds the transmit channel 0 status for byte 1 for sub frame A.

Transmit Status A1 Register

The `SPDIF_TX_STAT_A1` register holds the transmit status for bytes 5-8 for sub frame A.

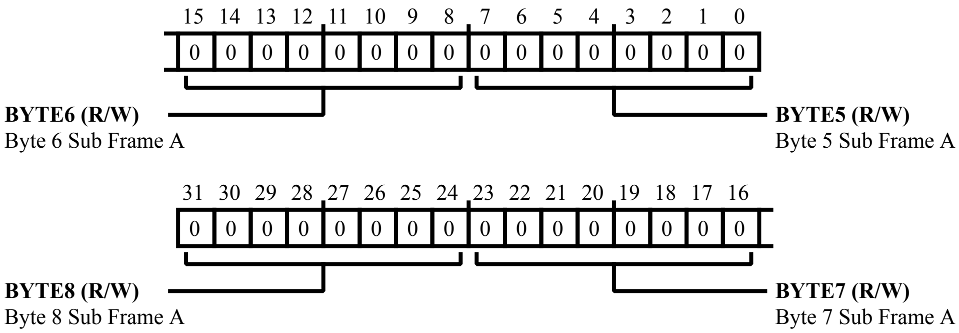

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE6 (R/W)
Byte 6 Sub Frame A
BYTE5 (R/W)
Byte 5 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE8 (R/W)
Byte 8 Sub Frame A
BYTE7 (R/W)
Byte 7 Sub Frame A

Figure 37-16: SPDIF_TX_STAT_A1 Register Diagram

Table 37-12: SPDIF_TX_STAT_A1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE8	Byte 8 Sub Frame A. The `SPDIF_TX_STAT_A1.BYTE8` bit field contains transmit status for byte 8 of sub frame A.
23:16 (R/W)	BYTE7	Byte 7 Sub Frame A. The `SPDIF_TX_STAT_A1.BYTE7` bit field contains transmit status for byte 7 of sub frame A.
15:8 (R/W)	BYTE6	Byte 6 Sub Frame A. The `SPDIF_TX_STAT_A1.BYTE6` bit field contains transmit status for byte 6 of sub frame A.
7:0 (R/W)	BYTE5	Byte 5 Sub Frame A. The `SPDIF_TX_STAT_A1.BYTE5` bit field contains transmit status for byte 5 of sub frame A.

Transmit Status A2 Register

The `SPDIF_TX_STAT_A2` register holds the transmit status for bytes 9-12 for sub frame A.

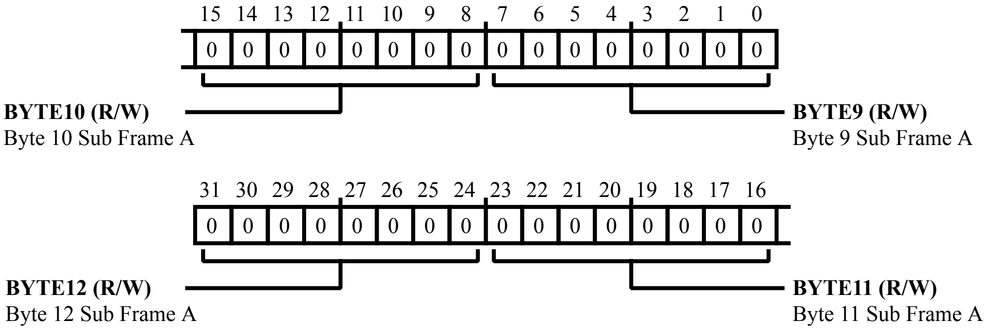

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE10 (R/W)
Byte 10 Sub Frame A
BYTE9 (R/W)
Byte 9 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE12 (R/W)
Byte 12 Sub Frame A
BYTE11 (R/W)
Byte 11 Sub Frame A

Figure 37-17: SPDIF_TX_STAT_A2 Register Diagram

Table 37-13: SPDIF_TX_STAT_A2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE12	Byte 12 Sub Frame A. The `SPDIF_TX_STAT_A2.BYTE12` bit field contains transmit status for byte 12 of sub frame A.
23:16 (R/W)	BYTE11	Byte 11 Sub Frame A. The `SPDIF_TX_STAT_A2.BYTE11` bit field contains transmit status for byte 11 of sub frame A.
15:8 (R/W)	BYTE10	Byte 10 Sub Frame A. The `SPDIF_TX_STAT_A2.BYTE10` bit field contains transmit status for byte 10 of sub frame A.
7:0 (R/W)	BYTE9	Byte 9 Sub Frame A. The `SPDIF_TX_STAT_A2.BYTE9` bit field contains transmit status for byte 9 of sub frame A.

Transmit Status A3 Register

The `SPDIF_TX_STAT_A3` register holds the transmit status for bytes 13-16 for sub frame A.

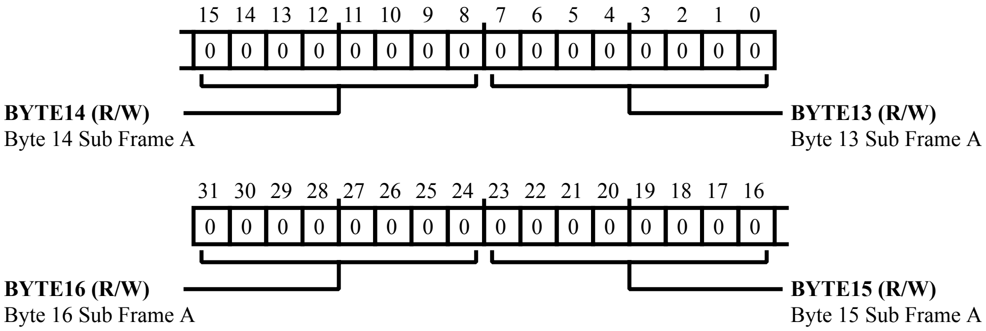

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE14 (R/W)
Byte 14 Sub Frame A
BYTE13 (R/W)
Byte 13 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE16 (R/W)
Byte 16 Sub Frame A
BYTE15 (R/W)
Byte 15 Sub Frame A

Figure 37-18: SPDIF_TX_STAT_A3 Register Diagram

Table 37-14: SPDIF_TX_STAT_A3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE16	Byte 16 Sub Frame A. The `SPDIF_TX_STAT_A3.BYTE16` bit field contains transmit status for byte 16 of sub frame A.
23:16 (R/W)	BYTE15	Byte 15 Sub Frame A. The `SPDIF_TX_STAT_A3.BYTE15` bit field contains transmit status for byte 15 of sub frame A.
15:8 (R/W)	BYTE14	Byte 14 Sub Frame A. The `SPDIF_TX_STAT_A3.BYTE14` bit field contains transmit status for byte 14 of sub frame A.
7:0 (R/W)	BYTE13	Byte 13 Sub Frame A. The `SPDIF_TX_STAT_A3.BYTE13` bit field contains transmit status for byte 13 of sub frame A.

Transmit Status A4 Register

The `SPDIF_TX_STAT_A4` register holds the transmit status for bytes 17-20 for sub frame A.

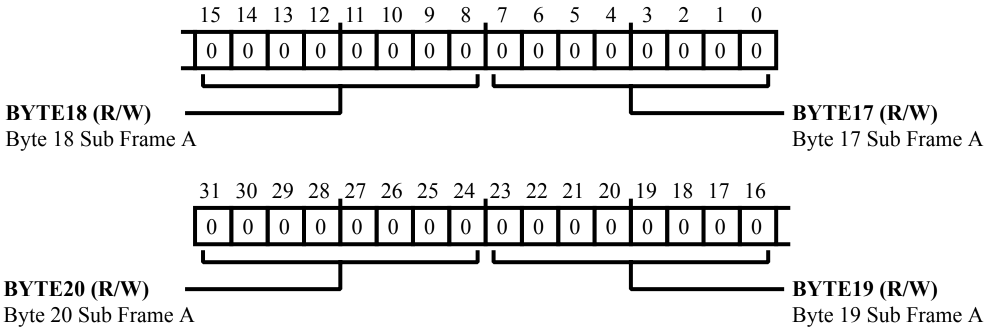

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE18 (R/W)
Byte 18 Sub Frame A
BYTE17 (R/W)
Byte 17 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE20 (R/W)
Byte 20 Sub Frame A
BYTE19 (R/W)
Byte 19 Sub Frame A

Figure 37-19: SPDIF_TX_STAT_A4 Register Diagram

Table 37-15: SPDIF_TX_STAT_A4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE20	Byte 20 Sub Frame A. The `SPDIF_TX_STAT_A4.BYTE20` bit field contains transmit status for byte 20 of sub frame A.
23:16 (R/W)	BYTE19	Byte 19 Sub Frame A. The `SPDIF_TX_STAT_A4.BYTE19` bit field contains transmit status for byte 19 of sub frame A.
15:8 (R/W)	BYTE18	Byte 18 Sub Frame A. The `SPDIF_TX_STAT_A4.BYTE18` bit field contains transmit status for byte 18 of sub frame A.
7:0 (R/W)	BYTE17	Byte 17 Sub Frame A. The `SPDIF_TX_STAT_A4.BYTE17` bit field contains transmit status for byte 17 of sub frame A.

Transmit Status A5 Register

The SPDIF_TX_STAT_A5 register holds the transmit status for bytes 21-23 for sub frame A.

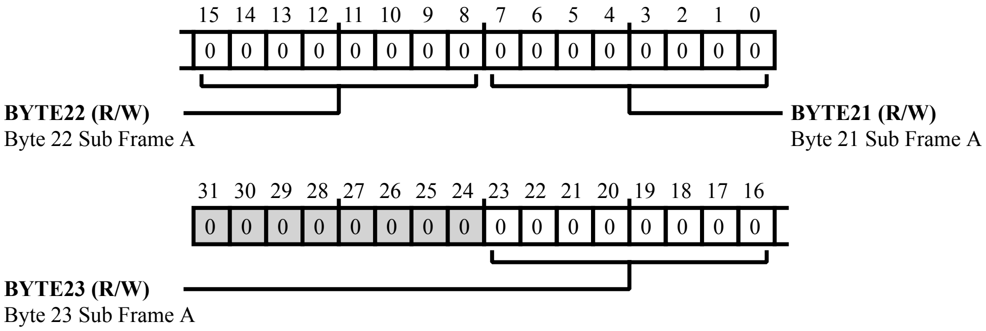

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE22 (R/W)
Byte 22 Sub Frame A
BYTE21 (R/W)
Byte 21 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE23 (R/W)
Byte 23 Sub Frame A

Figure 37-20: SPDIF_TX_STAT_A5 Register Diagram

Table 37-16: SPDIF_TX_STAT_A5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	BYTE23	Byte 23 Sub Frame A. The `SPDIF_TX_STAT_A5.BYTE23` bit field contains transmit status for byte 23 of sub frame A.
15:8 (R/W)	BYTE22	Byte 22 Sub Frame A. The `SPDIF_TX_STAT_A5.BYTE22` bit field contains transmit status for byte 22 of sub frame A.
7:0 (R/W)	BYTE21	Byte 21 Sub Frame A. The `SPDIF_TX_STAT_A5.BYTE21` bit field contains transmit status for byte 21 of sub frame A.

Transmit Status B0 Register

The `SPDIF_TX_STAT_B0` register holds the transmit channel 0 status for bytes 1-4 for sub frame B.

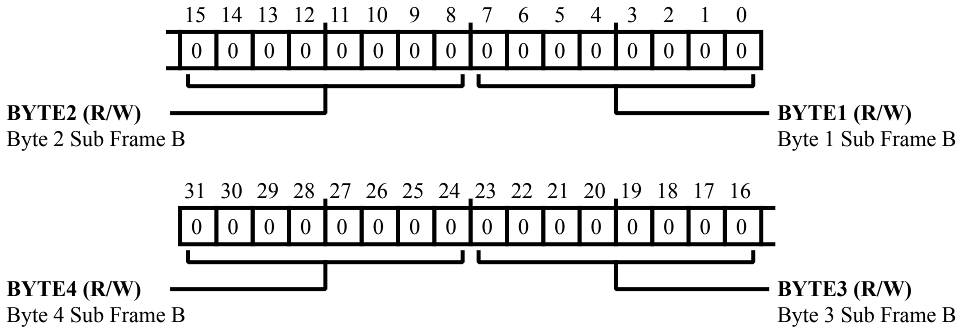

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE2 (R/W)
Byte 2 Sub Frame B
BYTE1 (R/W)
Byte 1 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE4 (R/W)
Byte 4 Sub Frame B
BYTE3 (R/W)
Byte 3 Sub Frame B

Figure 37-21: SPDIF_TX_STAT_B0 Register Diagram

Table 37-17: SPDIF_TX_STAT_B0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE4	Byte 4 Sub Frame B. The `SPDIF_TX_STAT_B0.BYTE4` bit field holds the transmit channel 0 status for byte 4 for sub frame B.
23:16 (R/W)	BYTE3	Byte 3 Sub Frame B. The `SPDIF_TX_STAT_B0.BYTE3` bit field holds the transmit channel 0 status for byte 3 for sub frame B.
15:8 (R/W)	BYTE2	Byte 2 Sub Frame B. The `SPDIF_TX_STAT_B0.BYTE2` bit field holds the transmit channel 0 status for byte 2 for sub frame B.
7:0 (R/W)	BYTE1	Byte 1 Sub Frame B. The `SPDIF_TX_STAT_B0.BYTE1` bit field holds the transmit channel 0 status for byte 1 for sub frame B.

Transmit Status B1 Register

The `SPDIF_TX_STAT_B1` register holds the transmit status for bytes 5-8 for sub frame B.

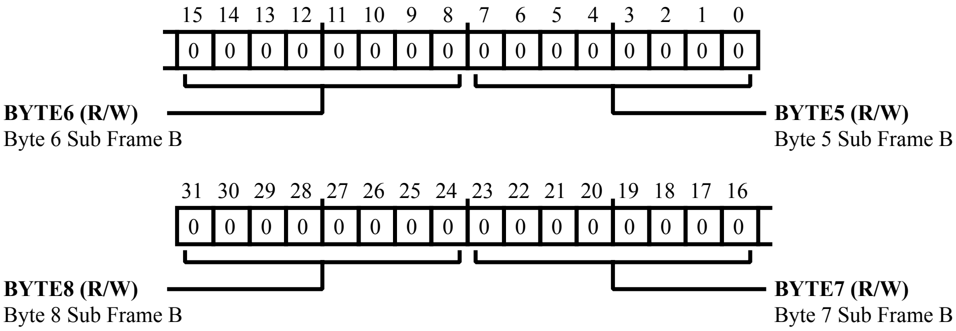

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE6 (R/W)
Byte 6 Sub Frame B
BYTE5 (R/W)
Byte 5 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE8 (R/W)
Byte 8 Sub Frame B
BYTE7 (R/W)
Byte 7 Sub Frame B

Figure 37-22: SPDIF_TX_STAT_B1 Register Diagram

Table 37-18: SPDIF_TX_STAT_B1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE8	Byte 8 Sub Frame B. The `SPDIF_TX_STAT_B1.BYTE8` bit field contains transmit status for byte 8 of sub frame B.
23:16 (R/W)	BYTE7	Byte 7 Sub Frame B. The `SPDIF_TX_STAT_B1.BYTE7` bit field contains transmit status for byte 7 of sub frame B.
15:8 (R/W)	BYTE6	Byte 6 Sub Frame B. The `SPDIF_TX_STAT_B1.BYTE6` bit field contains transmit status for byte 6 of sub frame B.
7:0 (R/W)	BYTE5	Byte 5 Sub Frame B. The `SPDIF_TX_STAT_B1.BYTE5` bit field contains transmit status for byte 5 of sub frame B.

Transmit Status B2 Register

The `SPDIF_TX_STAT_B2` register holds the transmit status for bytes 9-12 for sub frame B.

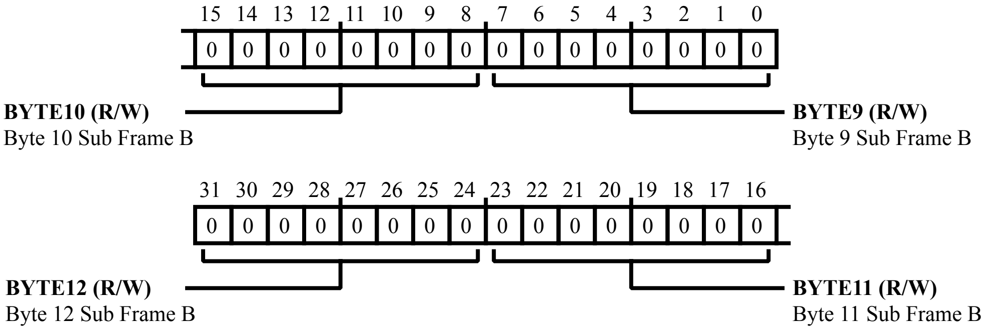

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE10 (R/W)
Byte 10 Sub Frame B
BYTE9 (R/W)
Byte 9 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE12 (R/W)
Byte 12 Sub Frame B
BYTE11 (R/W)
Byte 11 Sub Frame B

Figure 37-23: SPDIF_TX_STAT_B2 Register Diagram

Table 37-19: SPDIF_TX_STAT_B2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE12	Byte 12 Sub Frame B. The `SPDIF_TX_STAT_B2.BYTE12` bit field contains transmit status for byte 12 of sub frame B.
23:16 (R/W)	BYTE11	Byte 11 Sub Frame B. The `SPDIF_TX_STAT_B2.BYTE11` bit field contains transmit status for byte 11 of sub frame B.
15:8 (R/W)	BYTE10	Byte 10 Sub Frame B. The `SPDIF_TX_STAT_B2.BYTE10` bit field contains transmit status for byte 10 of sub frame B.
7:0 (R/W)	BYTE9	Byte 9 Sub Frame B. The `SPDIF_TX_STAT_B2.BYTE9` bit field contains transmit status for byte 9 of sub frame B.

Transmit Status B3 Register

The `SPDIF_TX_STAT_B3` register holds the transmit status for bytes 13-16 for sub frame B.

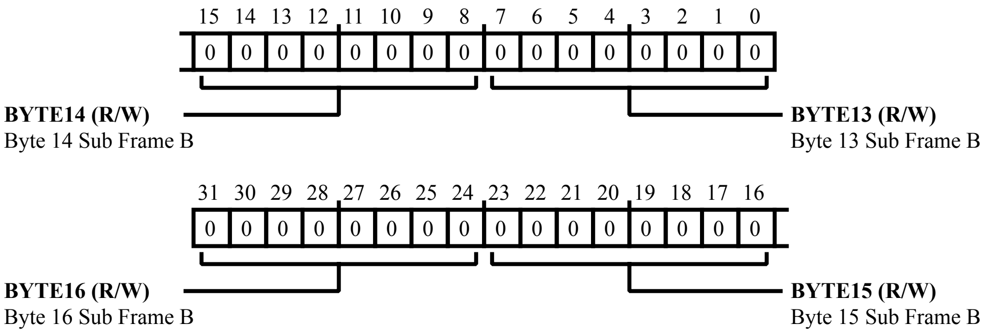

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE14 (R/W)
Byte 14 Sub Frame B
BYTE13 (R/W)
Byte 13 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE16 (R/W)
Byte 16 Sub Frame B
BYTE15 (R/W)
Byte 15 Sub Frame B

Figure 37-24: SPDIF_TX_STAT_B3 Register Diagram

Table 37-20: SPDIF_TX_STAT_B3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE16	Byte 16 Sub Frame B. The `SPDIF_TX_STAT_B3.BYTE16` bit field contains transmit status for byte 16 of sub frame B.
23:16 (R/W)	BYTE15	Byte 15 Sub Frame B. The `SPDIF_TX_STAT_B3.BYTE15` bit field contains transmit status for byte 15 of sub frame B.
15:8 (R/W)	BYTE14	Byte 14 Sub Frame B. The `SPDIF_TX_STAT_B3.BYTE14` bit field contains transmit status for byte 14 of sub frame B.
7:0 (R/W)	BYTE13	Byte 13 Sub Frame B. The `SPDIF_TX_STAT_B3.BYTE13` bit field contains transmit status for byte 13 of sub frame B.

Transmit Status B4 Register

The `SPDIF_TX_STAT_B4` register holds the transmit status for bytes 17-20 for sub frame B.

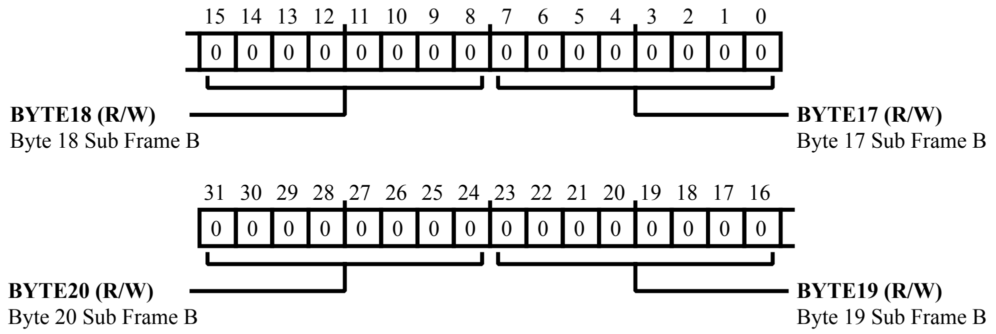

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE18 (R/W)
Byte 18 Sub Frame B
BYTE17 (R/W)
Byte 17 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE20 (R/W)
Byte 20 Sub Frame B
BYTE19 (R/W)
Byte 19 Sub Frame B

Figure 37-25: SPDIF_TX_STAT_B4 Register Diagram

Table 37-21: SPDIF_TX_STAT_B4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE20	Byte 20 Sub Frame B. The `SPDIF_TX_STAT_B4.BYTE20` bit field contains transmit status for byte 20 of sub frame B.
23:16 (R/W)	BYTE19	Byte 19 Sub Frame B. The `SPDIF_TX_STAT_B4.BYTE19` bit field contains transmit status for byte 19 of sub frame B.
15:8 (R/W)	BYTE18	Byte 18 Sub Frame B. The `SPDIF_TX_STAT_B4.BYTE18` bit field contains transmit status for byte 18 of sub frame B.
7:0 (R/W)	BYTE17	Byte 17 Sub Frame B. The `SPDIF_TX_STAT_B4.BYTE17` bit field contains transmit status for byte 17 of sub frame B.

Transmit Status B5 Register

The `SPDIF_TX_STAT_B5` register holds the transmit status for bytes 21-23 for sub frame B.

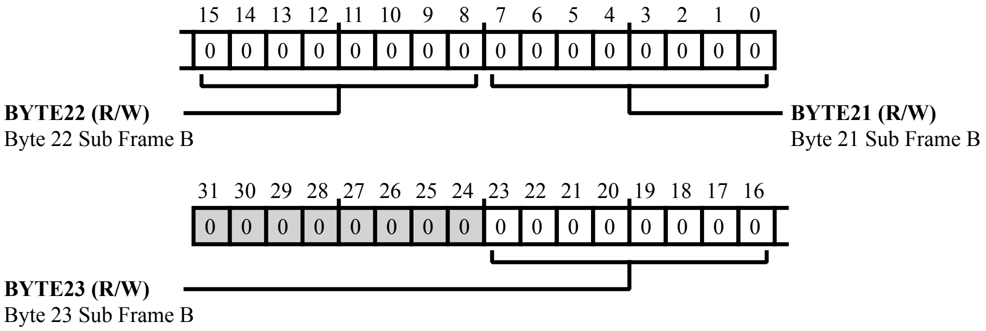

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE22 (R/W)
Byte 22 Sub Frame B
BYTE21 (R/W)
Byte 21 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE23 (R/W)
Byte 23 Sub Frame B

Figure 37-26: SPDIF_TX_STAT_B5 Register Diagram

Table 37-22: SPDIF_TX_STAT_B5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	BYTE23	Byte 23 Sub Frame B. The `SPDIF_TX_STAT_B5.BYTE23` bit field contains transmit status for byte 23 of sub frame B.
15:8 (R/W)	BYTE22	Byte 22 Sub Frame B. The `SPDIF_TX_STAT_B5.BYTE22` bit field contains transmit status for byte 22 of sub frame B.
7:0 (R/W)	BYTE21	Byte 21 Sub Frame B. The `SPDIF_TX_STAT_B5.BYTE21` bit field contains transmit status for byte 21 of sub frame B.

Transmit User Buffer A0 Register

The `SPDIF_TX_UBUFF_A0` register holds the transmit user buffer data for bytes 0-3 for sub frame A.

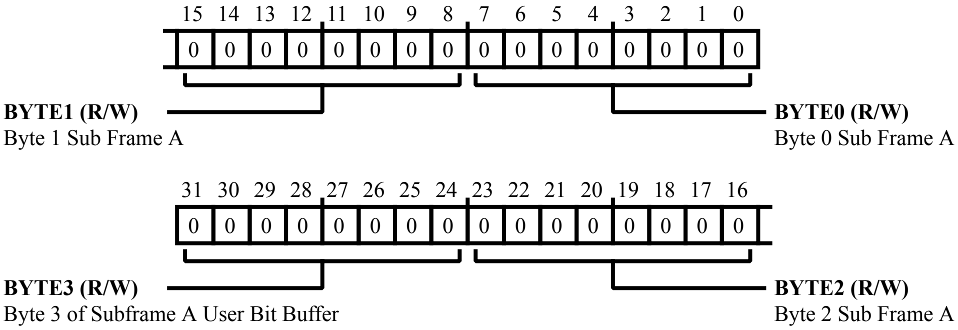

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE1 (R/W)
Byte 1 Sub Frame A
BYTE0 (R/W)
Byte 0 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE3 (R/W)
Byte 3 of Subframe A User Bit Buffer
BYTE2 (R/W)
Byte 2 Sub Frame A

Figure 37-27: SPDIF_TX_UBUFF_A0 Register Diagram

Table 37-23: SPDIF_TX_UBUFF_A0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE3	Byte 3 of Subframe A User Bit Buffer. The `SPDIF_TX_UBUFF_A0.BYTE3` bit field contains user bit data for byte 0 of sub frame A.
23:16 (R/W)	BYTE2	Byte 2 Sub Frame A. The `SPDIF_TX_UBUFF_A0.BYTE2` bit field contains user bit data for byte 2 of sub frame A.
15:8 (R/W)	BYTE1	Byte 1 Sub Frame A. The `SPDIF_TX_UBUFF_A0.BYTE1` bit field contains user bit data for byte 1 of sub frame A.
7:0 (R/W)	BYTE0	Byte 0 Sub Frame A. The `SPDIF_TX_UBUFF_A0.BYTE0` bit field contains user bit data for byte 0 of sub frame A.

Transmit User Buffer A1 Register

The `SPDIF_TX_UBUFF_A1` register holds the transmit user buffer data for bytes 4-7 for sub frame A.

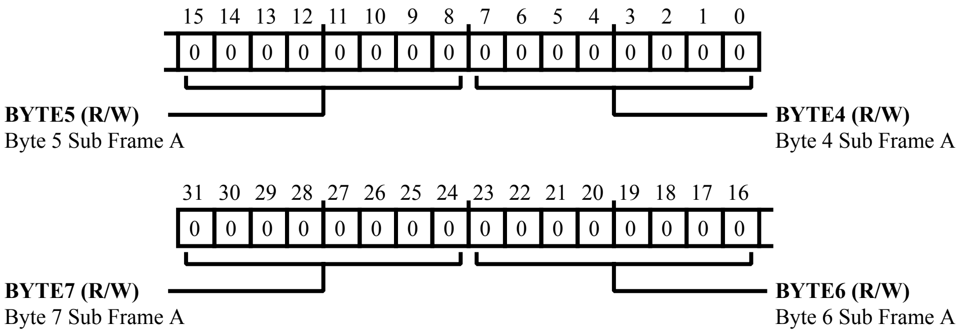

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE5 (R/W)
Byte 5 Sub Frame A
BYTE4 (R/W)
Byte 4 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE7 (R/W)
Byte 7 Sub Frame A
BYTE6 (R/W)
Byte 6 Sub Frame A

Figure 37-28: SPDIF_TX_UBUFF_A1 Register Diagram

Table 37-24: SPDIF_TX_UBUFF_A1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE7	Byte 7 Sub Frame A. The `SPDIF_TX_UBUFF_A1.BYTE7` bit field contains user bit data for byte 7 of sub frame A.
23:16 (R/W)	BYTE6	Byte 6 Sub Frame A. The `SPDIF_TX_UBUFF_A1.BYTE6` bit field contains user bit data for byte 6 of sub frame A.
15:8 (R/W)	BYTE5	Byte 5 Sub Frame A. The `SPDIF_TX_UBUFF_A1.BYTE5` bit field contains user bit data for byte 5 of sub frame A.
7:0 (R/W)	BYTE4	Byte 4 Sub Frame A. The `SPDIF_TX_UBUFF_A1.BYTE4` bit field contains user bit data for byte 0 of sub frame A.

Transmit User Buffer A2 Register

The `SPDIF_TX_UBUFF_A2` register holds the transmit user buffer data for bytes 8-11 for sub frame A.

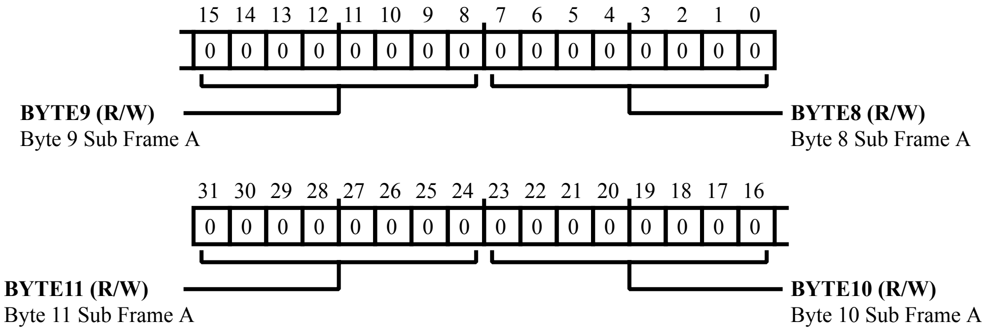

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE9 (R/W)
Byte 9 Sub Frame A
BYTE8 (R/W)
Byte 8 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE11 (R/W)
Byte 11 Sub Frame A
BYTE10 (R/W)
Byte 10 Sub Frame A

Figure 37-29: SPDIF_TX_UBUFF_A2 Register Diagram

Table 37-25: SPDIF_TX_UBUFF_A2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE11	Byte 11 Sub Frame A. The `SPDIF_TX_UBUFF_A2.BYTE11` bit field contains user bit data for byte 11 of sub frame A.
23:16 (R/W)	BYTE10	Byte 10 Sub Frame A. The `SPDIF_TX_UBUFF_A2.BYTE10` bit field contains user bit data for byte 10 of sub frame A.
15:8 (R/W)	BYTE9	Byte 9 Sub Frame A. The `SPDIF_TX_UBUFF_A2.BYTE9` bit field contains user bit data for byte 9 of sub frame A.
7:0 (R/W)	BYTE8	Byte 8 Sub Frame A. The `SPDIF_TX_UBUFF_A2.BYTE8` bit field contains user bit data for byte 8 of sub frame A.

Transmit User Buffer A3 Register

The `SPDIF_TX_UBUFF_A3` register holds the transmit user buffer data for bytes 12-15 for sub frame A.

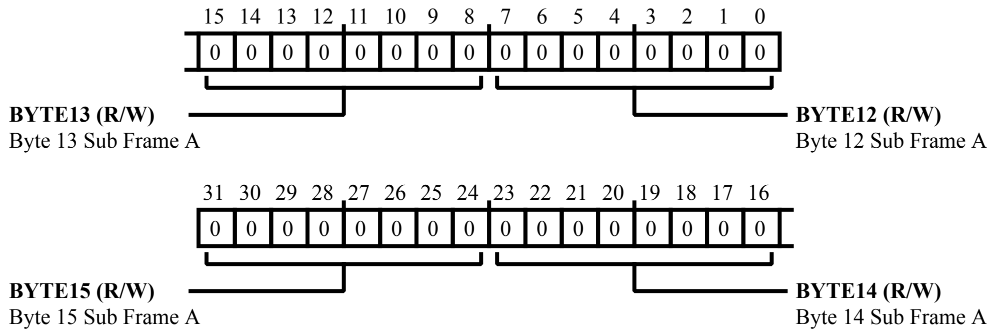

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE13 (R/W)
Byte 13 Sub Frame A
BYTE12 (R/W)
Byte 12 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE15 (R/W)
Byte 15 Sub Frame A
BYTE14 (R/W)
Byte 14 Sub Frame A

Figure 37-30: SPDIF_TX_UBUFF_A3 Register Diagram

Table 37-26: SPDIF_TX_UBUFF_A3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE15	Byte 15 Sub Frame A. The `SPDIF_TX_UBUFF_A3.BYTE15` bit field contains user bit data for byte 15 of sub frame A.
23:16 (R/W)	BYTE14	Byte 14 Sub Frame A. The `SPDIF_TX_UBUFF_A3.BYTE14` bit field contains user bit data for byte 14 of sub frame A.
15:8 (R/W)	BYTE13	Byte 13 Sub Frame A. The `SPDIF_TX_UBUFF_A3.BYTE13` bit field contains user bit data for byte 13 of sub frame A.
7:0 (R/W)	BYTE12	Byte 12 Sub Frame A. The `SPDIF_TX_UBUFF_A3.BYTE12` bit field contains user bit data for byte 12 of sub frame A.

Transmit User Buffer A4 Register

The `SPDIF_TX_UBUFF_A4` register holds the transmit user buffer data for bytes 16-19 for sub frame A.

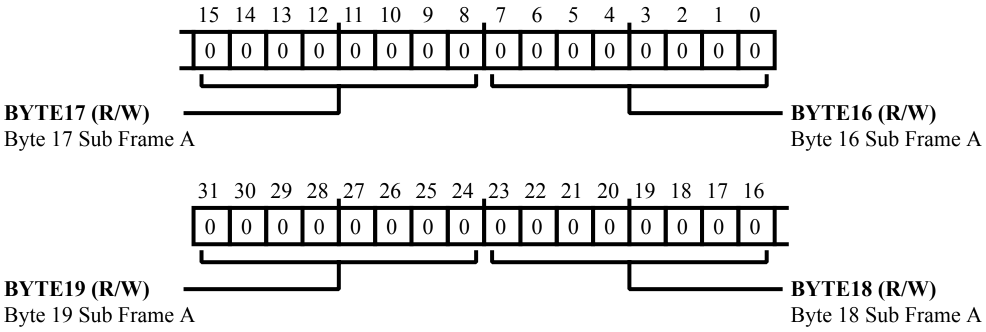

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE17 (R/W)
Byte 17 Sub Frame A
BYTE16 (R/W)
Byte 16 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE19 (R/W)
Byte 19 Sub Frame A
BYTE18 (R/W)
Byte 18 Sub Frame A

Figure 37-31: SPDIF_TX_UBUFF_A4 Register Diagram

Table 37-27: SPDIF_TX_UBUFF_A4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE19	Byte 19 Sub Frame A. The `SPDIF_TX_UBUFF_A4.BYTE19` bit field contains user bit data for byte 19 of sub frame A.
23:16 (R/W)	BYTE18	Byte 18 Sub Frame A. The `SPDIF_TX_UBUFF_A4.BYTE18` bit field contains user bit data for byte 18 of sub frame A.
15:8 (R/W)	BYTE17	Byte 17 Sub Frame A. The `SPDIF_TX_UBUFF_A4.BYTE17` bit field contains user bit data for byte 17 of sub frame A.
7:0 (R/W)	BYTE16	Byte 16 Sub Frame A. The `SPDIF_TX_UBUFF_A4.BYTE16` bit field contains user bit data for byte 16 of sub frame A.

Transmit User Buffer A5 Register

The `SPDIF_TX_UBUFF_A5` register holds the transmit user buffer data for bytes 20-23 for sub frame A.

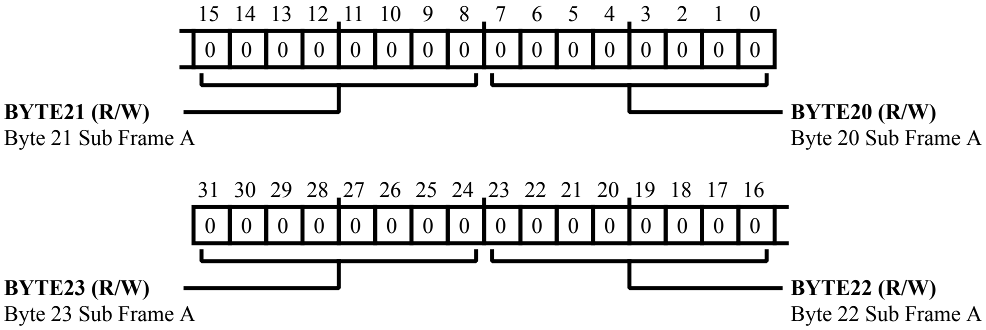

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE21 (R/W)
Byte 21 Sub Frame A
BYTE20 (R/W)
Byte 20 Sub Frame A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE23 (R/W)
Byte 23 Sub Frame A
BYTE22 (R/W)
Byte 22 Sub Frame A

Figure 37-32: SPDIF_TX_UBUFF_A5 Register Diagram

Table 37-28: SPDIF_TX_UBUFF_A5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE23	Byte 23 Sub Frame A. The `SPDIF_TX_UBUFF_A5.BYTE23` bit field contains user bit data for byte 23 of sub frame A.
23:16 (R/W)	BYTE22	Byte 22 Sub Frame A. The `SPDIF_TX_UBUFF_A5.BYTE22` bit field contains user bit data for byte 22 of sub frame A.
15:8 (R/W)	BYTE21	Byte 21 Sub Frame A. The `SPDIF_TX_UBUFF_A5.BYTE21` bit field contains user bit data for byte 21 of sub frame A.
7:0 (R/W)	BYTE20	Byte 20 Sub Frame A. The `SPDIF_TX_UBUFF_A5.BYTE20` bit field contains user bit data for byte 20 of sub frame A.

Transmit User Buffer B0 Register

The `SPDIF_TX_UBUFF_B0` register holds the transmit user buffer data for bytes 0-3 for sub frame B.

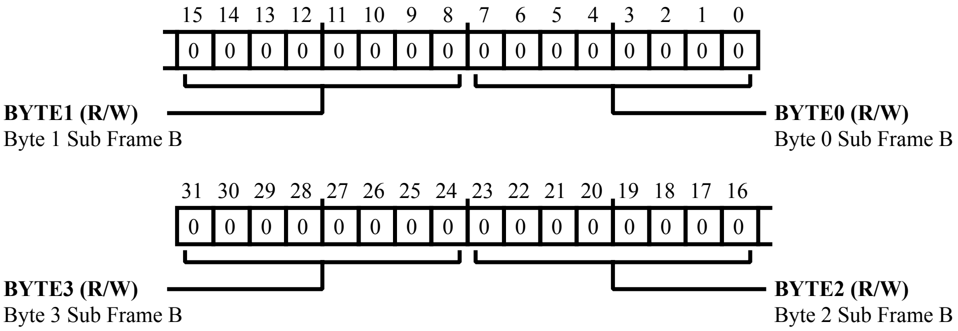

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE1 (R/W)
Byte 1 Sub Frame B
BYTE0 (R/W)
Byte 0 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE3 (R/W)
Byte 3 Sub Frame B
BYTE2 (R/W)
Byte 2 Sub Frame B

Figure 37-33: SPDIF_TX_UBUFF_B0 Register Diagram

Table 37-29: SPDIF_TX_UBUFF_B0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE3	Byte 3 Sub Frame B. The `SPDIF_TX_UBUFF_B0.BYTE3` bit field contains user bit data for byte 3 of sub frame B.
23:16 (R/W)	BYTE2	Byte 2 Sub Frame B. The `SPDIF_TX_UBUFF_B0.BYTE2` bit field contains user bit data for byte 2 of sub frame B.
15:8 (R/W)	BYTE1	Byte 1 Sub Frame B. The `SPDIF_TX_UBUFF_B0.BYTE1` bit field contains user bit data for byte 1 of sub frame B.
7:0 (R/W)	BYTE0	Byte 0 Sub Frame B. The `SPDIF_TX_UBUFF_B0.BYTE0` bit field contains user bit data for byte 0 of sub frame B.

Transmit User Buffer B1 Register

The `SPDIF_TX_UBUFF_B1` register holds the transmit user buffer data for bytes 4-7 for sub frame B.

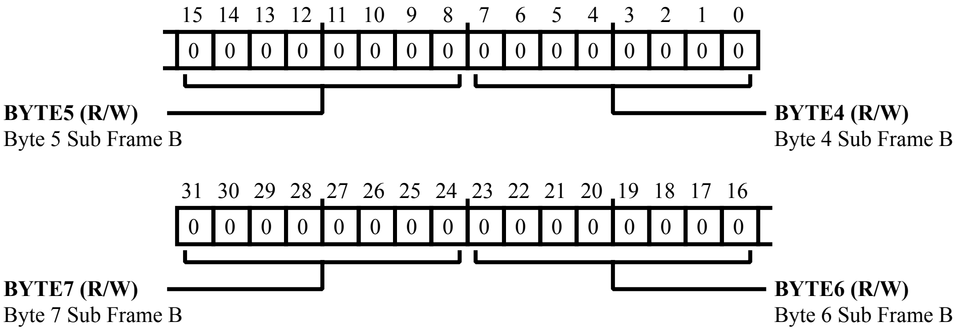

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE5 (R/W)
Byte 5 Sub Frame B
BYTE4 (R/W)
Byte 4 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE7 (R/W)
Byte 7 Sub Frame B
BYTE6 (R/W)
Byte 6 Sub Frame B

Figure 37-34: SPDIF_TX_UBUFF_B1 Register Diagram

Table 37-30: SPDIF_TX_UBUFF_B1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE7	Byte 7 Sub Frame B. The `SPDIF_TX_UBUFF_B1.BYTE7` bit field contains user bit data for byte 7 of sub frame B.
23:16 (R/W)	BYTE6	Byte 6 Sub Frame B. The `SPDIF_TX_UBUFF_B1.BYTE6` bit field contains user bit data for byte 6 of sub frame B.
15:8 (R/W)	BYTE5	Byte 5 Sub Frame B. The `SPDIF_TX_UBUFF_B1.BYTE5` bit field contains user bit data for byte 5 of sub frame B.
7:0 (R/W)	BYTE4	Byte 4 Sub Frame B. The `SPDIF_TX_UBUFF_B1.BYTE4` bit field contains user bit data for byte 4 of sub frame B.

Transmit User Buffer B2 Register

The `SPDIF_TX_UBUFF_B2` register holds the transmit user buffer data for bytes 8-11 for sub frame B.

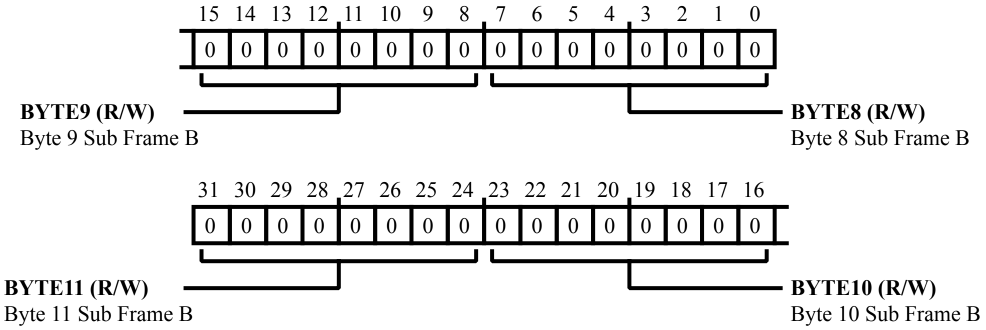

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE9 (R/W)
Byte 9 Sub Frame B
BYTE8 (R/W)
Byte 8 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE11 (R/W)
Byte 11 Sub Frame B
BYTE10 (R/W)
Byte 10 Sub Frame B

Figure 37-35: SPDIF_TX_UBUFF_B2 Register Diagram

Table 37-31: SPDIF_TX_UBUFF_B2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE11	Byte 11 Sub Frame B. The `SPDIF_TX_UBUFF_B2.BYTE11` bit field contains user bit data for byte 11 of sub frame B.
23:16 (R/W)	BYTE10	Byte 10 Sub Frame B. The `SPDIF_TX_UBUFF_B2.BYTE10` bit field contains user bit data for byte 10 of sub frame B.
15:8 (R/W)	BYTE9	Byte 9 Sub Frame B. The `SPDIF_TX_UBUFF_B2.BYTE9` bit field contains user bit data for byte 9 of sub frame B.
7:0 (R/W)	BYTE8	Byte 8 Sub Frame B. The `SPDIF_TX_UBUFF_B2.BYTE8` bit field contains user bit data for byte 8 of sub frame B.

Transmit User Buffer B3 Register

The `SPDIF_TX_UBUFF_B3` register holds the transmit user buffer data for bytes 12-15 for sub frame B.

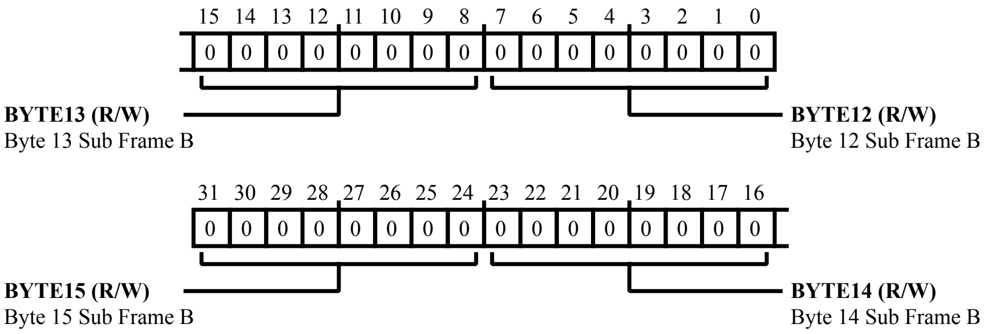

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE13 (R/W)
Byte 13 Sub Frame B
BYTE12 (R/W)
Byte 12 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE15 (R/W)
Byte 15 Sub Frame B
BYTE14 (R/W)
Byte 14 Sub Frame B

Figure 37-36: SPDIF_TX_UBUFF_B3 Register Diagram

Table 37-32: SPDIF_TX_UBUFF_B3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE15	Byte 15 Sub Frame B. The `SPDIF_TX_UBUFF_B3.BYTE15` bit field contains user bit data for byte 15 of sub frame B.
23:16 (R/W)	BYTE14	Byte 14 Sub Frame B. The `SPDIF_TX_UBUFF_B3.BYTE14` bit field contains user bit data for byte 14 of sub frame B.
15:8 (R/W)	BYTE13	Byte 13 Sub Frame B. The `SPDIF_TX_UBUFF_B3.BYTE13` bit field contains user bit data for byte 13 of sub frame B.
7:0 (R/W)	BYTE12	Byte 12 Sub Frame B. The `SPDIF_TX_UBUFF_B3.BYTE12` bit field contains user bit data for byte 12 of sub frame B.

Transmit User Buffer B4 Register

The `SPDIF_TX_UBUFF_B4` register holds the transmit user buffer data for bytes 16-19 for sub frame B.

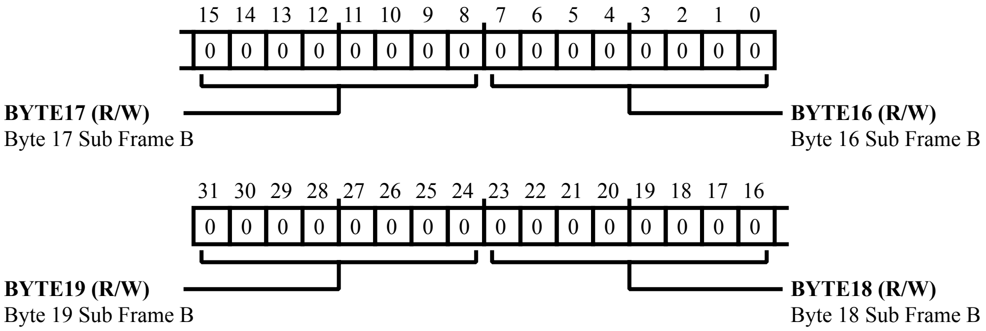

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE17 (R/W)
Byte 17 Sub Frame B
BYTE16 (R/W)
Byte 16 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE19 (R/W)
Byte 19 Sub Frame B
BYTE18 (R/W)
Byte 18 Sub Frame B

Figure 37-37: SPDIF_TX_UBUFF_B4 Register Diagram

Table 37-33: SPDIF_TX_UBUFF_B4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE19	Byte 19 Sub Frame B. The `SPDIF_TX_UBUFF_B4.BYTE19` bit field contains user bit data for byte 19 of sub frame B.
23:16 (R/W)	BYTE18	Byte 18 Sub Frame B. The `SPDIF_TX_UBUFF_B4.BYTE18` bit field contains user bit data for byte 18 of sub frame B.
15:8 (R/W)	BYTE17	Byte 17 Sub Frame B. The `SPDIF_TX_UBUFF_B4.BYTE17` bit field contains user bit data for byte 17 of sub frame B.
7:0 (R/W)	BYTE16	Byte 16 Sub Frame B. The `SPDIF_TX_UBUFF_B4.BYTE16` bit field contains user bit data for byte 16 of sub frame B.

Transmit User Buffer B5 Register

The SPDIF_TX_UBUFF_B5 register holds the transmit user buffer data for bytes 20-23 for sub frame B.

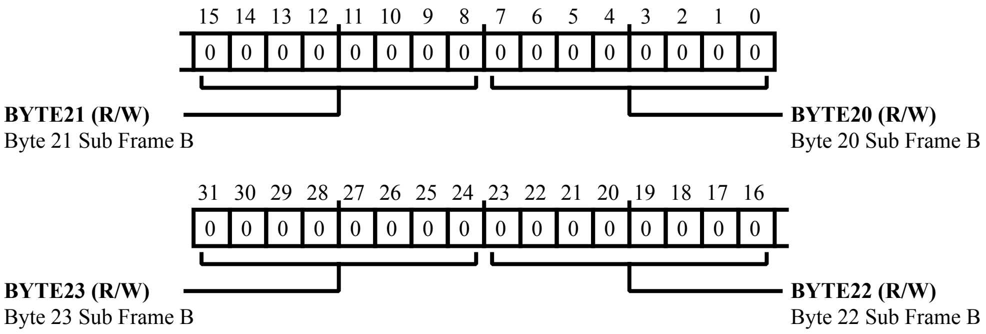

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE21 (R/W)
Byte 21 Sub Frame B
BYTE20 (R/W)
Byte 20 Sub Frame B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTE23 (R/W)
Byte 23 Sub Frame B
BYTE22 (R/W)
Byte 22 Sub Frame B

Figure 37-38: SPDIF_TX_UBUFF_B5 Register Diagram

Table 37-34: SPDIF_TX_UBUFF_B5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYTE23	Byte 23 Sub Frame B. The `SPDIF_TX_UBUFF_B5.BYTE23` bit field contains user bit data for byte 23 of sub frame B.
23:16 (R/W)	BYTE22	Byte 22 Sub Frame B. The `SPDIF_TX_UBUFF_B5.BYTE22` bit field contains user bit data for byte 22 of sub frame B.
15:8 (R/W)	BYTE21	Byte 21 Sub Frame B. The `SPDIF_TX_UBUFF_B5.BYTE21` bit field contains user bit data for byte 21 of sub frame B.
7:0 (R/W)	BYTE20	Byte 20 Sub Frame B. The `SPDIF_TX_UBUFF_B5.BYTE20` bit field contains user bit data for byte 20 of sub frame B.

User Bit Update Register

After writing to the transmit user buffer registers, a value of 0x1 must be written into the `SPDIF_TX_USRUPDT` register to enable the use of these user buffer bits in the next transfer block.

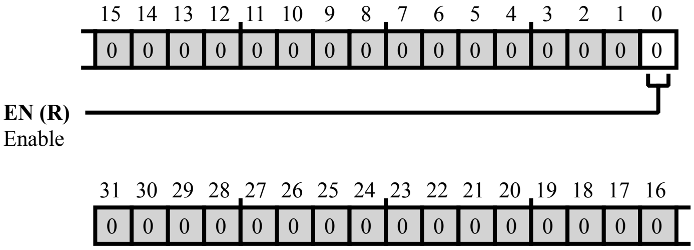

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R)
Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 37-39: SPDIF_TX_USRUPDT Register Diagram

Table 37-35: SPDIF_TX_USRUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	EN	Enable. After writing to the transmit user buffer registers, a value of 0x1 must be written into the `SPDIF_TX_USRUPDT` register to enable the use of these `SPDIF_TX_USRUPDT.EN` bits in the next transfer block.

38 Direct Memory Access (DMA)

The processor architecture distributes the DMA channels throughout the infrastructure. Often, the channels cluster together through system crossbars (SCB), sharing a single interface with the main system crossbar.

The DMA channels can perform transfers between memory and a peripheral or between one memory and another memory. Memory-to-memory DMA transfers (MDMA) require two DMA channels. One channel is the source channel, and the second, the destination channel.

All DMA channels can transport data to and from virtually all on-chip and off-chip memories.

DMA transfers on the processor use either a descriptor-based method or register-based method. Register-based DMA allows the processor directly to program DMA controller registers to initiate a DMA transfer. On completion, the controller registers can automatically update with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based transfers allow the chaining together of multiple DMA sequences. In descriptor-based DMA operations, DMA channel programming can automatically set up and start another DMA transfer after the current sequence completes.

The DMA channel does not connect external memories and devices directly. Rather, data passes through an external-memory interface port. DMA operations can access any device the external memory interface supports. These interfaces typically include:

- Flash memory
- SRAM
- FIFOs
- Memory-mapped peripheral devices
- Dynamic Memory (if present)

DMA Channel Features

The processor uses Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure and interface with the system crossbar unit (SCB).

The following is a list of DMA interface features.

- Supports integer byte strides including byte strides of 0 and negative byte strides
- Register-based configuration
    - Core writes DMA configuration
    - Supports automatic reloading for continuous operation
- Flexible descriptor-based configuration
    - DMA descriptors are fetched from memory
    - Support for variable descriptor sizes
- Flexible flow control – Transitions between the various descriptor-based modes and for DMA termination
- Orthogonal transfers
    - Support for three transfer dimensions
    - 1D and 2D transfers supported per descriptor set
    - 3D support provided by chained descriptor sets
- Configurable memory and peripheral-transfer word sizes
    - Memory interface supports 8-bit, 16-bit, 32-bit, 64-bit, 128-bit, and 256-bit transfers
    - Peripheral interface supports for 8-bit, 16-bit, and 32-bit transfers
- Interrupt notification
    - Row or work unit completion
    - Error conditions
- Incoming and outgoing trigger support
    - Trigger generation for row or work unit completion
    - Work unit can wait for incoming trigger
- MMR access bus – Provides access to memory-mapped registers for configuration, monitoring, and debug
- SCB crossbar interface connects the DMA channel to the system crossbar
- Peripheral DMA bus – Interfaces the DMA channel to a peripheral or another DMA channel
- Peripheral data-request interrupt support
- Bandwidth monitoring and limiting for MDMA channels

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

NOTE: There are two types of peripherals that use DMA. The first have dedicated DMA channels controlled by the Dedicated DMA Engine (DDE) and are described in this chapter. The second type is not controlled by the DDE. These peripherals have their own operating modes and programming models (see the peripheral chapter for this information). The complete list of DMA supported peripherals are shown in Table 54-4 DMA Channel Peripherals Controlled by SCBs.

ADSP-SC58x DMA Register List

The DMA channel controller (DMA) supports data transfers within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure, as DMAn's. A set of registers governs DMA operations. For more information on DMA functionality, see the DMA register descriptions.

Table 38-1: ADSP-SC58x DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

ADSP-SC58x DMA Channel List

Table 38-2: ADSP-SC58x DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA1	SPORT0_B_DMA	SPORT0 Channel B DMA
DMA2	SPORT1_A_DMA	SPORT1 Channel A DMA
DMA3	SPORT1_B_DMA	SPORT1 Channel B DMA
DMA4	SPORT2_A_DMA	SPORT2 Channel A DMA
DMA5	SPORT2_B_DMA	SPORT2 Channel B DMA
DMA6	SPORT3_A_DMA	SPORT3 Channel A DMA
DMA7	SPORT3_B_DMA	SPORT3 Channel B DMA
DMA8	MDMA0_SRC	Memory DMA Stream n Source Channel
DMA9	MDMA0_DST	Memory DMA Stream n Destination Channel
DMA10	SPORT4_A_DMA	SPORT4 Channel A DMA
DMA11	SPORT4_B_DMA	SPORT4 Channel B DMA
DMA12	SPORT5_A_DMA	SPORT5 Channel A DMA
DMA13	SPORT5_B_DMA	SPORT5 Channel B DMA
DMA14	SPORT6_A_DMA	SPORT6 Channel A DMA
DMA15	SPORT6_B_DMA	SPORT6 Channel B DMA
DMA16	SPORT7_A_DMA	SPORT7 Channel A DMA
DMA17	SPORT7_B_DMA	SPORT7 Channel B DMA
DMA18	MDMA1_SRC	Memory DMA Stream n Source Channel
DMA19	MDMA1_DST	Memory DMA Stream n Destination Channel
DMA20	UART0_TXDMA	UART0 Transmit DMA
DMA21	UART0_RXDMA	UART0 Receive DMA
DMA22	SPI0_TXDMA	SPI0 TX DMA Channel
DMA23	SPI0_RXDMA	SPI0 RX DMA Channel
DMA24	SPI1_TXDMA	SPI1 TX DMA Channel
DMA25	SPI1_RXDMA	SPI1 RX DMA Channel
DMA26	SPI2_TXDMA	SPI2 TX DMA Channel

Table 38-2: ADSP-SC58x DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA27	SPI2_RXDMA	SPI2 RX DMA Channel
DMA28	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA
DMA29	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA
DMA30	LP0_DMA	LP0 DMA Channel
DMA31	HAE0_TXDMA	HAE0 TX DMA Channel
DMA32	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0
DMA33	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1
DMA34	UART1_TXDMA	UART1 Transmit DMA
DMA35	UART1_RXDMA	UART1 Receive DMA
DMA36	LP1_DMA	LP1 DMA Channel
DMA37	UART2_TXDMA	UART2 Transmit DMA
DMA38	UART2_RXDMA	UART2 Receive DMA
DMA39	MDMA2_SRC	Memory DMA Stream n Source Channel
DMA40	MDMA2_DST	Memory DMA Stream n Destination Channel
DMA41	FFTA0_TXDMA	FFTA0 Transmit DMA
DMA42	FFTA0_RXDMA	FFTA0 Receive DMA
DMA43	MDMA3_SRC	Memory DMA Stream n Source Channel
DMA44	MDMA3_DST	Memory DMA Stream n Destination Channel

DMA Definitions

To make the best use of the DMA controller, it is useful to understand the following terms.

Descriptor

An individual configuration fetched from memory that maps to a single register within a DMA channel.

Descriptor Fetch

The action of retrieving descriptors from memory through memory read operations and loading then into the DMA channel registers upon their read return.

Descriptor Set

A group of descriptors associated with a single work unit.

Disabled State

The channel is disabled because the enable bit = 0 or as a result of an error.

DMAC

An acronym used for a DMA cluster.

DMA Channel

A single DMA engine that has all the capabilities and registers as defined for a given processor. A DMA channel or engine is connected to a single peripheral.

DMA Cluster

A grouping of multiple DMA channels with a shared SCB crossbar interface, controller, and arbiter. Also known as a DMAC.

Initial Descriptor

The first descriptor in the descriptor set.

MDMA

Memory-to-Memory DMA data transfer. Two DMA channels are paired to perform a memory read from one address location and a memory write of that data to another address location.

Stop State

A time where the channel is enabled but not currently programmed to perform a data transfer. Programming the flow to STOP causes the channel to enter the stop state at the end of the work unit.

User

Any person, debug, emulator, software routine, or action taken by the core that accesses the MMR registers of the DMA channel or peripherals, or sets up data and descriptors in memory.

Wait State

If instructed to wait for a trigger, the channel enters this state once it has completed a work unit. The channel remains in this state until a trigger occurs. If a trigger came in before reaching the wait state, the channel skips over the wait state upon completion of the work unit.

Work Unit

A single data transaction or series of data transactions performed based on the configuration of the DMA channel. For autobuffer mode, a new work unit is defined at the time all current count registers are initialized to start values. Once all the current count registers count down to zero, the work unit has completed.

Work Unit Chain

A single work unit or a series or work units separated by a `STOP` or disabled state. The work units in the chain are programmed to another descriptor flow. The last work unit in the chain is programmed to a flow of `STOP` or `AUTO`. `STOP` terminates the state at the end of that work unit. `AUTO` must be be terminated by disabling the DMA channel. A work unit chain is also known as a descriptor chain.

Block Diagram

The ***DMA Channel Block Diagram*** shows the functional blocks within the DMA interface.

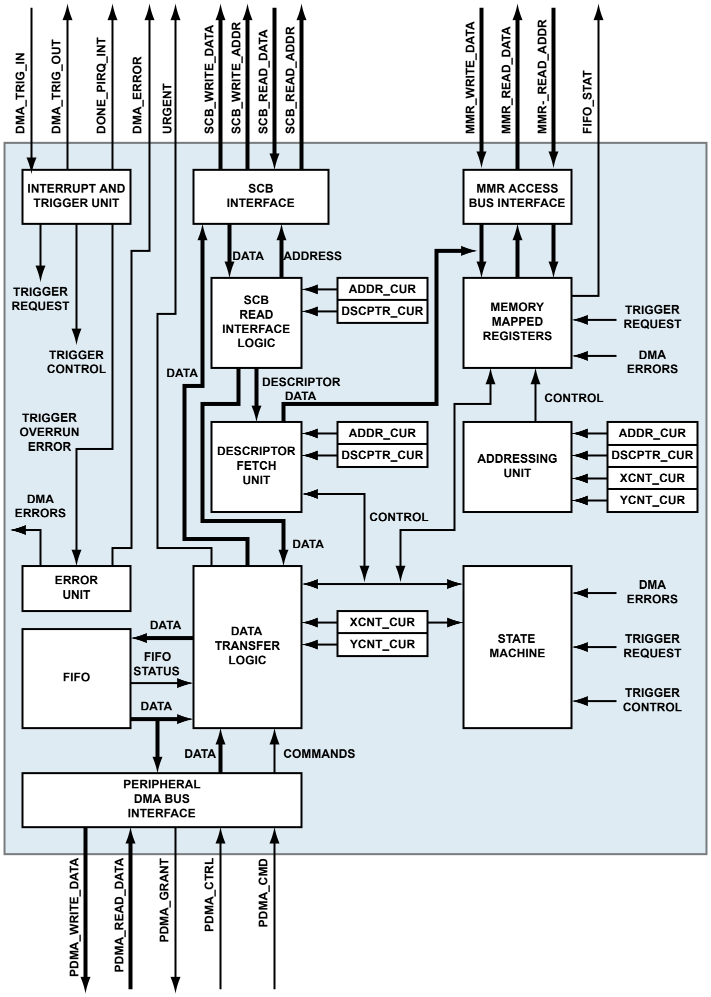

DMA_TRIG_IN
DMA_TRIG_OUT
DONE_PIRQ_INT
DMA_ERROR
URGENT
SCB_WRITE_DATA
SCB_WRITE_ADDR
SCB_READ_DATA
SCB_READ_ADDR
MMR_WRITE_DATA
MMR_READ_DATA
MMR_READ_ADDR
FIFO_STAT
INTERRUPT AND TRIGGER UNIT
SCB INTERFACE
MMR ACCESS BUS INTERFACE
DATA
ADDRESS
TRIGGER REQUEST
SCB READ INTERFACE LOGIC
ADDR_CUR
DSCPTR_CUR
MEMORY MAPPED REGISTERS
TRIGGER REQUEST
DMA ERRORS
TRIGGER CONTROL
DATA
DESCRIPTOR DATA
CONTROL
TRIGGER OVERRUN ERROR
DESCRIPTOR FETCH UNIT
ADDR_CUR
DSCPTR_CUR
ADDRESSING UNIT
ADDR_CUR
DSCPTR_CUR
XCNT_CUR
YCNT_CUR
DMA ERRORS
CONTROL
DATA
ERROR UNIT
DMA ERRORS
DATA
DATA TRANSFER LOGIC
XCNT_CUR
YCNT_CUR
STATE MACHINE
TRIGGER REQUEST
TRIGGER CONTROL
FIFO
FIFO STATUS
DATA
DATA
COMMANDS
PERIPHERAL DMA BUS INTERFACE
PDMA_WRITE_DATA
PDMA_READ_DATA
PDMA_GRANT
PDMA_CTRL
PDMA_CMD

Figure 38-1: DMA Channel Block Diagram

For more information on the interfaces shown in the block diagram, see:

- DMA Channel Peripheral DMA Bus
- Medium Band Width DMA Channel MMR Access Bus
- DMA Channel Event Control
- DMA Channel SCB Interface

Architectural Concepts

The DMA channel provides a method to transfer data between memory spaces or between memory and a peripheral using a number of system interfaces. The DMA channel provides an efficient method of distributing data throughout the system, freeing up the processor core for other operations. Each peripheral that supports DMA transfers has its own dedicated DMA channel or channels with its own register set. The register set configures and controls the operating modes of the DMA transfers.

DMA Channel SCB Interface

The SCB interface connects the DMA channel to the SCB crossbar allowing for transfers to and from the processors internal memory and other suitable system resources.

The DMA channel connects to the system interconnect through the SCB interface. This connection lets the DMA channel perform work-unit data transfers with memories such as L1, L2 (internal), and L3 (external). In addition to work unit data transfers, the SCB interface also is used for fetching descriptor sets for all the descriptor-based transfer modes.

The DMA channel can support data bus widths of 16, 32, 64, or 128 bits. The data bus widths for a given DMA channel on a specific processor can vary and are not configurable. Read the `DMA_STAT.MBWID` field to determine the assigned bus widths.

SCB Interface Signals

The DMA channel operates at one of the *SCLKn* frequencies, as does the SCB interface. SCLK0 clocks all but four DMA channels which are clocked by SCLK1. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance as shown in the ***SCB Interface Signals*** table.

Table 38-3: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16, 32, 64, or 128	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.MBWID`.
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction.
SCB_READ_DATA	16, 32, 64, or 128	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.MBWID`.
SCB_READ_ADDRESS	32	Read address bus. Provides the address of the first transfer in a burst transaction.

SCB Burst Transfers

The SCB interface supports burst transfers for memory read and write operations. The burst length is a function of the configurable memory size of the DMA channel for the work unit and the fixed bus width of the SCB data bus of the DMA channel.

- If the DMA channel configuration selects a memory transfer size less than or equal to the DMA channels bus width, the burst length is always 1.
- If the configured memory size is greater than the SCB interface bus width, the burst length is sufficient to transfer a transaction as specified by the configured memory size.

Table 38-4: DMA Channel SCB Burst Lengths

Configured Memory Size	Burst Length			
	16-bit Bus	*32-bit Bus*	*64-Bit Bus*	*128-bit Bus*

Table 38-4: DMA Channel SCB Burst Lengths (Continued)

Configured Memory Size	Burst Length			
1 Byte	1	1	1	1
2 Bytes	1	1	1	1
4 Bytes	2	1	1	1
8 Bytes	4	2	1	1
16 Bytes	8	4	2	1
32 Bytes	16	8	4	2

Data Address Alignment

To prevent addressing errors and to maximize bandwidth of the SCB interface to the DMA channel, data addresses align with a multiple of the programmable memory size of the DMA channels configuration. These configuration options appear in the Descriptor Set Address Alignment table.

There are situations in which entire work units may not transfer at the maximum configurable memory size. In this case, the entire work unit can transfer by reducing the configured memory size at the expense of bus bandwidth using descriptor sets as follows:

- The first descriptor set can be configured to transfer data until the larger memory size alignments are met.
- A second descriptor set with a larger memory size configuration then can be used to transfer the bulk of the data in the work unit.
- Finally, a third descriptor set can be used with a smaller memory size to complete any final data transfers that cannot meet the alignment requirements of the previous descriptor set configuration.

Table 38-5: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction
2 Bytes	ADDR[0] == 0
4 Bytes	ADDR[1:0] == 0
8 Bytes	ADDR[2:0] == 0
16 Bytes	ADDR[3:0] == 0
32 Bytes	ADDR[4:0] == 0

Descriptor Set Address Alignment

All descriptor set addresses and descriptors within a descriptor set must align to a 32-bit address. For descriptor set fetches, the DMA engine ignores the memory-size configuration of the DMA channel. This feature avoids the need to align descriptor sets based on the memory width configuration of the previous descriptor set.

For descriptor sets containing only a single descriptor, the transfer takes place as a single 32-bit transfer. For descriptor sets containing multiple descriptors, the DMA engine fetches each 32-bit descriptor individually and treats it as multiple 32-bit transfers.

DMA Channel Peripheral DMA Bus

The DMA channel connects to peripherals or other DMA channels through the peripheral DMA bus. This bus is a dedicated point-to-point interface supporting data bus widths of 8, 16, 32, or 64 bits. The data bus widths for a given DMA channel on a particular processor can vary and are not configurable. Reading the `DMA_STAT.PBWID` field permits determining the assigned bus width.

The DMA channel operates at one of the *SCLKn* frequencies, as does the peripheral DMA bus. The ***Peripheral DMA Bus Signals*** table provides descriptions of the peripheral DMA bus signals.

Table 38-6: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8, 16, 32, or 64	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.PBWID`.
PDMA_READ_DATA	8, 16, 32, or 64	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.PBWID`.
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit). These signals indicate that the DMA channel is ready to receive data for write operations (peripheral receive).
PDMA_CMD	3	The peripheral uses the signal for issuing DMA channel control commands.
PDMA_CTRL		The peripheral uses the control signals to send various commands to the DMA channel and control the direction of flow.

Peripheral Control Commands

The peripheral DMA bus of the DMA channel provides a means for peripherals on the processor to issue commands to the DMA channel. These commands provide greater control over the DMA channel operation. This control improves real-time performance and relieves control and interrupt demands on the core. Peripherals can send commands to the DMA controller over the 3-bit PERI_CMD bus. The DMA control commands extend the set of operations available to the peripheral beyond the simple “request data” command used by peripherals in general. Refer to the appropriate peripheral chapter for a description on how that peripheral uses DMA control commands.

These DMA control commands (see the ***PDMA_CMD Peripheral DMA Control Commands*** table) are not visible to or controlled by the program. But, their use by a peripheral has implications for the structure of the DMA transfers that the peripheral can support. It is important to write application software such that it complies with certain restrictions, regarding work units and descriptor chains. Complying with this guideline makes the peripheral operate properly whenever it issues DMA control commands.

The ***PDMA_CMD Peripheral DMA Control Commands*** table describes the commands the DMA controller issues. The following sections describe these commands in more detail.

Table 38-7: PDMA_CMD Peripheral DMA Control Commands

Command	Name	Description
b#000	NOP	No operation
b#001	Restart	Restarts the current work unit from the beginning
b#010	Finish	Finishes the current work unit and starts the next
b#011	Interrupt	Immediately sets the DMA completion interrupt in the DMA channel
b#100	Request Data	Typical DMA data request
b#101	Request Data Urgent	Urgent DMA data request
b#110	Reserved	Reserved
b#111	Reserved	Reserved

Idle Command

The DMA channel drives this command when the enabled peripheral has no data requests required.

Restart Command

This command causes the current work unit to interrupt processing and start again, using the addresses and count values from the `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_YCNT` registers. The DMA controller does not signal an interrupt request when the work unit terminates.

If a channel programmed to transmit (memory read) receives a restart command, the channel momentarily pauses, permitting any pending memory reads initiated before the restart command to complete. During this period, the channel does not grant DMA requests. After all pending reads flush from the pipelines of the channel, the channel resets its counters and FIFO, and then starts pre-fetch reads from memory. The DMA controller grants data requests from the peripheral as soon as new prefetched data is available in the DMA FIFO. In this case, the peripheral can use the restart command to reattempt a failed transmission of a work unit.

If a channel programmed to receive (memory write) receives a restart command, the channel stops writing to memory, discards any data held in its DMA FIFO, and resets its counters and FIFO. As soon as this initialization is complete, the channel again grants DMA write requests from the peripheral. In this case, the peripheral can use the restart command to abort the transfer of received data into a work unit, and reuse the memory buffer for a later data transfer.

The request from the restart control command is not granted or acknowledged. The DMA controller always accepts the request.

Finish Command

The finish command causes the current work unit to terminate processing and move on to the next work unit. If enabled within the `DMA_CFG` register, the DMA channel signals an interrupt or a trigger event. The peripheral can then use the finish command to partition the DMA stream into work units on its own. This partitioning occurs---perhaps as a result of parsing the data currently passing though its supported communication channel---without direct real-time control by the processor.

When a DMA channel programmed to transmit (memory read) then receives a finish command, the channel momentarily pauses for the completion of any pending memory reads, which were initiated prior to the finish command. During this time, the channel does not grant DMA requests. After the flush of all pending reads from the pipelines of the channel, the channel signals an interrupt request or a trigger (if enabled) and begins fetching the next descriptor (if any). DMA data requests from the peripheral are granted as soon as new prefetched data is available in the DMA FIFO.

If a channel programmed to receive (memory write) then receives a finish command, the channel stops granting new DMA requests while it drains its FIFO. The channel writes to memory any DMA data received by the DMA channel prior to the finish command. When the FIFO reaches an empty state, the channel signals an interrupt or a trigger (if enabled) and begins fetching the next descriptor (if any). After fetching the next descriptor, the channel initializes its FIFO, then resumes granting DMA requests from the peripheral.

The finish command request is not granted or acknowledged. The request is always accepted by the DMA channel.

Interrupt Command

The interrupt command causes the DMA channel to generate an interrupt request. When programming the channel to support this command, configure the `DMA_CFG.INT` bit field to PIRQ mode. This configuration directs the channel not to generate interrupt requests based on the work unit state. Instead, the channel generates interrupts only when it receives the interrupt command from the peripheral. When the channel receives an interrupt request command, the `DMA_STAT.PIRQ` bit indicates the event under the following conditions:

- The `DMA_CFG.EN` bit enables the DMA channel.
- The DMA channel is in the stop state.
- The interrupt in `DMA_CFG.INT` is configured for PIRQ mode.

The peripheral only issues the interrupt command in response to receiving the last grant command from the DMA channel, indicating that the transfer is the last transfer in the work unit.

Request-Data Command

The request data command is a request for data transfers between the DMA channel and the peripheral. The request is held by the peripheral until granted or acknowledged by the DMA channel.

Request-Data Urgent Command

The request-data urgent command behaves identically to the request data command, except that---during the commands assertion---the DMA channel performs its memory accesses with urgent priority. This priority includes both data and descriptor fetch memory accesses. For example, a DMA management capable peripheral can use this control command if an internal FIFO approaches a critical condition.

The request is held by the peripheral until granted or acknowledged by the DMA channel.

Peripheral-Control Command Restrictions

The proper operation of the DMA channel FIFO leads to certain restrictions in the sequence of DMA peripheral control commands issued by a peripheral. The following sections describe these restrictions.

Transmit-Restart or Transmit-Finish Command

A peripheral only can issue a restart or finish control command to a channel configured for memory read under the following conditions:

- The peripheral has already performed at least one DMA transfer in the current work unit.
- The current work unit has (`FIFO_SIZE/DMA_CFG.MSIZE`) + 1 memory transfers remaining.

The first item ensures that the work unit has started. The second item ensures that the work unit has not completed. The second item is sufficiently large that it is always at least five more than the maximum data count before any restart or finish command. If using restart or finish commands to manage a work unit, this requirement implies that the work unit must have `DMA_XCNT_CUR` and `DMA_YCNT_CUR` register values representing at least five data items.

To satisfy the second item, ensure that the number of memory transfers described by the descriptor is (`FIFO_SIZE/DMA_CFG.MSIZE`) + 1 larger than the maximum number of memory transfers expected.

Receive-Restart or Receive-Finish Commands

A peripheral only can issue a restart or finish control command to a channel configured for memory write under the following conditions:

- The number of peripheral transfers completed is less than (`DMA_CFG.MSIZE/DMA_CFG.PSIZE`) × (transfers described by descriptor).
- In addition to the previous condition, one of the following conditions also must apply:
  - A finish command terminated the previous work unit, *and* the peripheral has done at least one transfer in the current work unit.
  - The peripheral has done (`FIFO_SIZE/DMA_CFG.PSIZE`) + 1 transfers in the current work unit.

The first condition ensures that the descriptor is still active. The second set of conditions ensures that data from the previous descriptor has left the FIFO and that the current descriptor has started.

Memory DMA and Triggering

A memory DMA (MDMA) channel provides a means of doing memory-to-memory DMA transfers among the various memory spaces that have DMA support.

The DMA controller implements memory DMA (MDMA) channels by interfacing two DMA channels through the peripheral DMA bus interface. One DMA channel serves for memory read operations, and the second channel servers for memory writes. Depending on the processor, a memory DMA channel can have an additional peripheral, such as a CRC peripheral. The additional peripheral is inserted into the peripheral DMA bus that optionally can be enabled.

MDMA channel configurations that do not involve an additional peripheral impose no restrictions on which of the DMA channels is used for the read operation or the write operation. But, the configuration of both channels cannot have the same transfer direction. For MDMA channel configurations that enable a peripheral between the read and write channels, be aware of possible restrictions imposed on which channel can be used for a given transfer direction.

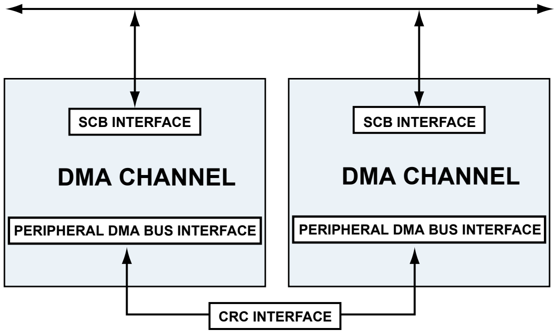

SCB INTERFACE
DMA CHANNEL
PERIPHERAL DMA BUS INTERFACE
SCB INTERFACE
DMA CHANNEL
PERIPHERAL DMA BUS INTERFACE
CRC INTERFACE

Figure 38-2: MDMA Channel Dedicated Pair

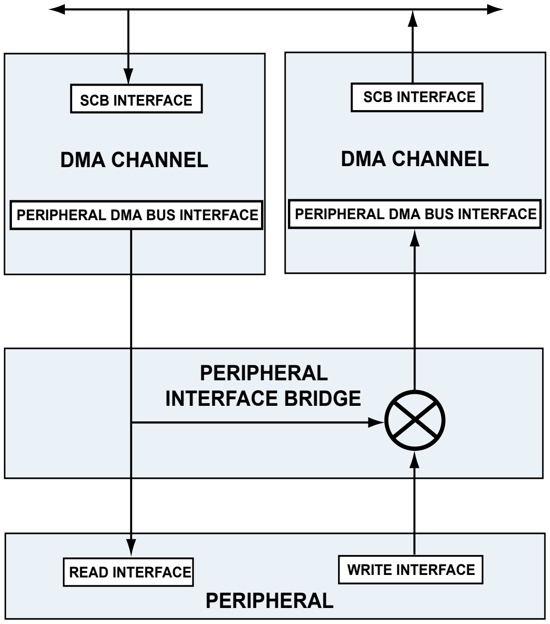

SCB INTERFACE
DMA CHANNEL
PERIPHERAL DMA BUS INTERFACE
SCB INTERFACE
DMA CHANNEL
PERIPHERAL DMA BUS INTERFACE
PERIPHERAL
INTERFACE BRIDGE
READ INTERFACE
WRITE INTERFACE
PERIPHERAL

Figure 38-3: MDMA Channel Pair with Peripheral

A memory-to-memory transfer always requires enabled source and destination channels. Because the channels interface through the peripheral DMA bus and can have an additional peripheral inserted into the peripheral DMA bus, programs must make sure to set the same values in the `DMA_CFG.PSIZE` of both the source and destination channels.

The memory DMA channels support the full range of the `DMA_CFG.MSIZE` options for the DMA transfers to and from the memories.

Because the MDMA channel consists of two DMA channels, the entire MDMA channel has two sets of FIFOs, one in the read channel and one in the write channel. This FIFO usage allows for more efficient bursting of both read and write transactions using the available bandwidth. While the `DMA_CFG.PSIZE` configuration must be identical for both source and destination DMA channels, this restriction does not apply for the `DMA_CFG.MSIZE` configuration.

Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus.

NOTE: This is applicable for all of the DMA channels, except the enhanced and high-speed MDMA channels. For the enhanced and high-speed MDMAs, the minimum *PSIZE* and *MSIZE* is 4 bytes (32 bits).

The independent source and destination DMA channels also have their own dedicated interrupt and trigger events. While it is normal practice to have only event generation performed at destination DMA completion, programs also can use other means of interrupt generation.

Configuration of an MDMA transfer is done in a similar manner to peripheral DMA transfers, except for writing two DMA channel registers instead of one.

To control the pace of data transfers, use triggers on either the memory read or the memory write channel pair used in an MDMA operation. Setting the `DMA_CFG.TWAIT` bit in the memory read channel prevents both channels from transferring data before the system is ready. However, only configuring the memory write channel to wait for a trigger allows for data fetch from the memory in anticipation of the memory write operation.

The processor supports these categories of MDMA channels:

Standard Bandwidth MDMA. The channels run in the SCLK0 domain and support 32-bit memory and peripheral bus width. The maximum theoretical bandwidth of these MDMA streams is up to 4 × 112.5 = 450 MB/s. This MDMA optionally supports CRC.

Enhanced Bandwidth MDMA. The channels run in SYSCLK domain and support 32-bit memory and peripheral bus width. The maximum theoretical bandwidth of this MDMA stream is up to 4 × 225 = 900 MB/s.

Maximum Bandwidth MDMA. The channels run in SYSCLK domain and support 64-bit memory and peripheral bus width. The maximum theoretical bandwidth of these MDMA streams is up to 8 × 125 = 1800 MB/s. However, the practical maximum throughput for these MDMA streams is only up to 1500 MB/s.

NOTE: One MDMA channel is shared with the FFTA.

Medium Band Width DMA Channel MMR Access Bus

The MMR access bus provides access to all the DMA channels memory-mapped registers for DMA channel configuration, monitoring, and debug. The interface has a fixed 32-bit data bus for read and write accesses.

The ***MMR Access Bus Signals*** table provides descriptions of the MMR access bus signals.

Table 38-8: MMR Access Bus Signals

Signal	Width (bits)	Description
MMR_WRITE_DATA	32	Data bus used for write operations to the MMRs from the core
MMR_READ_DATA	32	Data bus used to return read data from the MMRs
MMR_READ_ADDR	7	Address used to select the MMR to access

DMA Channel Operation Flow

A detailed description of the flow of operation of the DMA channel appears in the following topics:

- Startup Flow
- Refresh Flow
- DMA Operating Modes
- Stop Mode
- DMA Channel Errors

Startup Flow

Enabling a DMA operation on a given channel first requires directly writing some or all of the DMA parameter registers. The minimum set of register required to be initialized depends on the desired mode of operation as described in the following sections.

Startup Minimum-Enable Requirements

To start a DMA operation on a given channel, some or all of the DMA parameter registers must first be initialized and configured to the desired DMA channels operating mode.

- For descriptor-array-based flow modes, at minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For descriptor-list-based flow modes, at minimum, write the `DMA_DSCPTR_NXT` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.
- For non-descriptor-based flow modes, write the `DMA_ADDRSTART`, `DMA_XCNT`, and `DMA_XMOD` registers prior to writing the `DMA_CFG` register.

Programs can write other registers that can remain static throughout the course of the DMA activity. The write to the `DMA_CFG` register begins the DMA operation.

ATTENTION: When software directly writes the `DMA_CFG` register, the DMA controller recognizes this action as the special startup condition. This condition occurs when starting the DMA controller for the first time on this channel or occurs after the DMA channel stops. It is possible for the channel to flag a DMA error condition regardless of the `DMA_CFG.EN` bit setting.

Startup Operation

The startup operation is initiated by software directly writing the `DMA_CFG` register when starting DMA for the first time on a channel or after the channel has entered to the stop state.

When the descriptor fetch is complete and the DMA channel is enabled, the `DMA_CFG` descriptor element in the `DMA_CFG` register assumes control. Before this point, the direct write to the `DMA_CFG` register had control.

At startup, the selected flow mode and the descriptor size determine the course of the DMA initialization process. The `DMA_CFG.FLOW` field determines whether to load more current registers from descriptor sets in memory. The

DMA_CFG.NDSIZE field details how many descriptor elements to fetch before starting the DMA operation. This process does not affect DMA registers that are not in the descriptor; no modifications are made to their prior values.

For descriptor-list flow modes, the channel copies the DMA_DSCPTR_NXT register value into the DMA_DSCPTR_CUR register. Then, the channel fetches new descriptor elements from memory. The DMA_DSCPTR_CUR register indexes each fetch, and the channel increments the index after each fetch. After completion of the descriptor fetch, the DMA_DSCPTR_CUR register points to the next 32-bit word in memory past the end of the descriptor.

If the descriptor fetch is for a descriptor-array mode transfer, the channel does *not* copy the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register. *Instead*, the descriptor fetch indexing begins with the value in the DMA_DSCPTR_CUR register.

If DMA_CFG is not part of the fetched descriptor set, the previous value (originally as written on startup) controls the work unit operation. If the DMA_CFG register is part of the fetched descriptor set, the value programmed by the MMR access controls only the loading of the first descriptor fetched from memory. The configuration of the DMA_CFG register controls the subsequent DMA work units of the fetched descriptor set.

After the descriptor fetch is complete or if the flow configuration was originally for one of the register-based flow modes, the DMA operation begins. The DMA channel immediately fills its FIFO. For a memory-write operation, the DMA channel begins accepting data from the peripheral. For a memory-read operation, the DMA channel begins memory reads when the SCB bus grants access to the DMA channel.

When the DMA channel performs its first data-memory access, its address and count computations take their input operands from the start registers. These registers can include DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT, if necessary. The channel writes results back to the current registers. These registers include DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR. Note that the current registers are not valid until the channel performs the first memory access, which can be some time after the write to the DMA_CFG register starts the channel. Once started, the channel automatically loads the current registers from the appropriate descriptor elements, overwriting their previous contents. These automatic-load operations include:

- The channel copies the DMA_ADDRSTART value to DMA_ADDR_CUR.
- The channel copies the DMA_XCNT value to DMA_XCNT_CUR.
- The channel copies the DMA_YCNT to DMA_YCNT_CUR.

Refresh Flow

When the channel completes processing of a work unit, the DMA channel performs the following operations:

- Completes the transfer of all data between memory and the DMA channel.
- Performs a synchronized transition (if the DMA channel configuration is a memory read operation with the DMA_CFG.SYNC bit enabled) *and* transfers all data to the peripheral before continuing.
- Forwards the signals from the DMA channel (if interrupts or triggers are enabled) *and* updates the DMA_STAT register to indicate the interrupt request or trigger events.

- Clears the DMA_STAT.RUN bit field to stop DMA operation (if the flow was set to stop mode) *and* transfers any remaining data in the FIFO of the DMA channel to the peripheral.
- Loads a new descriptor from memory into the DMA registers by way of the contents of the DMA_DSCPTR_CUR register (for descriptor-array mode) *and* increments the DMA_DSCPTR_CUR register

  The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.
- Copies the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register (for descriptor-list mode), fetches the descriptor from the new contents of the DMA_DSCPTR_CUR register, *and* places these contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.
- Checks for detection of an incoming trigger event (for descriptor-on-demand array mode):
  - If the channel detects a trigger event, the DMA channel loads a new descriptor from memory into the DMA registers from the contents of the DMA_DSCPTR_CUR register, while incrementing the DMA_DSCPTR_CUR register. The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.
  - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers.
- Checks for detection of an incoming trigger event (for descriptor-on-demand list mode):
  - If the channel detects a trigger event, the DMA channel copies the DMA_DSCPTR_NXT register value to the DMA_DSCPTR_CUR register, fetches the descriptor memory from the DMA_DSCPTR_CUR register, *and* places the contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.
  - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers as described in the next step.
- Begins the next work unit (if flow configuration is anything other than stop mode) by reloading the current registers (DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR) from their descriptor registers (DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT)

Work Unit Transition Flow

The DMA_CFG.SYNC bit controls transitions from one work unit to the next work unit. In general, continuous transitions have lower latency at the cost of restrictions on changes of data format or addressed memory space in the two work units. These latency gains and data restrictions arise from the way the channel handles the DMA FIFO while fetching the next descriptor.

In continuous transitions, with disabled synchronization, the DMA FIFO pipeline continues to transfer data to and from the peripheral or destination memory. These transfers continue during the descriptor fetch and during the DMA channel pause between descriptor chains. By comparison, synchronized transitions provide better real-time synchronization of interrupts and triggers with a given peripheral state. Synchronized transitions also provide greater flexibility in the data formats and memory spaces of the two work units. This flexibility comes at the cost of higher latency in the transition. In synchronized transitions, the DMA FIFO pipeline drains to the destination or flushes (received data discarded) between work units.

NOTE: The `DMA_CFG.SYNC` bit of the MDMA source channel controls work unit transitions for MDMA streams. Clear this reserved bit of the MDMA destination channel, placing it in the disabled state. In transmit (memory read) channels, the `DMA_CFG.SYNC` bit of the last descriptor before the transition controls the transition behavior. In contrast, in receive channels, the `DMA_CFG.SYNC` bit of the first descriptor of the next descriptor chain controls the transition.

Work Unit Transmit and MDMA Source Transitions

In DMA transmit (memory read) and MDMA source channels, the `DMA_CFG.SYNC` bit controls the interrupt timing at the end of the work unit. This bit also controls the handling of the DMA FIFO between the current and the next work unit.

If the `DMA_CFG.SYNC` bit configuration disables synchronization, the DMA channel operates in continuous transition. In a continuous transition, just after reading the last data item from memory, the DMA channel starts all of the following operations parallel:

- Signals the interrupt request or trigger
- Updates the `DMA_STAT` register to indicate DMA completion status
- Begins fetching the next descriptor
- Delivers the final data items from the DMA FIFO to the destination memory or peripheral

This process lets the DMA channel provide data from the FIFO to the peripheral continuously during the descriptor fetch latency period.

If the configuration disables synchronization, the final interrupt request or trigger (if enabled) occurs when the channel reads the last data from memory. This event occurs at the earliest time that the channel safely can modify the output memory buffer without affecting the previous data transmission. There can be a number of data items remaining in the FIFO and not yet at the peripheral. This number depends on the FIFO depth of the DMA channel. In this configuration, do not use the DMA interrupt request as the sole means of synchronizing the shutdown or reconfiguration of the peripheral following a transmission.

NOTE: If the configuration selects continuous transition on a transmit (memory read) descriptor, the next descriptor must have the same:

- Peripheral transfer size (`DMA_CFG.PSIZE`)
- Read or write direction
- Source memory (internal versus external) as the current descriptor

It is possible to disable synchronization by selecting continuous transition on a work unit with configuration for stop-flow mode and with enabled interrupts or triggers. This approach can result in the execution of the event service routine while draining of the final data is ongoing from the FIFO to the peripheral. If data transfers are in-progress, the FIFO is not yet empty. The `DMA_STAT.RUN` bits of the DMA channels indicate this status. Do not start a new work unit with a different peripheral transfer size or direction while data transfers are in-progress.

CAUTION: Disabling the channel with the `DMA_CFG.EN` bit while data transfers are in-progress causes the loss of the data in the FIFO.

A synchronized transition configuration directs the channel to drain the DMA FIFO to the destination memory or peripheral. This FIFO operation occurs before the channel signals any interrupt and before the channel fetches any subsequent descriptor or data. This operation incurs greater latency, but provides direct synchronization between the DMA interrupt and the state of the data at the peripheral.

If the configuration enables synchronization and enables interrupts, on the last descriptor in a work unit, the interrupt occurs when the channel transfers the final data to the peripheral. This event allows the service routine to switch properly to non-DMA transmit operation. When the event vectors to the interrupt service routine, the DMA channel FIFO is empty, and the DMA channel is no longer running (indicated by the `DMA_STAT.RUN` bits).

A synchronized transition also allows greater flexibility in the format of the DMA descriptor chain. When enabled, the next descriptor can have any `DMA_CFG.PSIZE` configuration or read/write direction supported by the peripheral and can come from either memory space (internal or external). This feature can be useful in managing MDMA work unit queues, since it is no longer necessary to interrupt the queue between dissimilar work units.

Work Unit Receive and MDMA Destination Transitions

In DMA receive channels (memory write operations), the `DMA_CFG.SYNC` bit controls the handling of the DMA FIFO between descriptor chains (not individual descriptor sets), during the DMA channel pause. The DMA channel pauses after the descriptor sets configured with stop flow mode are complete. Restart the channel (for example, after an interrupt) by writing the `DMA_CFG` register of the channel with a value that enables the DMA channel. If the configuration disables synchronization in the `DMA_CFG` value of the new work unit, the configuration selects a continuous transition. In this mode, the DMA FIFO retains any data items received during the channel pause, and they are the first items written to memory in the new work unit. This mode of operation provides lower latency at work unit transitions and ensures no dropping of data items during a DMA pause. The channel provides this operation at the cost of certain restrictions on the DMA descriptors.

NOTE: If the `DMA_CFG.SYNC` bit disables synchronization on the first descriptor of a chain after a DMA pause, do not change the configuration of the `DMA_CFG.PSIZE` field of the new chain from the previous descriptor chain (active before the pause). This restriction applies unless the DMA channel is reset between chains by disabling and then re-enabling the DMA channel.

If the `DMA_CFG.SYNC` bit configuration enables synchronization, the channel uses a synchronized transition. In this mode, only the data that the DMA channel receives from the peripheral after the write to the `DMA_CFG` register gets to memory. The channel discards any prior data items transferred from the peripheral to the DMA FIFO before this register write occurs. This operation provides direct synchronization between the data stream received from the peripheral and the timing of the channel restart, which occurs on the write to the `DMA_CFG` register.

For receive DMA operations, the synchronization has no effect in transitions between work units in the same descriptor chain. When the flow mode of previous descriptor was not stopped, the DMA channel did not pause.

If a descriptor chain begins with synchronization enabled, there is no restriction on the `DMA_CFG.PSIZE` of the new chain in comparison with the previous chain.

NOTE: The peripheral transfer size (`DMA_CFG.PSIZE`) must not change between one descriptor and the next in any DMA receive (memory write) channel within a single descriptor chain, regardless of the `DMA_CFG.SYNC` bit setting. In other words, all memory write descriptor sets in a descriptor chain must have the same `DMA_CFG.PSIZE` value. For any DMA receive channel (memory write operation), there is no restriction on changes of peripheral transfer size (internal versus external) between descriptors or descriptor chains.

Transfer Termination and Shutdown Flow

This section describes channel transfer termination and shutdown in stop flow mode and in autobuffer flow mode.

Stop Flow Mode

In stop flow mode, the DMA channel stops automatically after the work unit is complete. If using a list or array of descriptors to control DMA transfers and if every descriptor contains a `DMA_CFG` descriptor element, configure the flow of the final `DMA_CFG` descriptor element to stop mode, stopping the channel gracefully. After completion, the DMA channel remains in the stop state. Do not confuse this state with the disabled state, which either occurs due to a DMA error or occurs through disabling the DMA channel by configuring the `DMA_CFG.EN` bit.

The intention of disabling the DMA channel through a write to the `DMA_CFG.EN` bit is to shut down the DMA channel and to enter the disabled state. All memory and peripheral data transfers cease, and only peripheral interrupts pass through the DMA channels interrupt signals. However, the DMA channel maintains the `DMA_STAT.RUN` bits. For a write to memory, the outstanding memory-transaction counter tracks returning memory write acknowledgments and updates as required.

For memory reads, the outstanding memory-transaction count also tracks returning memory reads. The channel does not write the memory reads into the FIFO. The channel updates the counter to reflect the completion of the transaction, but the channel ignores the data. The `DMA_STAT.RUN` bits remain in the *waiting for write ACK or FIFO drain to peripheral* state and do not change to *stop or idle state* until the return of all outstanding transactions.

When the `DMA_CFG.EN` bit again enables the DMA channel, the channel performs a full reset and clears all counters. If an outstanding memory transaction returns an acknowledgment or read data after this event, a memory transaction error occurred, which generates an error event. Programs must ensure that all outstanding memory transactions complete before reconfiguring the DMA channel. For example, programs can poll the `DMA_STAT.RUN` bits to return to the stop or idle state before proceeding.

Autobuffer Flow Mode

In this mode, the flow does not use any descriptors in stored memory. Instead, the channel performs DMA in a continuous circular buffer fashion, based on user-programmed DMA register settings. On completion of the work unit, the channel reloads the parameter registers into the current registers, and the DMA controller resumes immediately with zero overhead. Consider this mode as a succession of automatically restarted work units.

For autobuffer-flow modes, the only way to cease operations is to disable the DMA channel through the `DMA_CFG.EN` bit. One method of changing to a new work unit is:

- Disable the DMA channel
- Set up all the registers (and descriptors in memory, if used) except for `DMA_CFG`

- Poll `DMA_STAT.RUN` to wait for the status to reflect stop or idle state, and
- Write `DMA_CFG` to the new configuration to begin the next work unit

In autobuffer-flow mode or for a list or array of descriptor sets without `DMA_CFG` descriptors, use an MMR write to the `DMA_CFG` register to terminate the DMA transfer process. Configure the value of the `DMA_CFG.EN` bit in this register to disable the DMA channel.

CAUTION: When the configuration disables a DMA channel, the DMA controller disables interrupt logic that is based on work unit transitions. Be aware of the system environment and current actions, so that additional interrupts are not required from the DMA channel.

CAUTION: If disabled through `DMA_CFG.EN` in the middle of a transaction, the DMA channel completes any transactions that have begun and avoids generating bus errors. However, the channel considers the action of re-enabling the DMA as a hard reset for all internal DMA channel components. Therefore, pay attention to that particular action to avoid unexpected results.

DMA Channel Errors

When an error occurs, the DMA channel maintains all the state and register values that allow programs to diagnose error causes more thoroughly. The greatest benefit to the programmer is to know exactly what operational state the DMA channel was in at the exact moment the error occurred.

Take care to address the root cause of the error, whether or not the problem originated in the DMA channel. If not properly resolved, the error can result in an additional error shortly after operations resume. The problem can cause other errors elsewhere in the DMA channel or associated modules and circuitry. So, take care also to address those potential problems. Ensure that all outstanding memory reads and writes are complete or cleared before resuming DMA channel operation.

After addressing all issues and neutralizing all side effects of any errors, clear the `DMA_STAT.ERRC` status field and restart the DMA channel by disabling then re-enabling the DMA channel through the `DMA_CFG.EN` bit.

The following sections describe the error types.

Status and Debug Errors

DMA channel error conditions can cause the DMA process to end abnormally. The DMA channel provides error detection as a tool for system development and debug, helping to identify DMA-related programming errors. When the DMA channel detects an error, the channel immediately stops and discards any returned memory-read transactions. The `DMA_STAT.RUN` field of the DMA channel indicates the idle state after acknowledging all outstanding memory transactions. In addition, the channel asserts an error interrupt request and updates the `DMA_STAT.IRQERR` field. Also, the channel updates the `DMA_STAT.ERRC` field, indicating the error cause of the first detected error. Unless the error occurs at the exact moment that modification of register values occurs, the registers contain the error values.

All the DMA error interrupt requests are combined into a single shared interrupt request output. Combined error signals require reading the `DMA_STAT` register of each DMA channel associated with a combined error interrupt request to determine the DMA channel responsible for the generation of the interrupt.

The DMA channel error interrupt handler performs the following actions:

- Read the `DMA_STAT` register of each DMA channel, seeking a channel with the `DMA_STAT.IRQERR` set to indicate an error.
- Read the `DMA_STAT.ERRC` field of each DMA channel, determining the cause of the error.
- Clear the problem with the DMA channel. For example, fix the register values.
- Clear the error in the DMA channel through a write-1-to-clear operation to the `DMA_STAT.IRQERR` bit.

If the channel flags any uncleared error other than a bandwidth monitor error, the channel reports no other error. If the channel reports an uncleared bandwidth monitor error, the channel reports any newly detected error through updating the `DMA_STAT.ERRC` field.

DMA Configuration Register Errors

The channel only flags these configuration errors when the `DMA_CFG.EN` bit enables the DMA channel. Error flagging occurs when the configuration:

- Uses a reserved setting
- Enables `DMA_CFG.TWAIT` in descriptor on-demand flow mode
- Uses an illegal `DMA_CFG.NDSIZE`
- Uses an illegal `DMA_CFG.MSIZE`
- Configures `DMA_XCNT` = 0 or, when `DMA_YCNT` = 0 in 2D DMA mode
- Uses non-zero value in `DMA_CFG.NDSIZE` when DMA is configured in stop mode or auto mode
- Enables interrupt or outgoing triggers on `DMA_YCNT` when DMA is configured in 1D mode
- Use a `DMA_CFG.MSIZE` that exceeds the FIFO size of the DMA channel
- Uses an illegal `DMA_CFG.PSIZE`
- Uses a `DMA_CFG.PSIZE` that exceeds the FIFO size
- Uses a `DMA_CFG.PSIZE` that exceeds the bus width
- Attempts to change from a transmit operation (memory read) to a receive operation without properly synching in the previous work unit or when it is the first work unit in a new chain
- Attempts to change `DMA_CFG.PSIZE` of a transmit operation (memory read) without properly synching in previous work unit or when it is the first work unit in a new chain
- Attempts to change from receive operation (memory write) to a transmit operation during a descriptor chain.

  The channel only can change from receive to transmit if the new transmit is synchronized and is the first work unit.

- Attempts to change `DMA_CFG.PSIZE` of a receive operation (memory write) when the operation was not the first work unit (with `DMA_CFG.SYNC` enabled)

Illegal Register Write During Run

The channel generates an error when a write occurs to writable registers of an enabled, running DMA channel. The channel blocks the write. The `DMA_STAT`, `DMA_BWLCNT`, and `DMA_BWMCNT` registers are exempt from this behavior. The `DMA_STAT` register is exempt from this behavior.

Address Alignment Error

The channel generates an address alignment error when any of the following apply:

- Alignment of a descriptor address is not on a 32-bit boundary.
- The current `DMA_CFG.MSIZE` configuration contains an unaligned transfer address. The `DMA_ADDRSTART` register is not aligned according to the `DMA_CFG.MSIZE` field.

Memory Access Error

The channel generates a memory access error when the DMA process:

- attempts to access an unpopulated address,
- attempts to access an address defined as cache, or
- attempts to access a location that provokes a security violation

The error returned from the memory triggers the memory access error.

Trigger Overrun Error

A trigger overrun error is generated when a new trigger input occurred while an outstanding trigger is waiting. This error is only generated if `DMA_CFG.TOVEN` is enabled.

Bandwidth-Monitor Error

The channel generates this error when the bandwidth-monitor count expires. This error is not fatal, and the DMA channel continues operation.

Control Interface Error

The channel reports control-interface errors as bus errors to the bus master. This error can result from:

- An address error
- A register write error (write to a read-only register)

DMA Operating Modes

The DMA channel supports a number of different flow modes that control how the DMA channel progresses from one work unit to the next.

The flow mode of a DMA channel is not a global setting. A DMA descriptor set can include the descriptor responsible for configuring the flow of the work unit. There is no restriction, limiting the flow configuration to be the same for the entire descriptor chain. If the descriptor chain is not endless, the last descriptor set configures the flow to stop mode, which results in termination of the descriptor chain after the work unit completes. Another example for mixing flow modes is to create an endless descriptor-array. The configuration of the last descriptor set in the array selects the descriptor-list mode. The next descriptor pointer in this set of descriptors points to the first descriptor in the array.

Register-Based Flow Modes

Register-based DMA operations require configuration by directly writing to the memory-mapped registers of the DMA channel.

Register-based DMA is the traditional method of DMA operation. Software writes all of the configuration of the DMA channel into the memory-mapped registers. This configuration includes information such as the source or destination address and length of the data in the transfer. The DMA controller then starts channel operation. The DMA channel supports the following register-based flow modes.

- Stop Mode
- Autobuffer Mode

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor. The supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to the descriptor set size being configurable, descriptor-based DMA also allows altering the flow mode of the next descriptor set. This feature allows for the transition from descriptor-array mode to descriptor-list mode and permits configuring the flow to stop or autobuffer mode.

Stop Mode

In stop mode, the DMA operation executes only once. If started, the DMA channel transfers the desired number of data words and stops itself again when finished. If the DMA channel is no longer used, software configures the enable bit to disable a paused channel. The channel also can generate interrupts and triggers for each row or work unit completion, depending on the desired operation.

Autobuffer Mode

In autobuffer mode, the DMA operates repeatedly in a circular manner. If the transfer of all data words completes, the channel reloads the address pointer (`DMA_ADDR_CUR`) automatically with the `DMA_ADDRSTART` value. The channel also can generate an interrupt.

The `DMA_CFG.FLOW` field enables autobuffer mode. The configuration must load the `DMA_CFG.NDSIZE` field value, such that the next descriptor size is zero.

Descriptor-Based Flow Modes

Descriptor-based DMA operations fetch descriptor sets from memory allowing for autonomous loading of work units on other work units. Software does not need to set up the DMA sequences directly by writing into the DMA controller registers. Rather, software keeps DMA descriptor sets in memory.

Descriptor-based DMA operations have the following additional attributes.

- The DMA controller autonomously loads the descriptor set from memory to the affected DMA controller registers on demand.
- The channel can fetch descriptor sets from any memory space that supports DMA read operations.
- The descriptor set describes the next operation that the DMA controller performs.
- The descriptor set can include information such as the DMA configuration word as well as data source or destination address, transfer count, and address modify values.

A descriptor set describes a single work unit. The next work unit can reuse some values from the previous one descriptor set. But, this reusage is possible only if they are not overwritten in the subsequent descriptor set fetches and only if the work unit requires the use of this descriptor.

The DMA channel supports the following flow modes with descriptor-based operations.

- Descriptor-Array Mode
- Descriptor-List Mode
- Descriptor-On-Demand Modes

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor and the supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to configurable descriptor set size, descriptor-based DMA also allows for altering of the flow mode of the next descriptor set. Programs can transition from one descriptor-based mode to another descriptor-based mode and can also transition to any of the register-based flow modes.

Descriptor-Array Mode

When configured in this mode, the descriptor sets do not contain further descriptor pointers. Software writes the initial descriptor-pointer value, which points to an array of descriptors. This operation assumes that the individual descriptors reside next to each other and assumes that their addresses are known.

The *Offsets for Descriptor-Array Mode Parameters and Descriptors* table illustrates how to structure a descriptor set in memory. The descriptor sets must reside in a contiguous block or memory in the format shown in the table. Locate the first descriptor of the next descriptor set in the memory location immediately following the last descriptor of the current descriptor set. The values have the same order as the corresponding offset addresses of the memory-mapped register.

Table 38-9: Offsets for Descriptor -Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT

Table 38-9: Offsets for Descriptor -Array Mode Parameters and Descriptors (Continued)

Descriptor Offset	Parameter Register
0x14	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current registers between the descriptor set fetch and the start of the DMA operation for the work unit.

NOTE: At a minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Descriptor-List Mode

In this flow mode, multiple descriptors form a chained list in which each descriptor set contains a pointer to the next descriptor set, allowing greater flexibility in memory layout options. When the channel fetches the descriptor set, the operation loads this pointer value into the next descriptor pointer register of the DMA channel.

Descriptor Sets

The ***Offsets for Descriptor-List Mode Parameters and Descriptors*** table shows how to structure a descriptor set in memory. Disperse the placement of the descriptor sets throughout memory, having sets reside in different memory blocks. But, each descriptor of the descriptor set must reside in a contiguous section of memory in the format shown in the table. The values have the same order as the corresponding offset addresses of the memory-mapped registers.

Table 38-10: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current values of the registers between the descriptor set fetch and the start of the DMA operation for the work unit.

Minimum Startup Requirements

At a minimum, write the `DMA_DSCPTR_NXT` register prior to write to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Descriptor-On-Demand Modes

The Descriptor-Array Mode and Descriptor-List Mode each have an on-demand mode of operation.

In on-demand mode, at the end of the work unit, if the DMA channel has not detected an incoming trigger event, the channel repeats the current work unit. If the DMA channel receives an incoming trigger before completion of the work unit, the channel fetches a new descriptor set.

The *Offsets for Descriptor-Array Mode Parameters and Descriptors* and *Offsets for Descriptor-List Mode Parameters and Descriptors* tables illustrate how to structure each descriptor set in memory.

Table 38-11: Offsets for Descriptor-Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

NOTE: For descriptor-array mode, at a minimum, write the `DMA_DSCPTR_CUR` register prior to writing to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Table 38-12: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

NOTE: For descriptor-list mode, at a minimum, write the `DMA_DSCPTR_NXT` register prior to write to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Data Transfer Modes

In addition to supporting basic one-dimensional DMA transfers, the DMA channel also supports two-dimensional functionality.

Two-Dimensional DMA

Register-based flow modes and descriptor-based flow modes support two-dimensional data transfers.

In two-dimensional (2D) mode, the X-direction count (DMA_XCNT), the X-direction modifier (DMA_XMOD), the Y-direction count (DMA_YCNT), and the Y-direction modifier (DMA_YMOD) support arbitrary row and column sizes. Also, the modify values can be negative, allowing implementation of interleaved data streams. The DMA_XCNT value specifies the row size, and the DMA_YCNT value specifies the column size; where the DMA_XCNT value must be 2 or greater.

The DMA start address (DMA_ADDRSTART), the X-direction modifier (DMA_XMOD), and theY-direction modifier (DMA_YMOD) specifications all are in bytes. The alignment must be a multiple of the DMA transfer word size; configured using the DMA_CFG.MSIZE bit. Misalignment results in a DMA channel error.

The DMA_XMOD register value is the byte-address increment that the channel applies after each transfer, decrementing the DMA_XCNT register. The channel does not apply the DMA_XCNT when the inner loop count ends with the DMA_XCNT_CUR register decrementing to 0 from 1. Except, the channel does apply the DMA_XCNT on the final transfer, when the DMA_YCNT register is 1 and the DMA_XCNT register decrements from 1 to 0.

The DMA_YMOD register value is the byte-address increment that the channel applies after each decrement of the value in DMA_YCNT_CUR. However, the channel does not apply the DMA_YMOD value to the last item in the array on which the outer loop count (DMA_YCNT_CUR) also expires by decrementing from 1 to 0.

After the last transfer completes, DMA_YCNT_CUR is 1 and the DMA_XCNT_CUR register is 0. The DMA channels current address points to the last items address plus the DMA_XMOD register value. If the DMA channel programming selects automatic refresh (such as in autobuffer mode), the channel reloads the DMA_XCNT_CUR, DMA_YCNT_CUR, and DMA_ADDR_CUR for the first data transfer of the next work unit.

Interrupt notification is configurable for end-of-row or end-of-work unit completion.

For example, two-dimensional DMA can be used to extract interleaved data (such as RGB values for a video frame) by modifying both of the DMA_XMOD and DMA_YMOD values. The ***Capturing a Video Data Stream 2D DMA Example*** depicts the process of receiving a stream of the R, G, B values from an N*M frame. The inner loop of the 2D DMA configuration has three values (DMA_XCNT = 3) and a stride (DMA_XMOD) of N*M, chosen such that successive elements in each row are 1-2-3, 4-5-6 and so forth. The outer loop of the 2D DMA configuration has N*M values (DMA_YCNT = N*M) and a negative stride (DMA_YMOD) of 1-2*N*M chosen to instruct the DMA controller to jump from element 3 to 4, 6 to 7 and so forth at the end of each inner loop.

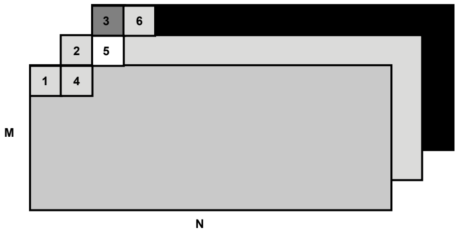


Figure 38-4: Capturing a Video Data Stream 2D DMA Example

DMA Channel Event Control

The DMA channel supports a number of events that provide notification of work unit state, peripheral data request, peripheral interrupt request and completion events, and DMA channel error conditions. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The DMA channel has two interrupt signals for support of a number of events such as work-unit state events, peripheral interrupt request (PIRQ) events, peripheral data request (PDR) events, and DMA channel errors. The channel reports DMA channel errors on a dedicated interrupt signal. All other interrupt sources share an interrupt signal. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The channel can signal the processor on DMA channel events using status information and optional interrupt requests. Programs can use these events to update the progress of data transfers and to request intervention from the processor core. Configure most DMA channel interrupts using bits in the `DMA_CFG` register. Dedicated bits in the `DMA_STAT` register report the occurrence of various events. Use write-one-to-clear (W1C) operations to clear interrupt requests from the status register.

> **NOTE:** Hardware does not clear the interrupt status bits automatically, even when programs disable then reenable the DMA channel. In this situation, the channel deasserts the interrupt signal, after the program disables the DMA channel. But, the status bit remains set until software either re-enables the DMA channel or clears the status bit.

The DMA channel supports the following categories of events on the interrupt signals:

- Work-unit state events generate interrupts on row or on work unit DMA completion.
- A peripheral uses peripheral interrupt request (PIRQ) events to signal when it has completed the transfer of all data.
- A peripheral uses peripheral data request (PDR) events to request data from a disabled or idle DMA channel.
- Error events signal a failure in the work unit.

> **ATTENTION:** While in an error state, the DMA channel does not generate an interrupt to the processor for a work-unit state event or a PIRQ event, nor does the channel forward a PDR event.

Event Signals

The *Event Signals* table provides descriptions of DMA channel events.

Table 38-13: Event Signals

Signal	Width (bits)	Description
DMA_ERROR	1	Used to signal an error condition in the DMA channel. The source of the error can be determined by reading the `DMA_STAT.ERRC` bit.

Table 38-13: Event Signals (Continued)

Signal	Width (bits)	Description
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. Read the corresponding fields in `DMA_STAT` to determine the source of the event.
DMA_TRIG_OUT	1	Trigger output that gets routed to the TRU and can be configured to provide notification on row or work unit completion.
DMA_TRIG_IN	1	Trigger input from the TRU that can be used to control the start of a work unit.

Work Unit State Events

Completing a row or a work unit generates a work-unit state event. For either of these events to generate an interrupt request, the configuration of the interrupt of the DMA channel must select one of the available work-unit completion modes.

- Current X count reaching 0 for row completion or 1D DMA work unit completion
- Current Y count reaching 0 for work unit completion of 2D DMA

NOTE: For 1D DMA, a DMA channel configuration error results if the configuration generates the interrupt request when the current Y counter reaches 0.

The DMA channel issues the last memory read or write transaction for the row or work unit, then pauses until the return of the read or write acknowledge. After successful acknowledge of the transfer, the DMA channel issues the interrupt request and continues to process the next row or work unit.

Waiting for acknowledgement of the memory access results in a delay. However, programs can read or modify data in the memory without adversely affecting or being affected by the DMA transfer.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

The channel configuration of the synchronization feature also affects interrupt timing. For memory-read operations with synchronization enabled, the channel delays the interrupt request until the completion of the last transfer from the DMA channel FIFO to the peripheral. The synchronization feature does not affect interrupt timing for memory write operations.

Peripheral Interrupt Request Events

For peripheral-transmit operations, a peripheral connected to the DMA channel can use peripheral interrupt request (PIRQ) events to indicate that data has left the channel FIFO and to indicate transfer completion.

In order to support PIRQ interrupts, correctly configure the interrupt of the DMA channel. This configuration disables the generation of interrupt requests based on the work unit state and, instead, results in generating an interrupt request when the DMA channel receives the command from the peripheral.

The channel only generates the interrupt request under the following conditions:

- The configuration enables the DMA channel
- The DMA channel is in the stop state
- The configuration of the DMA channel interrupt selects PIRQ operation

Peripheral Data Request Events

Peripheral data request (PDR) events occur when an interfaced peripheral requests data from the DMA channel and the DMA channel (either disabled or enabled) is in the stop state.

When a peripheral sends a data request command to a disabled DMA channel, the DMA channel generates an interrupt to the System Event Controller (SEC). There is no status information reported about this event in the status register of the DMA channel. Instead, the channel identifies the PDR event from the fact that the DMA channel generated an interrupt while disabled. It is possible to further confirm event status by verifying the status of the peripheral interfaced to the DMA channel.

This operation forwards data requests as interrupts when the DMA channel is in the disabled state. Also, the DMA channel is able to forward PDR events as an interrupt request when the DMA channel is in the stop state after the completion of a work unit. The forwarding of this interrupt when the DMA channel is in the stop state is optional and configured by the program during DMA channel configuration.

DMA Channel Triggers

DMA channel triggers are useful for synchronizing the DMA channel with other events in the system. One usage is to combine channel triggers with each other to create ping-pong buffers. Another usage is to combine the triggers with interrupt requests to notify the processor on reaching a particular milestone that requires service. The channel also can use triggers to enforce a handshake DMA operation in which the trigger acts as a signal for a DMA request.

> **NOTE:** Using the trigger to control the pace of data transfers, such as for handshake DMA, requires that all the data for the entire work unit is ready for transfer.

The DMA channel has a single incoming trigger that can control the pace of the data transfers performed by the DMA channel. The configuration can direct the DMA channel to wait for the incoming trigger before starting the work unit transfer or fetching a descriptor set from memory.

The DMA channel also has a single outgoing trigger signal. This configuration can direct this trigger to signal the end of row or an entire work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

Issuing Triggers

The DMA channel configuration can direct the channel to generate an outgoing trigger signal at the end of row or the end of a work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until the return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

Waiting For Triggers

Programs can use triggering to control the pace of data transfers performed by the DMA channel. The DMA channel enters a wait state before beginning the next work unit if the configuration enables `DMA_CFG.TWAIT` and either of the following apply:

- The channel receives a trigger since the last time the DMA channel left the wait state.
- The channel receives a trigger since its transition from disable to enable.

In the wait state, the DMA channel also does not perform a descriptor fetch. After receiving a trigger, the DMA channel leaves the wait state and begins the next work unit or fetches the next descriptor if configured for a descriptor-based mode of operation.

If a memory-mapped register write operation programs the channel with stop flow mode enabled (`DMA_CFG.TWAIT` bit) and the channel has not already received a trigger, the DMA channel enters a wait state before performing the data transfer. On receiving the trigger, the DMA channel begins the data transfer portion of the work unit. Once the data transfer is complete, the DMA channel enters the stop state.

If a memory-mapped register write operation programs the DMA channel with the flow mode configured to one of the descriptor-based modes, the DMA channel enters the wait state before performing the descriptor fetch. After completing the descriptor fetch, the DMA channel immediately proceeds to the data transfer, regardless of the value of the `DMA_CFG.TWAIT` bit. If another (next) descriptor fetch follows the descriptor fetch, the DMA channel enters a wait state before fetching the next descriptor.

If the descriptor fetch returns a descriptor with stop flow mode, the `DMA_CFG.TWAIT` value for that descriptor does not affect the DMA as the channel enters the stop state after completing the data transfer. The DMA channel only enters the wait state based on `DMA_CFG.TWAIT` before the next work unit or descriptor fetch.

If the descriptor fetch returns a descriptor configured for autobuffer flow mode, the `DMA_CFG.TWAIT` for that descriptor does not affect the DMA for the first work unit of the autobuffer transfer. After completing the first work unit and not receiving another trigger, the DMA channel enters the wait state before reinitializing its counters and address registers (if not configured for current addressing). The channel performs the next work unit after receiving the trigger.

The incoming trigger can occur when the DMA channel has not entered the wait state. The trigger can occur while the DMA channel is executing a work unit, is performing descriptor fetch, or is in the stop state. The trigger is held internally. After the work unit is complete, the DMA channel skips the wait state and proceeds directly to executing the following work unit. If the `DMA_CFG.TWAIT` bit is not enabled, the DMA channel also skips the wait state. However, the trigger is held internally and is used the next time the configuration enables `DMA_CFG.TWAIT`. This trigger retention allows programs to enable the `DMA_CFG.TWAIT` functionality several work units apart without concern for losing a trigger. The DMA channels trigger-overrun enable functionality can be enabled in all work units to ensure that multiple triggers do not occur between the work units with the `DMA_CFG.TWAIT` bit enabled.

DMA Channel Programming Model

Several synchronization and control methods are available for use in development of software tasks which manage peripheral DMA and memory DMA. Software must accept requests for new DMA transfers from other software tasks, integrate these transfers into existing transfer queues, and reliably notify other tasks when the transfers are complete.

In the processor, it is possible to manage each peripheral DMA and memory DMA stream with a separate task or to manage them together with any other stream. Each DMA channel has independent, orthogonal control registers, resources, and interrupts. So, the selection of the control scheme for one channel does not affect the choice of control scheme on other channels. For example, one peripheral can use a linked-descriptor-list, interrupt-driven scheme while another peripheral can simultaneously use a demand-driven, buffer-at-a-time scheme synchronized by polling DMA events.

The topics that follow describe the steps required to configure the DMA channel for the various modes in addition to the programming concepts required for software synchronization.

Mode Configuration

Use the step-by-step directions that follow to set up the DMA channel for operating modes.

Register-Based Linear-Buffer Stop Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and to send it to the peripheral for transmission.

Assume that the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_ADDRSTART` register.

   ADDITIONAL INFORMATION: Software can use the address to calculate the most optimum possible `DMA_CFG.MSIZE`.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE`, and the calculation also must consider the start address alignment.

3. Write the `DMA_XCNT` register based on the calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: The `DMA_XCNT` value is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. Always specify this register as a number of bytes.

5. Write the `DMA_CFG` register with `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` bit for STOP mode. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus.

   - The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
   - Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer is transferred. The DMA channel enters the IDLE state upon completion of the work unit.

Register-Based Autobuffer Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and send it to the peripheral for transmission. The transmission of the buffer repeats endlessly.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel restarts the DMA operation, creating an endless circular buffer transfer.

1. Write the `DMA_ADDRSTART` register.

   ADDITIONAL INFORMATION: Use the address to calculate the optimum possible `DMA_CFG.MSIZE`.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE`, and the calculation must consider the start address alignment.

3. Write the `DMA_XCNT` register based on calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: The `DMA_XCNT` register value is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. Always specify this register as a number of bytes.

5. Write the `DMA_CFG` register with the `DMA_CFG.EN` bit configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` bit for autobuffer mode. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bit to a value no larger than the supported bus width of the peripheral DMA bus.

   - The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
   - Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer transfers until the DMA channel is disabled.

Descriptor-Array Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the array, and
- Send the data to the peripheral for transmission.

Descriptor sets are read from an array in memory to configure the individual work units.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel. Assume that the array of descriptors is to be initialized with the last descriptor set configured for STOP flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the array in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_DSCPTR_CUR` register with the address of the array in which the descriptor sets are stored.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the `DMA_CFG` register with the `DMA_CFG.EN` bit configured to enable the DMA channel.

ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` bit for descriptor-array mode. Configure the `DMA_CFG.NDSIZE` bits to describe the number of descriptor elements contained within the first descriptor set. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bits to a value no larger than the supported bus width of the peripheral DMA bus.

- The descriptor set that is fetched controls the `DMA_CFG.SYNC` configuration and the interrupt or trigger configurations.

The first descriptor set is fetched from memory location provided by the `DMA_DSCPTR_CUR` register and loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor array. The DMA channel enters the IDLE state on completion of the final work unit that was configured for STOP flow mode.

Descriptor-List Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the list, and
- Send it to the peripheral for transmission.

The DMA controller reads the descriptor sets from a list of descriptors. With the list, each descriptor set has a descriptor that points to the next descriptor set location in memory.

Assume the peripheral must be in a state where it is ready to transmit data received from the DMA channel. Assume that the list of descriptors must be initialized with the last descriptor set in the list configured for Stop flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the list in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_DSCPTR_NXT` register with the address of the first descriptor in the list to be processed.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the `DMA_CFG` register with the `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the `DMA_CFG.FLOW` for descriptor-list mode. Configure the `DMA_CFG.NDSIZE` bit to describe the number of descriptor elements contained within the first descriptor set. Configure the `DMA_CFG.WNR` bit for memory read operation. Configure the `DMA_CFG.PSIZE` bit to a value no larger than the supported bus width of the peripheral DMA bus.

- The descriptor set that is fetched controls the `DMA_CFG.SYNC` configuration and controls the interrupt or trigger configurations.

The first descriptor set is fetched from the memory location provided by `DMA_DSCPTR_NXT` and is loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor list. The DMA channel enters the idle state when the final work unit that was configured for stop-flow mode is complete.

Register-Based Memory-to-Memory Transfer in Stop Flow Mode

This procedure configures a memory DMA channel pair in stop flow mode. One DMA channel is configured for memory read operations, while the other DMA channel is configured for memory write.

The task involves writing to a number of DMA channels on two DMA channels that create a memory DMA pair. On DMA completion, the DMA channel enters the idle state, until either the DMA channel is disabled or is reconfigured for a new transfer.

1. Write the `DMA_ADDRSTART` register of the source DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the optimum `DMA_CFG.MSIZE` possible.

2. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE` and the start address alignment must also be considered.

3. Write the `DMA_XCNT` register of the source DMA channel based on calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

4. Write the `DMA_XMOD` register of the source DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

5. Write the `DMA_ADDRSTART` register of the destination DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

6. Calculate the optimum `DMA_CFG.MSIZE` based on the `DMA_ADDRSTART` register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected `DMA_CFG.MSIZE` and the start address alignment must also be considered.

7. Write the `DMA_XCNT` register of the destination DMA channel based on the calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers to make up the entire work unit.

8. Write the `DMA_XMOD` register of the destination DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in `DMA_XMOD` from the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

9. Write the `DMA_CFG` register of the source DMA channel with `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: The `DMA_CFG.FLOW` bit must be set for stop mode. The `DMA_CFG.WNR` bit must be configured for memory read operation. The `DMA_CFG.PSIZE` bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - The `DMA_CFG.SYNC` bit can be configured to control DMA completion notification timing.
   - Interrupts and triggers also can be configured at this step, depending on requirements. The interrupts and triggers are enabled within the destination DMA channel configuration.

   The memory read DMA transfer begins.

10. Write the `DMA_CFG` register of the destination DMA channel with `DMA_CFG.EN` configured to enable the DMA channel.

    ADDITIONAL INFORMATION: The `DMA_CFG.FLOW` bit must be set for stop mode. The `DMA_CFG.WNR` bit must be configured for memory write operation. The `DMA_CFG.PSIZE` bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus. This value must also match the value written for the source DMA channel configuration.

    - Interrupts and triggers also can be configured at this step depending on requirements.

    The memory write DMA transfer begins.

Both memory DMA channels are now running and the data is transferred from the source address to the destination address. The DMA channel enters the IDLE state upon completion of the work unit.

Programming Concepts

Using the features, operating modes, and event control for the DMA channel to their greatest potential requires an understanding of some DMA channel-related concepts.

Synchronization of Software and DMA

A critical element of software DMA management is the synchronization of DMA work unit completion with software. This synchronization can be achieved using DMA channel interrupt request and trigger events and using a poll of the status bits of these events within the DMA channel registers, or combining these techniques. Processor polling of DMA address/count/status for completion is not a recommended programming practice. The requirements and limitations of processor polling place significant responsibility onto the code developer to be deeply aware of the underlying hardware. The interrupt requests and triggers are designed for efficient code development and reuse.

Interrupt and Trigger Event-Based Synchronization

Interrupt and trigger based synchronization methods must avoid overrun. An overrun occurs when some events fail to invoke the event handler of a DMA channel for every event due to excessive latency in processing of events. The system design must ensure to either:

- Schedule only one event per channel (for example, at the end of a descriptor list), or
- Space the generated events sufficiently far apart in time that system processing budgets can guarantee service of every event.

The DMA channel issues status information through an interrupt request or trigger event or changes event status bits in the `DMA_STAT` register. This status guarantees that the last memory operation of the work unit is complete. For memory read DMA transactions, this status means that the FIFO of the DMA channel safely receives the final memory read data. For DMA transactions writing to memory, this status indicates that the DMA channel received an acknowledge of completion of the last write transfer of the work unit.

Register Polling Based Synchronization

Do not poll the DMA channel registers (`DMA_ADDR_CUR`, `DMA_DSCPTR_CUR`, `DMA_XCNT_CUR`, or `DMA_YCNT_CUR`) as a method of precisely synchronizing DMA with data processing. This approach is inaccurate due to the operation of the DMA channel FIFOs and DMA or memory pipelining. The current address, pointer, and count registers change several cycles in advance of the completion of the corresponding memory operation. This timing is measurable from the time at which the results of the operation are first visible to the core by memory read or write instructions.

For example, in a DMA channel memory write operation to external memory, assume DMA channel *A* initiates a DMA channel write operation. For memories with access latency, this operation requires many system-clock cycles. Meanwhile, DMA channel *B* (which does not in itself incur latency) initiates a transfer, which stalls behind the slow operation of channel *A*. Software monitoring channel *B* could not safely conclude whether the memory location pointed to by the `DMA_ADDR_CUR` of channel *B*. Also, the software cannot conclude whether the register has been written based solely on the contents of this register.

Polling of the current address, pointer, and count registers can permit loose synchronization of DMA with software. But, the software must allow for the lengths of the DMA or memory pipeline. Also, software must consider the length of the DMA FIFO for a particular peripheral. If the FIFOs are filled with incomplete work, the DMA channel does not advance current address, pointer, or count registers. The incomplete work includes reads that have been started but have not yet finished.

Additionally, software must consider the length of the pipelines to the destination memory. If the DMA FIFO length and channel memory-pipeline length are added, software can estimate the maximum number of incomplete memory operations in progress.

NOTE: The estimate would be a maximum, as the DMA or memory pipeline can include traffic from other DMA channels.

Descriptor Queues

A system designer may want to write a DMA manager facility which accepts DMA requests from other software. The DMA manager software does not know in advance when new work requests are received or what these requests contain. The software could manage these transfers using a circular linked list of DMA descriptors. In such a list, each descriptor sets the `DMA_DSCPTR_NXT` descriptor, which points to the next descriptor set. And, the last descriptor set in the list points to the first descriptor set.

The code that writes into this descriptor list could use the circular addressing modes of the processor. This approach does not need to use comparison and conditional instructions to manage the circular structure. In this case, the `DMA_DSCPTR_NXT` descriptor of each descriptor set can be written once at startup, and skipped over as new contents are written for each descriptor.

The recommended method for synchronization of a descriptor queue is to use an interrupt or trigger. The descriptor queue is structured, such that (at least) the final valid descriptor set is always programmed to generate an interrupt or trigger event upon completion. More detail is provided in the following sections.

- Queues Using Event Generation for Every Descriptor Set
- Queues Using Minimal Events

Queues Using Event Generation for Every Descriptor Set

In this system, the DMA manager software synchronizes with the DMA channel by enabling an interrupt request or trigger on every descriptor set. Only use this method if the system design can guarantee that each work unit completion event is serviced separately (no interrupt or trigger overrun).

To maintain synchronization of the descriptor set queue, the non-interrupt software maintains a count of descriptor sets added to the queue. The event handler (either interrupt or trigger) maintains a count of completed descriptor sets removed from the queue. The counts are equal only when the DMA channel is paused after having processed all the descriptor sets.

When each new work unit event is received, the DMA manager software initializes a new descriptor set, taking care to set the flow to stop mode. Next, the software compares the descriptor set counts to determine whether the DMA channel is running. If the DMA channel is paused (counts equal), the software increments its count. Then, the software starts the DMA channel by writing the `DMA_CFG` of the new descriptor set.

If the counts are unequal, the software instead modifies the `DMA_CFG` of the next-to-last descriptor set, such that it now describes the newly queued descriptor set. This operation does not disrupt the DMA channel provided the rest of the descriptors of the set are initialized in advance. It is necessary to synchronize the software to the DMA to determine whether the DMA channel read the new or the old `DMA_CFG` value.

The event handler performs the synchronization operation. When an event is detected, the handler reads the `DMA_STAT` register of the DMA channel. If the `DMA_STAT.RUN` bit indicates that the DMA channel is running, the channel has moved on to processing another descriptor. The event handler can increment its count and exit. If the `DMA_STAT.RUN` bit indicates that the channel is not running, the channel is paused because either:

- There are no more descriptor sets to process, or
- The last descriptor set was queued too late

Where *too late* means that the modification of the `DMA_CFG` of the next-to-last descriptor set occurred *after* that descriptor was read into the DMA channel. In this case, the event handler does the following:

- Writes the `DMA_CFG` value appropriate for the last descriptor set to `DMA_CFG` register of the DMA channel,
- Increments the completed descriptor count, and
- Exits

If the event latencies of the system are large enough to cause any of the events to be dropped, this system can fail. An event handler capable of safely synchronizing multiple descriptor set interrupt requests is complex, performing several MMR accesses to ensure robust operation. In such a system environment, a minimal event synchronization method is preferred.

Queues Using Minimal Events

In this system, only one DMA interrupt request or trigger event is generated in the queue at any time. The DMA event handler for this system can also be extremely short. Here, the descriptor queue is organized into an *active* and a *waiting* portion, where events are enabled only on the last descriptor set in each portion.

When each new DMA request is processed, the software fills in the content of a new descriptor set and adds it to the waiting portion of the queue. The `DMA_CFG` descriptor of the descriptor set must have the flow set to stop mode. If more than one request is received before the DMA queue completion event occurs, the non-interrupt code queues later descriptor sets. It forms a waiting portion of the queue separate from the active portion of the queue that the DMA channel is processing. In other words, all but the last active descriptor sets contain flow values for a descriptor-based mode and have no event enable set.

The last active descriptor set has the stop flow mode and an event generation enabled. Also, all but the last waiting descriptor sets are configured for descriptor-based flow modes with no event generation. Only the last waiting descriptor set is configured for stop flow mode and event generation enabled. This configuration ensures that the DMA channel can automatically process the whole active queue before then issuing one event. Also, this arrangement makes it easy to start the waiting queue within the event handler by a single `DMA_CFG` register write.

After queuing a new waiting descriptor, the non-interrupt software leaves a message for its interrupt handler in a memory mailbox location. The location contains the desired `DMA_CFG` value for starting the first waiting descriptor set in the waiting queue (or 0, indicating no waiting descriptors).

The software must not modify the contents of the active descriptor set queue directly once processing by the DMA channel has started, unless careful synchronization measures are taken. In the most straightforward implementation of a descriptor set queue, the DMA manager software never modifies descriptors on the active queue. Instead, the DMA manager waits until the DMA queue completion event indicates that the processing of the entire active queue is complete.

When a DMA queue completion event is received, the event handler reads the mailbox from the non-interrupt software and writes the value to the `DMA_CFG` register of the DMA channel. This write to a register restarts the queue, effectively transforming the waiting queue to an active queue. The event handler then passes a message back to the non-interrupt software indicating the location of the last descriptor set accepted into the active queue.

However, the event handler can read its mailbox and find a DMA_CFG value of zero, indicating there is no more work to perform. It then passes an appropriate message back to the non-interrupt software indicating that the queue has stopped.

The non-interrupt software which accepts new DMA work unit requests must synchronize the activation of a new work unit with the interrupt handler. If the queue has stopped (the mailbox from the event handler is zero), the non-interrupt software must start the queue. (The queue starts by writing the first descriptor sets DMA_CFG value to the DMA_CFG register of the channel). If the queue is not stopped, the non-interrupt software must not write the DMA_CFG register. (This write causes a DMA error). Instead, it must queue the descriptor onto the waiting queue and update its mailbox directed to the event handler.

ADSP-SC58x DMA Register Descriptions

The Direct Memory Access module (DMA) contains the following registers.

Table 38-14: ADSP-SC58x DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

Start Address of Current Buffer Register

The `DMA_ADDRSTART` register contains the start address of the work unit currently targeted for DMA. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

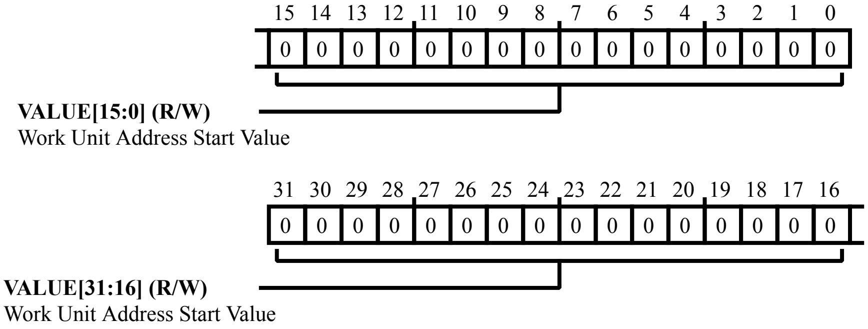

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Work Unit Address Start Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Work Unit Address Start Value

Figure 38-5: DMA_ADDRSTART Register Diagram

Table 38-15: DMA_ADDRSTART Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Address Start Value. The `DMA_ADDRSTART.VALUE` bit field contains the start address of the work unit currently targeted for DMA.

Current Address Register

The DMA_ADDR_CUR register contains the present memory transfer address for a given work unit. At the start of a work unit, the DMA_ADDR_CUR register is loaded from the DMA_ADDRSTART register, and the DMA_ADDR_CUR register is incremented as each transfer occurs. The DMA_ADDR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel.

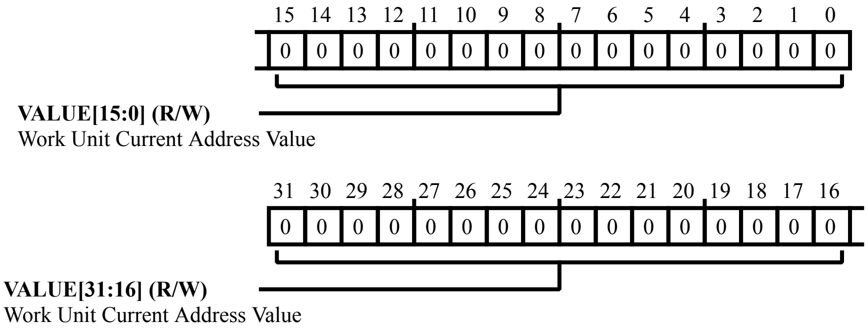

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Work Unit Current Address Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Work Unit Current Address Value

Figure 38-6: DMA_ADDR_CUR Register Diagram

Table 38-16: DMA_ADDR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Current Address Value. The `DMA_ADDR_CUR.VALUE` bit field contains the present memory transfer address for a given work unit.

Bandwidth Limit Count Register

The DMA_BWLCNT register contains a count that determines how often the DMA issues memory transactions. The DMA loads the value from the DMA_BWLCNT register into the DMA_BWLCNT_CUR register and decrements the current value each SCLK cycle. When DMA_BWLCNT_CUR reaches 0x0000, the next request is issued, and the DMA reloads DMA_BWLCNT_CUR. This bandwidth limit functionality is not applied to descriptor fetch requests. Programming 0x0000 allows the DMA to request as often as possible. 0xFFFF is a special case and causes requests to stop.

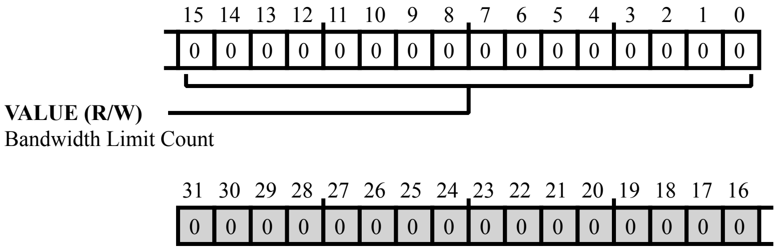

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Bandwidth Limit Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 38-7: DMA_BWLCNT Register Diagram

Table 38-17: DMA_BWLCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Bandwidth Limit Count. The `DMA_BWLCNT.VALUE` bit field contains a count that determines how often the DMA issues memory transactions.

Bandwidth Limit Count Current Register

The `DMA_BWLCNT_CUR` register contains the number of SCLK count cycles remaining before the next request is issued.

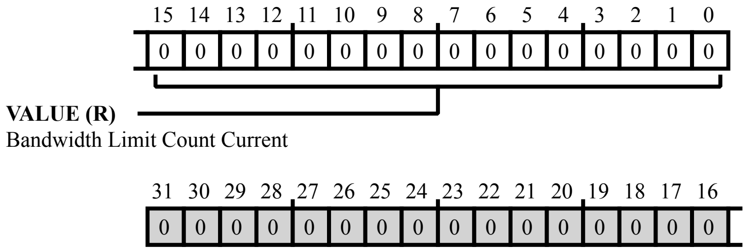

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Bandwidth Limit Count Current
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 38-8: DMA_BWLCNT_CUR Register Diagram

Table 38-18: DMA_BWLCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Bandwidth Limit Count Current. The `DMA_BWLCNT_CUR.VALUE` bit field contains the number of SCLK count cycles remaining before the next request is issued.

Bandwidth Monitor Count Register

The `DMA_BWMCNT` register contains the maximum number of SCLK cycles allowed for a work unit to complete. Each time the `DMA_CFG` register is written (MMR access only), a work unit ends or an autobuffer wraps. The DMA loads the value in this register into the `DMA_BWMCNT_CUR` register.

The DMA decrements `DMA_BWMCNT_CUR` every SCLK a work unit is active. If the `DMA_BWMCNT_CUR` register reaches 0x0000_0000, the `DMA_STAT.IRQERR` bit is set, and the `DMA_STAT.ERRC` bit field is set to 0x6. The `DMA_BWMCNT_CUR` remains at 0x0000_0000 until it is reloaded when the work unit completes.

Unlike other errors, a bandwidth monitor error does not stop work unit processing. Programming 0x0000_0000 disables bandwidth monitor functionality.

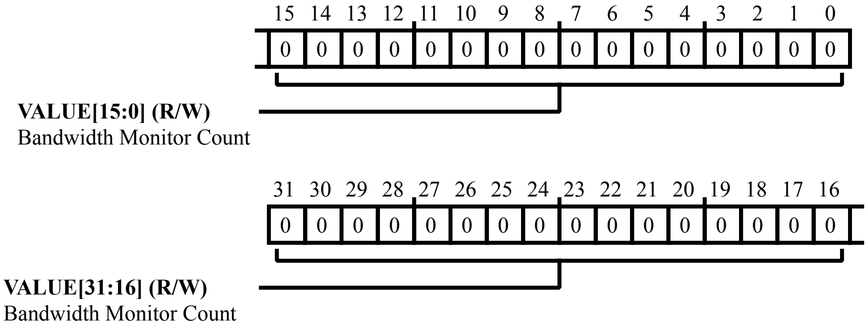

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Bandwidth Monitor Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Bandwidth Monitor Count

Figure 38-9: DMA_BWMCNT Register Diagram

Table 38-19: DMA_BWMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Bandwidth Monitor Count. The `DMA_BWMCNT.VALUE` bit field contains the maximum number of SCLK cycles allowed for a work unit to complete.

Bandwidth Monitor Count Current Register

The `DMA_BWMCNT_CUR` register contains the number of cycles remaining for the current descriptor to complete.

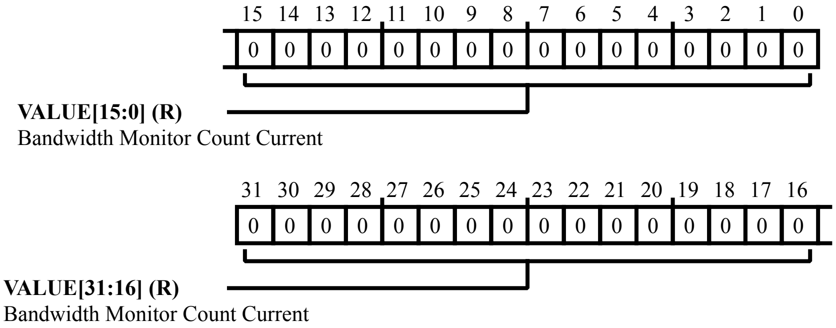

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Bandwidth Monitor Count Current
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Bandwidth Monitor Count Current

Figure 38-10: DMA_BWMCNT_CUR Register Diagram

Table 38-20: DMA_BWMCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bandwidth Monitor Count Current. The `DMA_BWMCNT_CUR.VALUE` bit field contains the number of cycles remaining for the current descriptor to complete.

Configuration Register

The `DMA_CFG` register sets up DMA parameters and operation modes. Writing to the `DMA_CFG` register while a DMA process is already running causes a DMA error (except when clearing the `DMA_CFG.EN` bit).

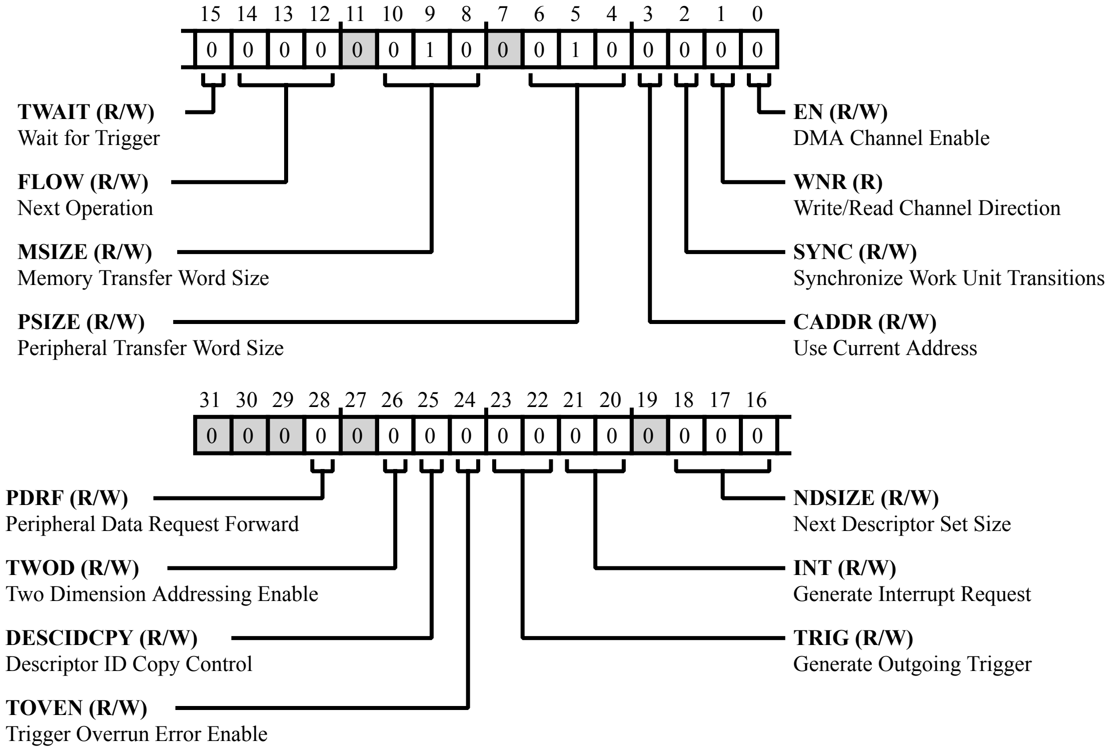

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
TWAIT (R/W)
Wait for Trigger
FLOW (R/W)
Next Operation
MSIZE (R/W)
Memory Transfer Word Size
PSIZE (R/W)
Peripheral Transfer Word Size
EN (R/W)
DMA Channel Enable
WNR (R)
Write/Read Channel Direction
SYNC (R/W)
Synchronize Work Unit Transitions
CADDR (R/W)
Use Current Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PDRF (R/W)
Peripheral Data Request Forward
TWOD (R/W)
Two Dimension Addressing Enable
DESCIDCPY (R/W)
Descriptor ID Copy Control
TOVEN (R/W)
Trigger Overrun Error Enable
NDSIZE (R/W)
Next Descriptor Set Size
INT (R/W)
Generate Interrupt Request
TRIG (R/W)
Generate Outgoing Trigger

Figure 38-11: DMA_CFG Register Diagram

Table 38-21: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	PDRF	Peripheral Data Request Forward. The `DMA_CFG.PDRF` bit defines how the DMA handles data requests from the peripheral while in idle state after a stop mode or memory read work unit. If set, the DMA forwards the peripheral data request as an interrupt. This bit applies only to the `DMA_CFG.FLOW` bits configured for stop and `DMA_CFG.WNR` bits configured for memory read.	
		0	Peripheral Data Request Not Forwarded
		1	Peripheral Data Request Forwarded

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	TWOD	Two Dimension Addressing Enable. The `DMA_CFG.TWOD` bit selects whether the DMA addressing involves only `DMA_XCNT` and `DMA_XMOD` (one-dimensional DMA) or also involves `DMA_YCNT` and `DMA_YMOD` (two-dimensional DMA).	
		0	One-Dimensional Addressing
		1	Two-Dimensional Addressing
25 (R/W)	DESCIDCPY	Descriptor ID Copy Control. The `DMA_CFG.DESCIDCPY` bit specifies when to copy the initial descriptor pointer to the `DMA_DSCPTR_PRV` register. A bus write to the `DMA_CFG` register to clear the `DMA_CFG.EN` bit causes the DMA to immediately use the new value of the `DMA_CFG.DESCIDCPY` bit. To preserve consistency (if required by application), match the new value of this bit to the previous value.	
		0	Never Copy
		1	Copy on Work Unit Complete
24 (R/W)	TOVEN	Trigger Overrun Error Enable. A trigger overrun occurs if more than one trigger was received before the DMA reached the trigger wait state. If `DMA_CFG.TOVEN` is set, a trigger overrun causes the DMA to flag an error. In cases where a trigger overrun is not a problem, clearing `DMA_CFG.TOVEN` prevents the overrun from causing an error and halting the DMA. The `DMA_CFG.TOVEN` operates independently of the `DMA_CFG.TWAIT` bit selection.	
		0	Ignore Trigger Overrun
		1	Error on Trigger Overrun

Table 38-21: DMA_CFG Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="5">23:22
(R/W)</td><td rowspan="5">TRIG</td><td colspan="2">Generate Outgoing Trigger.
The DMA_CFG.TRIG selects whether the DMA issues an outgoing trigger, based on the work unit counter values. In one-dimensional mode, the only options are to trigger at the end of the whole work unit (trigger when DMA_XCNT_CUR reaches 0) or not to trigger at all. If in one-dimensional addressing mode, programming the DMA_CFG.TRIG bit field to trigger when DMA_YCNT_CUR reaches 0 (or to reserved) causes the DMA to flag a configuration error.
In two-dimensional addressing mode, the trigger options are: at the end of each row of the inner loop (when DMA_XCNT_CUR reaches 0), only after completing the whole work unit (when DMA_YCNT_CUR reaches 0), or no trigger. If in two-dimensional mode and set to trigger when DMA_XCNT_CUR reaches 0, the DMA also issues a trigger at the end of the work unit. If in two-dimensional addressing mode, programming DMA_CFG.TRIG to reserved causes the DMA to flag a configuration error.
If DMA_CFG.TRIG is non-zero and the peripheral issues a finish command, the DMA issues a trigger after the finish procedure is complete.
For more information about trigger generation timing, see the trigger section of the DMA functional description.</td></tr>
<tr><td>0</td><td>Never Assert Trigger</td></tr>
<tr><td>1</td><td>Trigger When XCNTCUR Reaches 0</td></tr>
<tr><td>2</td><td>Trigger When YCNTCUR Reaches 0</td></tr>
<tr><td>3</td><td>Reserved</td></tr>
</table>

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	INT	Generate Interrupt Request. The `DMA_CFG.INT` bit field selects whether an interrupt request goes to the core based on work unit status or a peripheral interrupt request. For one-dimensional mode, setting the `DMA_CFG.INT` bits to generate an interrupt request when the `DMA_YCNT_CUR` register reaches 0 causes the DMA to flag a configuration error. The peripheral interrupt setting lets the DMA generate the last grant indication and to accept and or forward the peripheral interrupt command. The peripheral interrupt selection applies only to the `DMA_CFG.FLOW` bits set for stop and the `DMA_CFG.WNR` bits set for memory read. If the `DMA_CFG.INT` is set for interrupt request on count completion (`DMA_XCNT_CUR` or `DMA_YCNT_CUR` reach 0) and the peripheral issues a finish command, the DMA issues an interrupt request after the finish procedure is complete. For more information, see the sections on interrupt generation and peripheral control in the DMA functional description.	
		0	Never Assert Interrupt
		1	Interrupt When X Count Expires
		2	Interrupt When Y Count Expires
		3	Peripheral Interrupt request
18:16 (R/W)	NDSIZE	Next Descriptor Set Size. The `DMA_CFG.NDSIZE` bit field specifies the number of descriptor elements in memory to load during the next descriptor fetch. The DMA loads the descriptors in a specific order. The descriptor set contains the next descriptor pointer when it is a descriptor list. The descriptor set does not contain the next descriptor pointer when it is a descriptor array.	
		0	Fetch One Descriptor Element
		1	Fetch Two Descriptor Elements
		2	Fetch Three Descriptor Elements
		3	Fetch Four Descriptor Elements
		4	Fetch Five Descriptor Elements
		5	Fetch Six Descriptor Elements
		6	Fetch Seven Descriptor Elements
		7	Reserved

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	TWAIT	Wait for Trigger. The `DMA_CFG.TWAIT` bit controls whether the DMA waits for an incoming trigger from another channel or user. If the `DMA_CFG.TWAIT` bit is set, the DMA enters the wait state before starting the next work unit, including descriptor fetch when in descriptor mode. Do not use the wait for trigger control for descriptor-on-demand mode which causes an error. For more information, see the trigger section of the DMA functional description.	
		0	Begin Work Unit Automatically (No Wait)
		1	Wait for Trigger (Halt before Work Unit)
14:12 (R/W)	FLOW	Next Operation. The `DMA_CFG.FLOW` bit field selects the descriptor handling options.	
		0	STOP. See the Stop Flow Mode section.
		1	AUTO. See the Autobuffer Flow Mode section.
		2	Reserved
		3	Reserved
		4	DSCL. See the Descriptor List Mode section.
		5	DSCA. See the Descriptor Array Mode section.
		6	Descriptor On-Demand List. See the Descriptor List Mode section.
		7	Descriptor On Demand Array. See the Descriptor On Demand section.
10:8 (R/W)	MSIZE	Memory Transfer Word Size. The `DMA_CFG.MSIZE` bits select memory transfer sizes of 8-, 16-, 32-, 64-, 128-, or 256-bit words. The transfer start address (`DMA_ADDRSTART`) and transfer increment values (`DMA_XMOD`, and, if needed, `DMA_YMOD`) must be a multiple of the memory transfer unit size.	
		2	4 Bytes
		3	8 Bytes
		4	16 Bytes
		5	32 Bytes

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	PSIZE	Peripheral Transfer Word Size. The `DMA_CFG.PSIZE` bits select peripheral transfer sizes as 8, 16, 32, or 64 bits wide. Each request and grant results in a single peripheral access. There are no bursts on the peripheral bus, so the `DMA_CFG.PSIZE` selection must be less than or equal to the width of the bus. If the selection is greater than the bus width, a configuration error occurs. The peripheral bus of the processor is dedicated to DMA and peripheral accesses.	
		2	4 Bytes
		3	8 Bytes
3 (R/W)	CADDR	Use Current Address. When the `DMA_CFG.CADDR` bit is cleared, the DMA loads the `DMA_ADDRSTART` register on the first access of the work unit. When the `DMA_CFG.CADDR` bit is set, the DMA uses the `DMA_ADDR_CUR` register value for the starting address for the work unit and writes the same value to the `DMA_ADDRSTART` register. This operation permits continuation of a previous work unit. When DMA uses this mode, the fetched start address value (as part of the descriptor set at the end of a descriptor list or array) is ignored.	
		0	Load Starting Address
		1	Use Current Address

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SYNC	Synchronize Work Unit Transitions. Setting the `DMA_CFG.SYNC` bit clears the DMA FIFO and pointers before starting the first work unit of a work unit chain. When the transfer direction is memory read/transmit (`DMA_CFG.WNR` =0), the DMA waits until all data transmits to a peripheral before moving on to the next work unit, clearing the FIFO and pointers. When the transfer direction is memory write/receive (`DMA_CFG.WNR` =1), the DMA ignores the `DMA_CFG.SYNC` bit value after processing the first work unit of a work unit chain. As a channel can receive data when turned on but idle, data from the peripheral can still be in the FIFO even though the channel was not programmed. When the `DMA_CFG.SYNC` bit field is set at the beginning of a work unit chain (during the first work unit), the DMA clears the FIFO, erasing the data put into the FIFO while the channel was idle. Syncing lets programs change the `DMA_CFG.PSIZE` between individual work units and (in some cases) work unit chains. The sync resets the pointers in the FIFO, preventing misaligned FIFO access. Programs can change the `DMA_CFG.MSIZE` field between consecutive work units, independent of the `DMA_CFG.SYNC` bit setting. Syncing also permits changes to transfer direction. Because the data in the FIFO is eliminated, the data that went into the FIFO from one direction (transmit or receive) is not sent back in the other direction after the direction change.	
		0	No Synchronization
		1	Synchronize Channel
1 (R/NW)	WNR	Write/Read Channel Direction. The `DMA_CFG.WNR` selects receive (write to memory) or transmit (read from memory) channel direction.	
		0	Transmit (Read from memory)
		1	Receive (Write to memory)

Table 38-21: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Channel Enable. The `DMA_CFG.EN` bit enables the selected DMA channel. When a peripheral DMA channel is enabled, data requests from the peripheral denote DMA requests. When a channel is disabled, the DMA unit ignores the peripheral data request and passes it directly to the system event controller. To avoid unexpected results, enable the DMA channel before enabling the peripheral, and disable the peripheral before disabling the DMA channel. A transition of the `DMA_CFG.EN` bit from 0 to 1 creates a hard reset of all internal counters and states, including the `DMA_STAT` register. (All other register values remain unaffected.) A transition from 1 to 0 maintains all counters and registers for reading and analysis. Note that if a descriptor loads when this bit is cleared (see the `DMA_CFG.FLOW` field), the DMA transitions to the off or idle state after the descriptor load is complete.	
		0	Disable
		1	Enable

Current Descriptor Pointer Register

The DMA_DSCPTR_CUR register contains the memory address for the next descriptor to be loaded. The DMA_DSCPTR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel. For DMA_CFG.FLOW mode settings that involve descriptor fetches, this register is used to read descriptors into appropriate MMRs before a work unit begins. For descriptor list mode, the DMA_DSCPTR_CUR register is initialized from the DMA_DSCPTR_NXT register before fetching each descriptor set. Then, the address in the DMA_DSCPTR_CUR register increments as each descriptor is read.

When the entire descriptor set has been read, the DMA_DSCPTR_CUR register contains this value:

DMA_DSCPTR_CUR = Descriptor Start Address + Descriptor Size (# of elements)

For descriptor array mode, the DMA_DSCPTR_CUR register, and not the DMA_DSCPTR_NXT register, must be programmed by a MMR access before starting DMA operation.

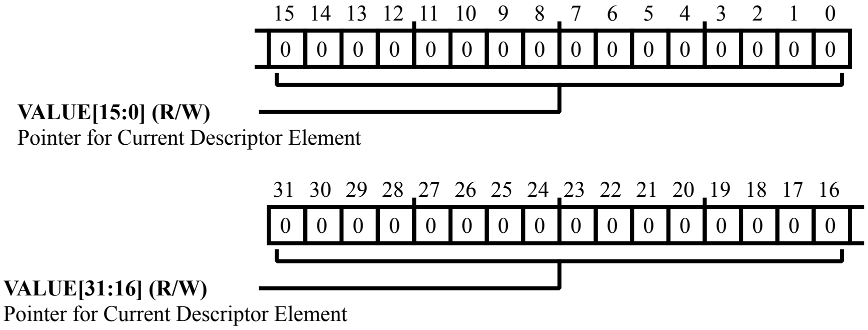

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Pointer for Current Descriptor Element
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Pointer for Current Descriptor Element

Figure 38-12: DMA_DSCPTR_CUR Register Diagram

Table 38-22: DMA_DSCPTR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer for Current Descriptor Element. The DMA_DSCPTR_CUR.VALUE bit field contains the memory address for the next descriptor to be loaded.

Pointer to Next Initial Descriptor Register

The DMA_DSCPTR_NXT register specifies the start location of the next descriptor set, which begins when the DMA activity specified by the current descriptor set completes. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

The DMA_DSCPTR_NXT register is only used in descriptor list mode. At the start of a descriptor fetch in this mode, the DMA_DSCPTR_NXT register is copied into the DMA_DSCPTR_CUR register. During descriptor fetch, the DMA increments the DMA_DSCPTR_CUR register value after reading each element of the descriptor set.

In descriptor list mode, the DMA_DSCPTR_NXT register (not the DMA_DSCPTR_CUR register) must be programmed directly through MMR access, before the DMA operation is started. In descriptor array mode, the DMA disregards the DMA_DSCPTR_NXT register and uses the DMA_DSCPTR_CUR register to control descriptor fetch.

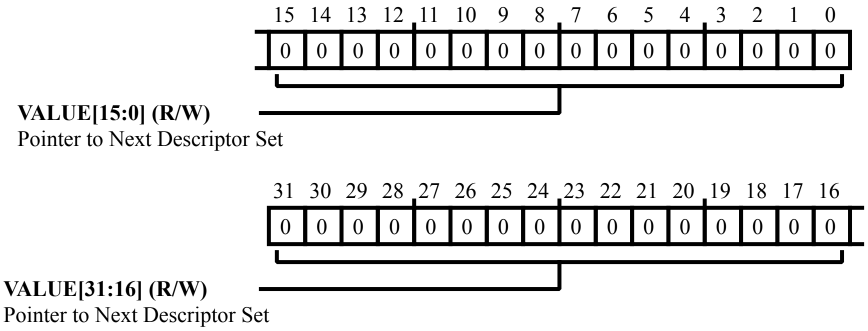

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Pointer to Next Descriptor Set
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Pointer to Next Descriptor Set

Figure 38-13: DMA_DSCPTR_NXT Register Diagram

Table 38-23: DMA_DSCPTR_NXT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer to Next Descriptor Set. The DMA_DSCPTR_NXT.VALUE bit field specifies the start location of the next descriptor set.

Previous Initial Descriptor Pointer Register

The `DMA_DSCPTR_PRV` register contains the initial descriptor pointer for the previous work unit. If `DMA_CFG.DESCIDCPY`is set, the DMA copies the initial descriptor pointer to `DMA_DSCPTR_PRV` after the work unit completes. Otherwise, the value is not updated.

To indicate an overrun, bit 0 of the `DMA_DSCPTR_PRV` register is used as a previous descriptor pointer overrun (PDPO) status bit. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error occurs when used for descriptor fetches. As a result, bit 1 and 0 of the `DMA_DSCPTR_PRV` register can be used for status. For more information, see the section on descriptor pointer capture in the DMA functional description.

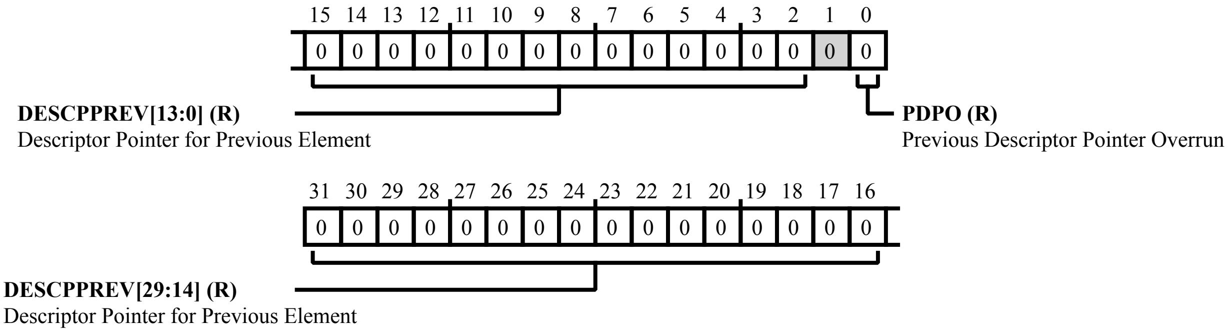

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DESCPPREV[13:0] (R)
Descriptor Pointer for Previous Element
PDPO (R)
Previous Descriptor Pointer Overrun
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DESCPPREV[29:14] (R)
Descriptor Pointer for Previous Element

Figure 38-14: DMA_DSCPTR_PRV Register Diagram

Table 38-24: DMA_DSCPTR_PRV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:2 (R/NW)	DESCPPREV	Descriptor Pointer for Previous Element. The `DMA_DSCPTR_PRV.DESCPPREV` bit field contains the initial descriptor pointer for the previous work unit.	
0 (R/NW)	PDPO	Previous Descriptor Pointer Overrun. The `DMA_DSCPTR_PRV.PDPO` bit signifies an alignment error. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error would occur when used for descriptor fetches.	
		0	No Error Occurred
		1	Error Occurred

Status Register

The `DMA_STAT` register indicates the status of DMA work units, the FIFO, errors, interrupts, and triggers.

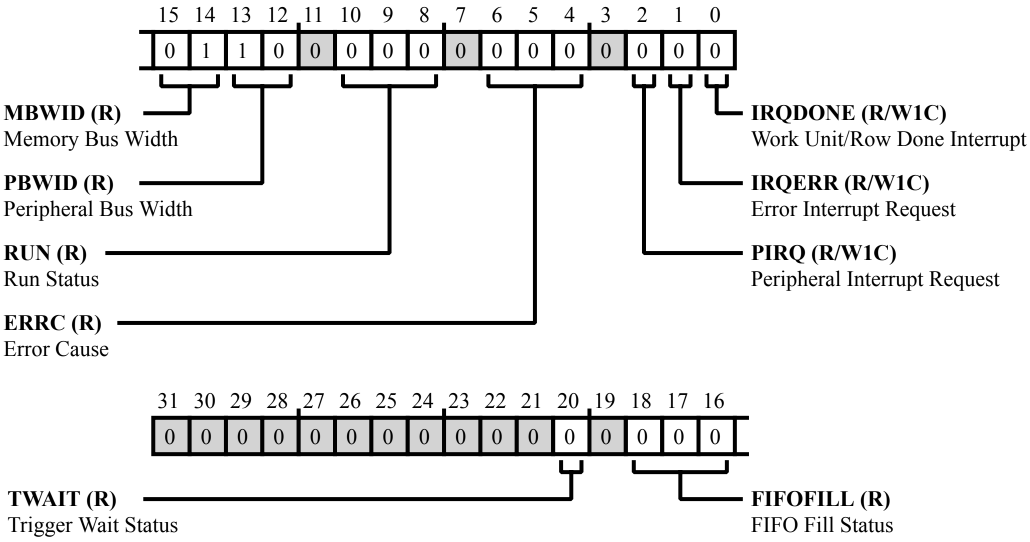

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
MBWID (R)
Memory Bus Width
PBWID (R)
Peripheral Bus Width
RUN (R)
Run Status
ERRC (R)
Error Cause
IRQDONE (R/W1C)
Work Unit/Row Done Interrupt
IRQERR (R/W1C)
Error Interrupt Request
PIRQ (R/W1C)
Peripheral Interrupt Request
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TWAIT (R)
Trigger Wait Status
FIFOFILL (R)
FIFO Fill Status

Figure 38-15: DMA_STAT Register Diagram

Table 38-25: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	TWAIT	Trigger Wait Status. The `DMA_STAT.TWAIT` bit indicates whether the DMA has or has not received a trigger. This bit is set until it reaches the next wait state. At that point, the bit is cleared, the DMA stops processing that work unit, and the following work unit processes. The DMA does not distinguish between one or more triggers received.	
		0	No Trigger Received
		1	Trigger Received
18:16 (R/NW)	FIFOFILL	FIFO Fill Status. The `DMA_STAT.FIFOFILL` bit field reports the quantity of data in the FIFO relative to available space.	
		0	Empty
		1	Empty < FIFO = 1/4 Full
		2	1/4 Full < FIFO = 1/2 Full
		3	1/2 Full < FIFO = 3/4 Full
		4	3/4 Full < FIFO = Full
		5	Reserved

Table 38-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	Reserved
		7	Full
15:14 (R/NW)	MBWID	Memory Bus Width. The `DMA_STAT.MBWID` bit field indicates the width of the memory bus connected to this DMA.	
		0	2 Bytes
		1	4 Bytes
		2	8 Bytes
		3	16 Bytes
13:12 (R/NW)	PBWID	Peripheral Bus Width. The `DMA_STAT.PBWID` bit field indicates the width of the peripheral bus connected to this DMA.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
10:8 (R/NW)	RUN	Run Status. The `DMA_STAT.RUN` bit field reports the DMA's current operational state. If the DMA is in idle or stop state, the `DMA_CFG.EN` bit is either 0 or 1. This bit field does not clear when a transition of the `DMA_CFG.EN` bit from 0 to 1 occurs. The `DMA_STAT.RUN` clears automatically when the DMA completes.	
		0	Idle/Stop State
		1	Descriptor Fetch
		2	Data Transfer
		3	Waiting for Trigger
		4	Waiting for Write ACK/FIFO Drain to Peripheral
		5	Reserved
		6	Reserved
		7	Reserved
6:4 (R/NW)	ERRC	Error Cause. When an interrupt request error occurs (`DMA_STAT.IRQERR`), the DMA updates the `DMA_STAT.ERRC` bit field to identify the type of error. For more information, see the errors section of the DMA functional description.	

Table 38-25: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Configuration Error
		1	Illegal Write Occurred While Channel Running
		2	Address Alignment Error
		3	Memory Access or Fabric Error
		4	Reserved
		5	Trigger Overrun
		6	Bandwidth Monitor Error
		7	Reserved
2 (R/W1C)	PIRQ	Peripheral Interrupt Request. The `DMA_STAT.PIRQ` bit indicates a peripheral interrupt request. Programs can use the `DMA_STAT.PIRQ` bit status to help determine which DMA asserted the interrupt request. This bit also helps to distinguish between an interrupt request caused by the state of the work unit and an interrupt request caused by the peripheral.	
		0	No Interrupt request
		1	Interrupt Request signaled by peripheral
1 (R/W1C)	IRQERR	Error Interrupt Request. The `DMA_STAT.IRQERR` bit indicates that the DMA has detected a documented rule violation during DMA programming or operation. The DMA cannot, however, flag all possible programming or operation issues to indicate errors. Programmers can use the `DMA_STAT.IRQERR` bit to help determine which DMA issued the error interrupt request. The `DMA_STAT.IRQERR` does not clear a transition of the `DMA_CFG.EN` bit from 0 to 1. Clear the `DMA_STAT.IRQERR` bit with a write-1-to-clear operation prior to the `DMA_CFG.EN` transition for the fields to be reset.	
		0	No Error
		1	Error Occurred
0 (R/W1C)	IRQDONE	Work Unit/Row Done Interrupt. The `DMA_STAT.IRQDONE` bit indicates that the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt request. Programs can use the `DMA_STAT.IRQDONE` status to help determine which DMA asserted the interrupt request. Programs can also use these bits to help distinguish between an interrupt request based on the state of the work unit and an interrupt request made by the peripheral. For more information, see the interrupts section of the DMA functional description.	
		0	Inactive
		1	Active

Inner Loop Count Start Value Register

For 2D DMA, the `DMA_XCNT` register contains the inner loop count. This value selects the number of `DMA_CFG.MSIZE` size data transfers that make up the length of a row. For 1D DMA, the `DMA_XCNT` register specifies the number of `DMA_CFG.MSIZE` size data transfers for the entire work unit. The `DMA_XCNT` register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

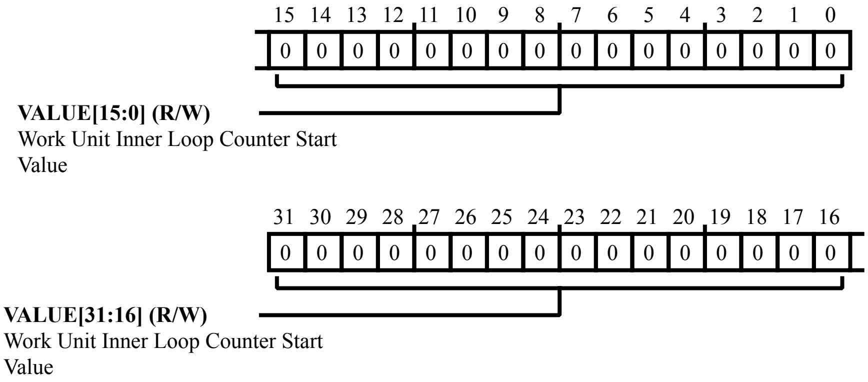

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Work Unit Inner Loop Counter Start Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Work Unit Inner Loop Counter Start Value

Figure 38-16: DMA_XCNT Register Diagram

Table 38-26: DMA_XCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Start Value. For 2D DMA, the `DMA_XCNT.VALUE` bit field contains the inner loop count. For 1D DMA, `DMA_XCNT.VALUE` specifies the number of `DMA_CFG.MSIZE` size data transfers for the entire work unit.

Current Count (1D) or Intra-row XCNT (2D) Register

For 1D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register at the beginning of each work unit. For 2D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register after the end of each row. The DMA decrements the value in DMA_XCNT_CUR register each time a DMA_CFG.MSIZE size data transfer occurs. When the count in DMA_XCNT_CUR register expires, the work unit is complete. In 2D DMA, the DMA_XCNT_CUR value is 0 only when the entire transfer is complete.

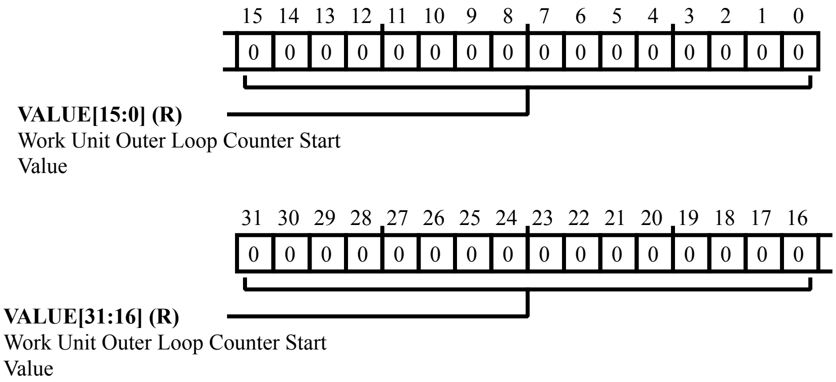

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Work Unit Outer Loop Counter Start Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Work Unit Outer Loop Counter Start Value

Figure 38-17: DMA_XCNT_CUR Register Diagram

Table 38-27: DMA_XCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Start Value. For 1D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value at the beginning of each work unit. For 2D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value after the end of each row.

Inner Loop Address Increment Register

The `DMA_XMOD` register contains a signed, two's-complement byte address increment. In 1D DMA, this increment is the stride that is applied after each `DMA_CFG.MSIZE` size data transfer. The `DMA_XMOD` register is read/write prior to enabling the channel, but is read-only after enabling the channel.

The `DMA_XMOD` register value is specified in bytes, regardless of the work unit size. In 2D DMA, this increment is applied after each `DMA_CFG.MSIZE` size data transfer in the inner loop, up to but not including the last `DMA_CFG.MSIZE` size data transfer in each inner loop. After the last `DMA_CFG.MSIZE` size data transfer in each inner loop, the `DMA_YMOD` register is applied instead, including the last `DMA_CFG.MSIZE` size data transfer of a work unit.

The `DMA_XMOD` field can be set to 0. In this case, DMA is performed repeatedly to or from the same address. This approach can be useful for transferring data between a data register and an external memory-mapped peripheral.

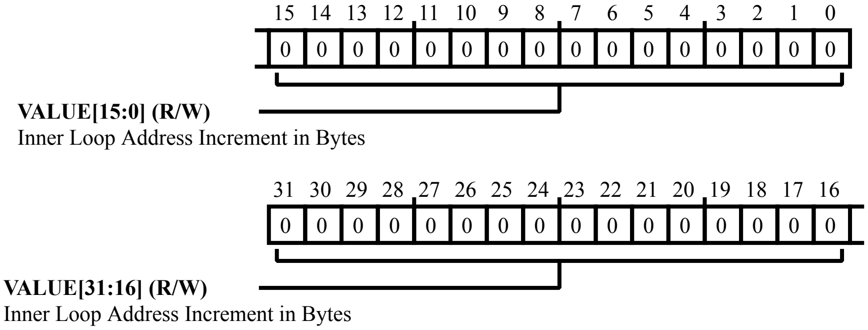

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Inner Loop Address Increment in Bytes
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Inner Loop Address Increment in Bytes

Figure 38-18: DMA_XMOD Register Diagram

Table 38-28: DMA_XMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Loop Address Increment in Bytes. The `DMA_XMOD.VALUE` bit field contains a signed, two's-complement byte address increment.

Outer Loop Count Start Value (2D only) Register

For 2D DMA, the `DMA_YCNT` register contains the outer loop count. This register is not used in 1D DMA mode. The `DMA_YCNT` register specifies the number of rows in the outer loop of a 2D DMA sequence. The `DMA_YCNT` register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

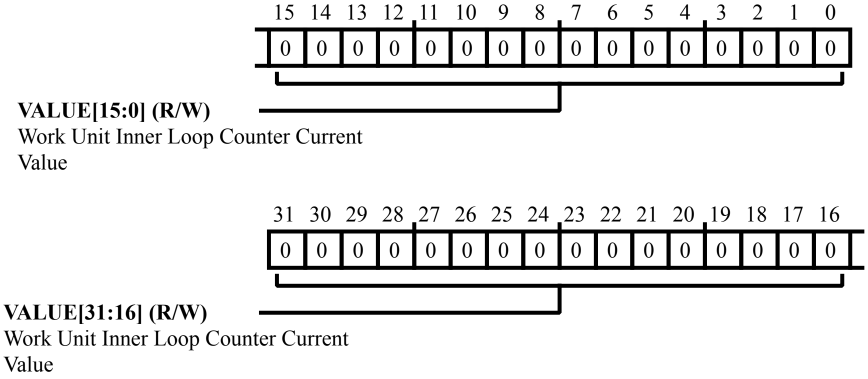

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Work Unit Inner Loop Counter Current Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Work Unit Inner Loop Counter Current Value

Figure 38-19: DMA_YCNT Register Diagram

Table 38-29: DMA_YCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Current Value. For 2D DMA, the `DMA_YCNT.VALUE` bit field contains the outer loop count.

Current Row Count (2D only) Register

For 2D DMA, the DMA loads the `DMA_YCNT_CUR` register from the `DMA_YCNT` register at the beginning of each 2D DMA session. The `DMA_YCNT_CUR` register is not used for 1D DMA. The DMA decrements the `DMA_YCNT_CUR` register each time the `DMA_XCNT_CUR` register expires during 2D DMA operation, signifying the completion of an entire row transfer.

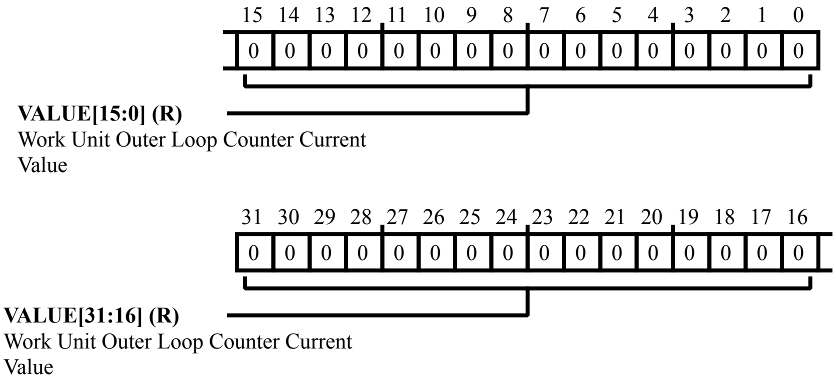

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Work Unit Outer Loop Counter Current Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Work Unit Outer Loop Counter Current Value

Figure 38-20: DMA_YCNT_CUR Register Diagram

Table 38-30: DMA_YCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Current Value. For 2D DMA, the `DMA_YCNT_CUR.VALUE` bit field holds the value from the `DMA_YCNT` register at the beginning of each 2D DMA session.

Outer Loop Address Increment (2D only) Register

The DMA_YMOD register contains a signed, two's-complement value. This byte address increment is applied after each decrement of the DMA_YCNT_CUR register. The value is the offset between the last word of one row and the first word of the next row. Note that DMA_YMOD is specified in bytes, regardless of the work unit size. The DMA_YMOD register is read/write prior to enabling the channel, but is read-only after enabling the channel.

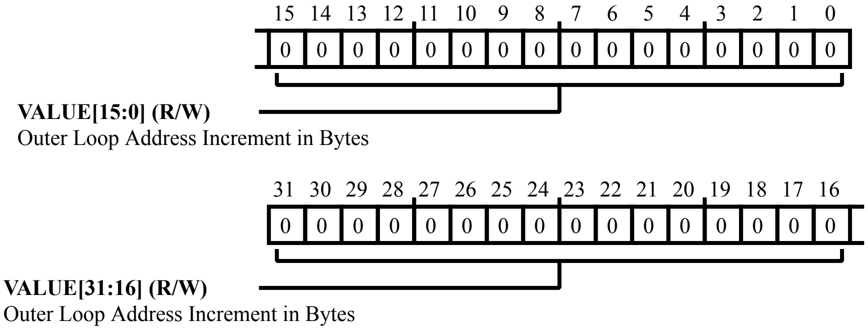

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Outer Loop Address Increment in Bytes
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Outer Loop Address Increment in Bytes

Figure 38-21: DMA_YMOD Register Diagram

Table 38-31: DMA_YMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Outer Loop Address Increment in Bytes. The `DMA_YMOD.VALUE` bit field contains a signed, two's-complement value.

39 Extended Memory DMA (EMDMA)

The Extended Memory DMA engine can be used in applications that copy data in a non-sequential manner. This includes delay lines, scatter and gather, and circular access types.

NOTE: Unlike previous SHARC processors which contained external port DMA, the current EMDMA module can access all memory locations (L1/L2/L3) for source and destination DMA operations.

EMDMA Features

EMDMA frees the processor core, allowing it to perform other operations while the data transfers between memories occurs as a background task.

EMDMA has the following features and capabilities.

- Standard mode DMA Transfer
- Chained mode with direction on-the-fly
- Tap list mode (scatter/gather)
- Delay line mode (write to read)
- All the DMA modes can operate in circular fashion
- In circular operation, some modes allow a write back of the index pointer for correct addressing of the next Transfer Control Block (TCB)

EMDMA Functional Description

The following sections provide information on the functional operations and operating modes of EMDMA.

Internal-to-internal DMA

This is accomplished by indexing all external parameter registers with internal addresses.

DMA bursting

DMA supports burst transfers only when the modifier is 1, for any other modifier values, single accesses are performed. The burst size is not user configurable and is chosen by the processor for optimal performance. The maximum burst size is 8 and the minimum is 1.

Transfer control blocks

The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously re-run the same DMA.

Chain pointer DMA

In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete.

ADSP-SC58x EMDMA Register List

The EMDMA controllers support a variety of direct memory access operations which can access any system memory and transfer the entire block of data. A set of registers govern EMDMA operations. For more information on EMDMA functionality, see the EMDMA register descriptions.

Table 39-1: ADSP-SC58x EMDMA Register List

Name	Description
`EMDMA_BASE`	External Base Address Register
`EMDMA_BUFLEN`	Circular Buffer Length Register
`EMDMA_CHNPTR`	Chain Pointer Register
`EMDMA_CNT0`	Internal Count Register
`EMDMA_CNT1`	External Count Register
`EMDMA_CTL`	External Memory DMA Control Register
`EMDMA_INDX0`	Internal Index Register
`EMDMA_INDX1`	External Index Register
`EMDMA_MOD0`	Internal Modifier Register
`EMDMA_MOD1`	External Modifier Register
`EMDMA_TCNT`	Delay Line Tap Count Register
`EMDMA_TPTR`	Tap List Pointer Register

ADSP-SC58x EMDMA Interrupt List

Table 39-2: ADSP-SC58x EMDMA Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
170	EMDMA0_DONE	EMDMA0 Transfer Done	Edge	
171	EMDMA1_DONE	EMDMA1 Transfer Done	Edge	

ADSP-SC58x EMDMA Trigger List

Table 39-3: ADSP-SC58x EMDMA Trigger List Masters

Trigger ID	Name	Description	Sensitivity
82	`EMDMA0_DONE`	EMDMA0 DMA Done	Edge
83	`EMDMA1_DONE`	EMDMA1 DMA Done	Edge

Table 39-4: ADSP-SC58x EMDMA Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

DMA Addressing

Besides the traditional internal-to-external addressing type, the DMA module also supports internal-to-internal transfers. This is accomplished by indexing all external parameter registers with internal addresses. The DMA controller recognizes the transfer by addresses and not by an additional control bit setting.

All the DMA addresses given by EMDMA are word-aligned byte addresses. The programming for the index registers is provided in the *Register Descriptions* section.

NOTE: Similar to EPDMA on the previous generation of SHARC processors, EMDMA on the ADSP-SC58x processors also has both internal and external sets of DMA parameter registers. The register sets on the new ADSP-SC58x processors can point to either internal or external memory and are referred to as channel 0 and channel 1 in this manual. These channels should not be considered as two EPDMA channels (which is equivalent to two EMDMA module instances) of the previous SHARC processors.

DMA Burst Transfers

DMA supports burst transfers only when the modifier is 1. For any other modifier values, single accesses are performed.

The burst size is not user-configurable and is chosen by the processor for optimal performance. The maximum burst size is 8 and the minimum is 1.

The EMDMA uses appropriate burst transfer sizes for optimal throughput. For example, if the word count is 15, then 5 bursts are performed with burst sizes of 8+4+1+1+1 transfers.

Transfer Control Block (TCB) Memory Storage

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the processor's internal memory. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Assignment

The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously rerun the same DMA. The EMDMA reads each word of the TCB and loads it into the corresponding register. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

The address field of the chain pointer registers is only 30 bits wide. If a program writes a symbolic address to bit 30 of the chain pointer there may be a conflict with the `EMDMA_CHNPTR.PCI` bit. Programs should clear the upper bits of the address then AND the `EMDMA_CHNPTR.PCI` bit separately, if needed.

Starting Chain Loading

A DMA sequence is defined as the sum of the DMA transfer from when the parameter registers initialize to when the count register decrements to zero. The EMDMA module has a chaining enable bit (`EMDMA_CTL.CHEN`).

To start the chain, write the internal (channel 0) index address of the first TCB to the chain pointer register (`EMDMA_CHNPTR`). When chaining is enabled, DMA transfers are initiated by writing a memory address to the chain pointer register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

During TCB chain loading, the EMDMA loads the DMA channel parameter registers with values retrieved from system memory.

The address in the chain pointer register points to the highest address of the TCB. This contains the internal (channel 0) index parameter. This means that if a program declares an array to hold the TCB, the chain pointer register should point to the last location of the array and not to the first TCB location.

Buffered Chain Loading Register

The chain pointer register (`EMDMA_CHNPTR`) is buffered. Before the chain loading starts, the buffer is copied into the chain pointer register and is decremented after each register is loaded.

The chain pointer register can be loaded with a new address at any time during the DMA sequence (`EMDMA_CTL.CHEN` =1). This allows a DMA channel to have chaining status deactivated (chain pointer register = 0x0) until some event occurs that loads the chain pointer register with a non-zero value. Writing all zeros to the address field of the chain pointer register also deactivates chaining for the next TCB.

TCB Storage

The EMDMA supports many different types of DMA, resulting in different lengths of TCBs. The TCB size varies from six locations (chained DMA) to 13 locations (delay line DMA). The ***EMDMA TCBs*** table shows the required TCBs for chained DMA.

In the following tables, TCB refers to the start address of the TCB array.

Table 39-5: EMDMA TCBs

Address	Register
TCB	`EMDMA_CHNPTR`
TCB + 0x1	`EMDMA_MOD1`
TCB + 0x2	`EMDMA_INDX1`
TCB + 0x3	`EMDMA_CNT0`
TCB + 0x4	`EMDMA_MOD0`
TCB + 0x5	`EMDMA_INDX0`

The order the descriptors are fetched with circular buffering enabled is shown in the ***EMDMA TCBs for Circular DMA*** table.

Table 39-6: EMDMA TCBs for Circular DMA

Address	Register
TCB	`EMDMA_CHNPTR`
TCB + 0x1	`EMDMA_BUFLEN`
TCB + 0x2	`EMDMA_BASE`
TCB + 0x3	`EMDMA_MOD1`
TCB + 0x4	`EMDMA_INDX1`
TCB + 0x5	`EMDMA_CNT0`
TCB + 0x6	`EMDMA_MOD0`
TCB + 0x7	`EMDMA_INDX0`

For delay line DMA, TCB loading is split into two sequences to improve overall priority. The first TCB loads the write parameters (`EMDMA_INDX0` – `EMDMA_BUFLEN`) and the second loads the read parameters (`EMDMA_INDX0` – `EMDMA_CHNPTR`). This two stage loading is transparent to the application. The order the descriptors are fetched with circular buffering enabled is shown in the ***EMDMA TCBs for Scatter/Gather DMA*** table.

Table 39-7: EMDMA TCBs for Scatter/Gather DMA

Address	Register
Delay Line Read	

Table 39-7: EMDMA TCBs for Scatter/Gather DMA (Continued)

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_TPTR
TCB + 0x2	EMDMA_TCNT
TCB + 0x3	EMDMA_MOD1
TCB + 0x4	EMDMA_CNT0
TCB + 0x5	EMDMA_INDX0
Delay Line Write	
TCB + 0x6	EMDMA_BUFLEN
TCB + 0x7	EMDMA_BASE
TCB + 0x8	EMDMA_MOD1
TCB + 0x9	EMDMA_INDX1
TCB + 0xA	EMDMA_CNT0
TCB + 0xB	EMDMA_MOD0
TCB + 0xC	EMDMA_INDX0

The order the descriptors are fetched for scatter/gather DMA with circular buffering enabled is shown in the *EMDMA TCBs for Scatter/Gather DMA* table and the *EMDMA TCBs for Circular Scatter/Gather DMA Address Register* table.

Table 39-8: EMDMA TCBs for Scatter/Gather DMA

Address	Register
TCB	EMDMA_CHNPTR
TCB + 0x1	EMDMA_TPTR
TCB + 0x2	EMDMA_TCNT
TCB + 0x3	EMDMA_MOD1
TCB + 0x4	EMDMA_INDX1
TCB + 0x5	EMDMA_CNT0
TCB + 0x6	EMDMA_MOD0
TCB + 0x7	EMDMA_INDX0

Table 39-9: EMDMA TCBs for Circular Scatter/Gather DMA Address Register

Address	Register
TCB	EMDMA_CHNPTR

Table 39-9: EMDMA TCBs for Circular Scatter/Gather DMA Address Register (Continued)

Address	Register
TCB + 0x1	`EMDMA_BUFLEN`
TCB + 0x2	`EMDMA_BASE`
TCB + 0x3	`EMDMA_TPTR`
TCB + 0x4	`EMDMA_TCNT`
TCB + 0x5	`EMDMA_MOD1`
TCB + 0x6	`EMDMA_INDX1`
TCB + 0x7	`EMDMA_CNT0`
TCB + 0x8	`EMDMA_MOD0`
TCB + 0x9	`EMDMA_INDX0`

EMDMA Operating Modes

This section and the *EMDMA_CTL Register Bit to Operating Modes* table show the different DMA modes which can be used. The complete register bit descriptions are in the *External Memory DMA Control Registers (EMDMA_CTL)*.

Table 39-10: EMDMA_CTL Register Bit to Operating Modes

<table>
<tr><th>Bit (Name)</th><th>Standard</th><th>Chained</th><th>Scatter/Gather</th><th>Delay Line</th></tr>
<tr><td colspan="5">Control Bits</td></tr>
<tr><td>0 (EN)</td><td colspan="4">Valid</td></tr>
<tr><td>1 (TRAN)</td><td colspan="3">Valid</td><td>N/A</td></tr>
<tr><td>2 (CHEN)</td><td colspan="4">Valid</td></tr>
<tr><td>3 (DLEN)</td><td colspan="3">N/A</td><td>Valid</td></tr>
<tr><td>4 (CBEN)</td><td colspan="4">Valid</td></tr>
<tr><td>5 (DFLSH)</td><td colspan="4">Valid</td></tr>
<tr><td>6</td><td colspan="4">N/A</td></tr>
<tr><td>7 (WRBEN)</td><td>N/A</td><td>Valid</td><td>(=0)</td><td>(=1)</td></tr>
<tr><td>8 (OFCEN)</td><td colspan="3">Valid</td><td>N/A</td></tr>
<tr><td>9 (TLEN)</td><td colspan="2">N/A</td><td>Valid</td><td>N/A</td></tr>
<tr><td>11–10</td><td colspan="4">N/A</td></tr>
<tr><td>12 (INTDONE0)</td><td colspan="4">Valid</td></tr>
<tr><td>15–13</td><td colspan="4">N/A</td></tr>
<tr><td colspan="5">Status Bits</td></tr>
</table>

Table 39-10: EMDMA_CTL Register Bit to Operating Modes (Continued)

Bit (Name)	Standard	Chained	Scatter/Gather	Delay Line
17–16 (DFS)	Valid			
19–18	N/A			
20 (DMAS0)	Valid			
21 (CHS)	N/A	Valid	N/A	Valid
22 (TLS)	N/A		Valid	N/A
23 (WBS)	N/A			Valid
24 (DMAS1)	Valid			
25 (DIRS)	Valid			
31–26	N/A			

NOTE: Reading additional bit-field information from N/A (Not applicable) bits does not generate a meaningful result.

A program sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers.

NOTE: The `EMDMA_CNT1` parameter register (read only) is a copy of the `EMDMA_CNT0` register. If `EMDMA_CNT0` is written, the `EMDMA_CNT1` register is updated automatically.

Standard DMA

A standard DMA (once it is configured) transfers data from location A to location B. An interrupt can be used to indicate the end of the transfer. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit (`EMDMA_CTL.EN`), write new parameters to the index, modify, and count registers (parameter registers), then set the DMA enable bit to re-enable DMA.

This DMA type resembles the traditional DMA type to initialize the different internal and external parameters (channel0 and channel1) (index, modify and count) registers and configuration of the DMA control registers.

Circular Buffered DMA

Circular buffered DMA resembles the traditional core DAG circular buffered mode by using registers for circular buffering. In this mode, the DMA needs two additional registers (base and length) to support reads and writes to a circular buffer. The *Circular Buffering Write DMA* and *Circular Buffering Read DMA* figures illustrate circular buffered DMA, in contrast with the *Standard Write DMA* figure.

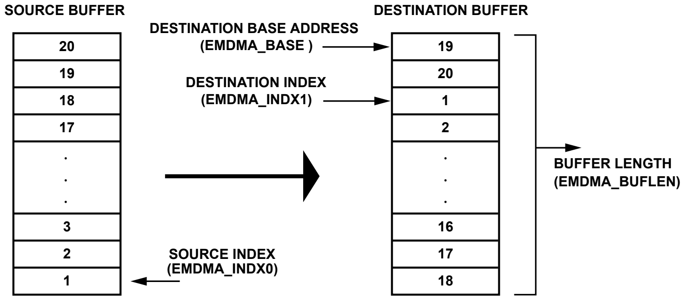

SOURCE BUFFER
20
19
18
17
3
2
1
DESTINATION BASE ADDRESS
(EMDMA_BASE)
DESTINATION INDEX
(EMDMA_INDX1)
SOURCE INDEX
(EMDMA_INDX0)
DESTINATION BUFFER
19
20
1
2
16
17
18
BUFFER LENGTH
(EMDMA_BUFLEN)

Figure 39-1: Circular Buffering Write DMA

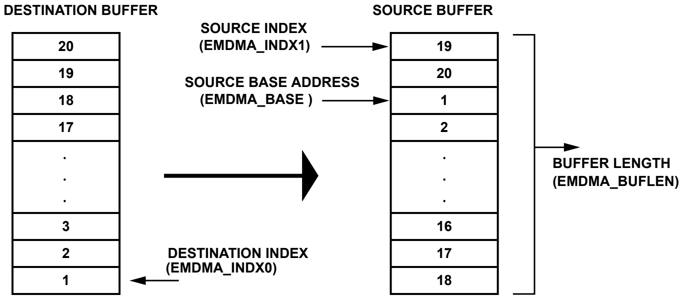

DESTINATION BUFFER
20
19
18
17
3
2
1
SOURCE INDEX
(EMDMA_INDX1)
SOURCE BASE ADDRESS
(EMDMA_BASE)
DESTINATION INDEX
(EMDMA_INDX0)
SOURCE BUFFER
19
20
1
2
16
17
18
BUFFER LENGTH
(EMDMA_BUFLEN)

Figure 39-2: Circular Buffering Read DMA

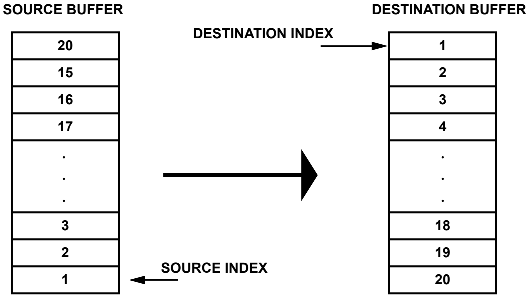

SOURCE BUFFER
20
15
16
17
3
2
1
DESTINATION INDEX
SOURCE INDEX
DESTINATION BUFFER
1
2
3
4
18
19
20

Figure 39-3: Standard Write DMA

NOTE: Circular buffering is available for all operating modes (standard, chained, tap list and delay line DMA).

Chained DMA Mode

Chained DMA sequences are a set of multiple DMA operations, each auto-initializing the next in line. It is used to support automated access by a linked list (repetitive reads and writes to a defined location defined by the individual TCBs). To start a new DMA sequence after the current one is finished, the EMDMA automatically loads new index, modify, and count values pointed to by that channel's `EMDMA_CHNPTR` register. Using chaining, programs can set up consecutive DMA operations and each operation can have different attributes.

DMA data transfers can be set up as continuous or periodic. With chained DMA, the attributes of a specific DMA are stored in internal memory and are referred to as a Transfer Control Block or TCB. Extended Memory DMA loads these attributes in chains for execution. This allows for multiple chains that are finite or infinite.

NOTE: When chaining is enabled, polling should not be used to determine DMA status only because the DMA appears inactive if it is sampled while the next TCB is loading. In such cases where chaining is enabled, along with the polling of DMA status bit, polling of the chaining status bit should also occur to so that the correct status of the DMA is known. For example, the `EMDMA_CTL.CHS` bit should be polled as well as the `EMDMA_CTL.DMAS0` and `EMDMA_CTL.DMAS1` bits when EMDMA is configured in DMA chaining mode.

Data Direction On-the-Fly

A change of external memory data direction for each individual TCB in a chain sequence is allowed.

The `EMDMA_CHNPTR.CPDR` bit changes the data flow direction. If the `EMDMA_CHNPTR.CPDR` =0, writes through channel 0 are performed; if `EMDMA_CHNPTR.CPDR` =1, channel 0 reads are performed. This works similarly to the `EMDMA_CHNPTR.PCI` bit. The `EMDMA_CTL.OFCEN` and `EMDMA_CTL.CHEN` bits must be set (=1) to enable this functionality.

NOTE: If chaining is enabled with the `EMDMA_CTL.OFCEN` =1, then the `EMDMA_CTL.TRAN` bit has no effect, and the data direction is determined by the `EMDMA_CHNPTR.CPDR` bit.

Write Back Circular Index Pointer

Operating the DMA in circular mode requires some special considerations. The index pointer of start address within the buffer may wrap around for the case when IC × IM > EL or it does not finish if IC × IM < EL where:

IC is the value of the `EMDMA_INDX0` register

IM is the value of the `EMDMA_MOD0` register

EL is the value of the `EMDMA_BUFLEN` register

In both cases, the TCB start address is no longer valid.

Setting the `EMDMA_CTL.WRBEN` bit writes (at the end of current TCB block) the current index address + 1 into the TCB memory which is the start address for the next TCB. This bit is only selectable for chained DMA mode. For tap list and delay line modes, this bit is hardwired to 0 or 1.

Scatter/Gather DMA

The purpose of scatter/gather DMA is the transfer of data from/to non-contiguous memory blocks.

The scatter/gather DMA type is a fixed block size scatter/gather DMA that relies on tap list entries in system memory to calculate the (Channel 1 DMA Address) to scatter/gather the DMA. If the DMA direction is Channel 1 write (`EMDMA_CTL.TRAN` =1) then it is a scatter DMA. If `EMDMA_CTL.TRAN` =0 then it is a gather DMA. This mode also supports chained and circular buffer chained DMAs.

See the ***Scatter DMA (Writes)***, ***Gather DMA (Reads)***, ***Circular Buffering Scatter DMA (Writes)***, and ***Circular Gather DMA (Reads)*** figures.

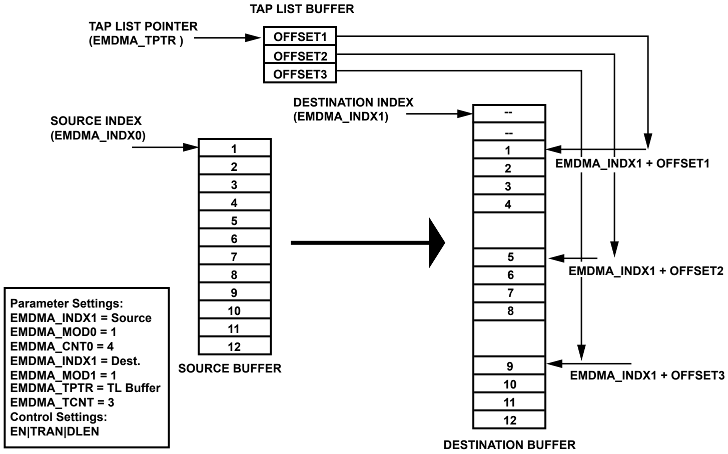

TAP LIST BUFFER
TAP LIST POINTER
(EMDMA_TPTR)
OFFSET1
OFFSET2
OFFSET3
DESTINATION INDEX
(EMDMA_INDX1)
SOURCE INDEX
(EMDMA_INDX0)
EMDMA_INDX1 + OFFSET1
EMDMA_INDX1 + OFFSET2
EMDMA_INDX1 + OFFSET3
SOURCE BUFFER
DESTINATION BUFFER
Parameter Settings:
EMDMA_INDX1 = Source
EMDMA_MOD0 = 1
EMDMA_CNT0 = 4
EMDMA_INDX1 = Dest.
EMDMA_MOD1 = 1
EMDMA_TPTR = TL Buffer
EMDMA_TCNT = 3
Control Settings:
EN|TRAN|DLEN

Figure 39-4: Scatter DMA (Writes)

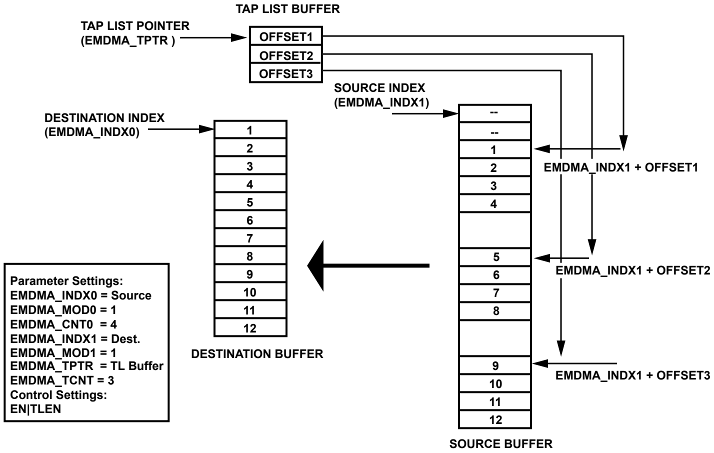

TAP LIST BUFFER
TAP LIST POINTER
(EMDMA_TPTR)
OFFSET1
OFFSET2
OFFSET3
SOURCE INDEX
(EMDMA_INDX1)
DESTINATION INDEX
(EMDMA_INDX0)
EMDMA_INDX1 + OFFSET1
EMDMA_INDX1 + OFFSET2
EMDMA_INDX1 + OFFSET3
DESTINATION BUFFER
SOURCE BUFFER
Parameter Settings:
EMDMA_INDX0 = Source
EMDMA_MOD0 = 1
EMDMA_CNT0 = 4
EMDMA_INDX1 = Dest.
EMDMA_MOD1 = 1
EMDMA_TPTR = TL Buffer
EMDMA_TCNT = 3
Control Settings:
EN|TLEN

Figure 39-5: Gather DMA (Reads)

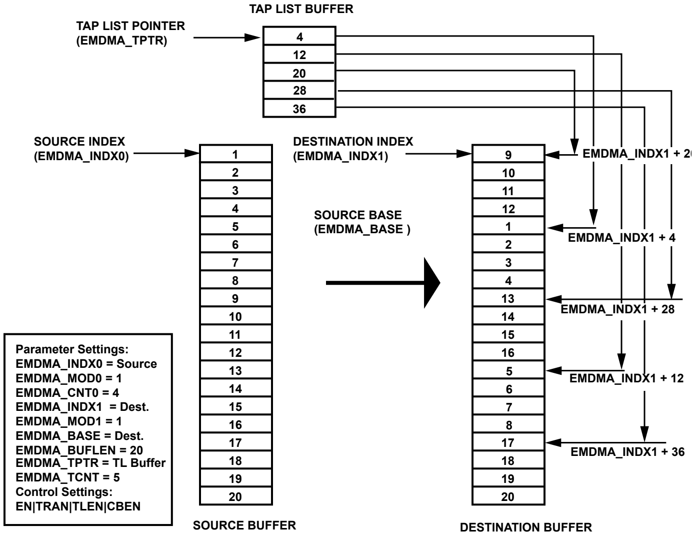

TAP LIST BUFFER
TAP LIST POINTER
(EMDMA_TPTR)
4
12
20
28
36
SOURCE INDEX
(EMDMA_INDX0)
DESTINATION INDEX
(EMDMA_INDX1)
SOURCE BASE
(EMDMA_BASE)
EMDMA_INDX1 + 2
EMDMA_INDX1 + 4
EMDMA_INDX1 + 28
EMDMA_INDX1 + 12
EMDMA_INDX1 + 36
SOURCE BUFFER
DESTINATION BUFFER
Parameter Settings:
EMDMA_INDX0 = Source
EMDMA_MOD0 = 1
EMDMA_CNT0 = 4
EMDMA_INDX1 = Dest.
EMDMA_MOD1 = 1
EMDMA_BASE = Dest.
EMDMA_BUFLEN = 20
EMDMA_TPTR = TL Buffer
EMDMA_TCNT = 5
Control Settings:
EN|TRAN|TLEN|CBEN

Figure 39-6: Circular Buffering Scatter DMA (Writes)

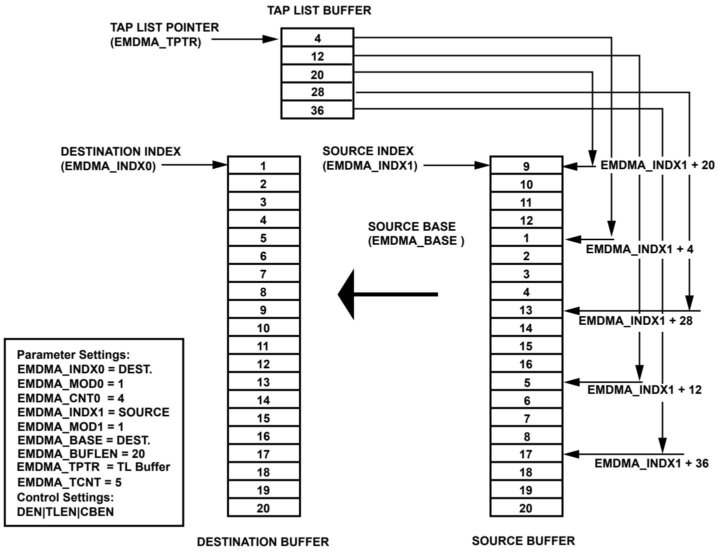

TAP LIST BUFFER
TAP LIST POINTER
(EMDMA_TPTR)
4
12
20
28
36
DESTINATION INDEX
(EMDMA_INDX0)
SOURCE INDEX
(EMDMA_INDX1)
SOURCE BASE
(EMDMA_BASE)
EMDMA_INDX1 + 20
EMDMA_INDX1 + 4
EMDMA_INDX1 + 28
EMDMA_INDX1 + 12
EMDMA_INDX1 + 36
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
9
10
11
12
1
2
3
4
13
14
15
16
5
6
7
8
17
18
19
20
Parameter Settings:
EMDMA_INDX0 = DEST.
EMDMA_MOD0 = 1
EMDMA_CNT0 = 4
EMDMA_INDX1 = SOURCE
EMDMA_MOD1 = 1
EMDMA_BASE = DEST.
EMDMA_BUFLEN = 20
EMDMA_TPTR = TL Buffer
EMDMA_TCNT = 5
Control Settings:
DEN|TLEN|CBEN
DESTINATION BUFFER
SOURCE BUFFER

Figure 39-7: Circular Gather DMA (Reads)

For each 32-bit tap read, the Channel 1 read index is shown in the ***Read/Write Index Pre-Modify (Scatter/Gather DMA)*** table. Note that one tap list entry starts multiple reads.

Table 39-11: Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation	Result	
EMDMA_INDX1 + EMDMA_TPTR [EMDMA_TCNT] + (EMDMA_MOD1 × EMDMA_CNT0)	Block Size	Tap
EMDMA_INDX1 + EMDMA_TPTR[0] + EMDMA_MOD1 × 1	N	0
EMDMA_INDX1 + EMDMA_TPTR[0] + EMDMA_MOD1 × 2		
EMDMA_INDX1 + EMDMA_TPTR[0] + EMDMA_MOD1 × 3		
EMDMA_INDX1 + EMDMA_TPTR[0] + EMDMA_MOD1 × N		
EMDMA_INDX1 + EMDMA_TPTR[1] + EMDMA_MOD1 × 1	N	1
EMDMA_INDX1 + EMDMA_TPTR[1] + EMDMA_MOD1 × 2		
EMDMA_INDX1 + EMDMA_TPTR[1] + EMDMA_MOD1 × 3		
EMDMA_INDX1 + EMDMA_TPTR[1] + EMDMA_MOD1 × N		
EMDMA_INDX1 + EMDMA_TPTR[M] + EMDMA_MOD1 × 1	N	M
EMDMA_INDX1 + EMDMA_TPTR[M] + EMDMA_MOD1 × 2		
EMDMA_INDX1 + EMDMA_TPTR[M] + EMDMA_MOD1 × 3		
EMDMA_INDX1 + EMDMA_TPTR[M] + EMDMA_MOD1 × N		

Pre Modified Read/Write Index

For scatter/gather DMA, the tap list modifiers are employed and the number of taps is determined by the tap list count register (EMDMA_TCNT). The number of sequential reads (block size) from every tap is determined by the internal count register (`EMDMA_CNT0`), and is the same for every tap. The read/write pointer in external index register (`EMDMA_INDX1`) serves as the index address for these read/writes.

`TL[N]` is the first tap list entry in the memory as pointed to by the tap list pointer register (`EMDMA_TPTR`). The tap list entries are 27-bit signed integers. For each read/write block, the DMA state machine fetches the offset from the tap list. The offset is added to the `EMDMA_INDX1` register value to get the start address of the next block. The Channel 1 addresses are circular buffered if circular buffering is enabled (see the ***Circular Buffering Scatter DMA (Writes)*** and ***Circular Buffering Gather DMA (Reads)*** figures in Scatter/Gather DMA).

Once the `EMDMA_CNT0` register for the final tap decrements to zero (both `EMDMA_TCNT` and `EMDMA_CNT0` are zero), then the tap list DMA access is complete and the DMA completion interrupt is generated (if chaining is enabled the interrupt depends on the `EMDMA_CHNPTR.PCI` bit setting).

The write back mode (`EMDMA_CTL.WRBEN` bit) is hardwired to zero for tap list based DMA (as the addressing is pre-modify, and therefore the `EMDMA_INDX1` value coincides with the TCB value even at the end of the DMA).

Delay Line DMA

Delay line DMA is used to support reads and writes to delay line buffers with limited core interaction. In this sense, delay line DMA is a quantity of integrated writes followed by reads from channel 1 (a delay line DMA access). Delay line DMA is described in the following sections.

NOTE: Delay line DMA can only operate by using chained DMA mode (`EMDMA_CTL.CHEN` bit set). In order to use delay line DMA for a single DMA sequence, initialize the `EMDMA_CHNPTR` register to zero in the TCB.

NOTE: Delay line DMA can be used in any system memory.

The delay line DMA operation follows these steps:

1. Load the first half of TCB for writing (seven parameters).
2. DMA writes to the delay line buffer until IC = 0.
3. Update the EI index pointer if circular mode is enabled.
4. Load the second half of TCB for reading (six parameters).
5. DMA tap based reads from the delay line buffer until RC = 0.

   Jump to step 1.

Writes to delay line. The number of writes is determined by the `EMDMA_CNT0` register. The data is fetched from the `EMDMA_INDX0` register and the `EMDMA_MOD0` register is used as the internal modifier. The `EMDMA_INDX1` register serves as the external index and is incremented by the `EMDMA_MOD1` register after each write. These writes are circular buffered if circular buffering is enabled. See the ***Write to Delay Line Buffer*** figure.

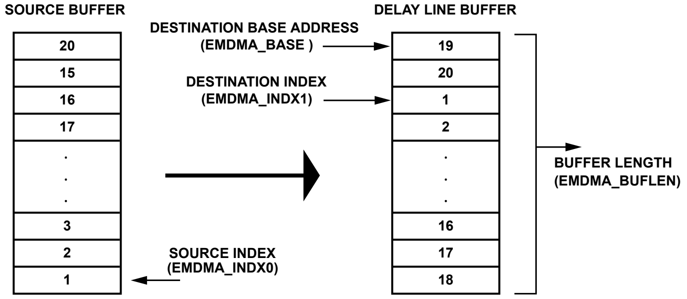

SOURCE BUFFER
20
15
16
17
3
2
1
DESTINATION BASE ADDRESS (EMDMA_BASE)
DESTINATION INDEX (EMDMA_INDX1)
SOURCE INDEX (EMDMA_INDX0)
DELAY LINE BUFFER
19
20
1
2
16
17
18
BUFFER LENGTH (EMDMA_BUFLEN)

Figure 39-8: Write to Delay Line Buffer

When the writes are complete, (`EMDMA_CNT0` =0) the `EMDMA_INDX1` register, which serves as the write pointer of the delay line, is written back (`EMDMA_CTL.WRBEN` is hardwired to 1) to the TCB location from where it was fetched.

Reads from the delay line. For reads, the tap list (TL) modifiers are used and the number of reads is determined by the `EMDMA_CNT0` register. The write pointer in the `EMDMA_INDX0` register serves as the index address for these reads (reads start from where writes end). The `EMDMA_INDX0` register, along with tap list modifiers, are used in a pre-modify addressing mode to create the external address for the reads. Therefore, for each read, the DMA controller fetches the external modifier (`EMDMA_TCNT` register) from the tap list and the reads are circular buffered (if enabled). See the ***Read From Delay Line Buffer*** figure.

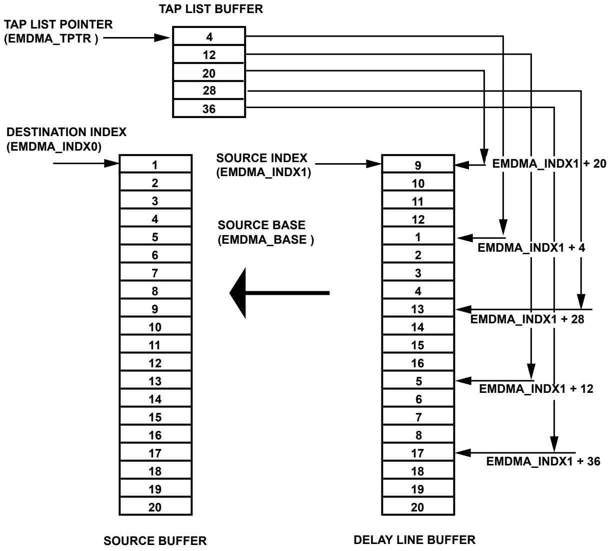

TAP LIST POINTER (EMDMA_TPTR)
TAP LIST BUFFER
4
12
20
28
36
DESTINATION INDEX (EMDMA_INDX0)
SOURCE INDEX (EMDMA_INDX1)
SOURCE BASE (EMDMA_BASE)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
9 10 11 12 1 2 3 4 13 14 15 16 5 6 7 8 17 18 19 20
EMDMA_INDX1 + 20
EMDMA_INDX1 + 4
EMDMA_INDX1 + 28
EMDMA_INDX1 + 12
EMDMA_INDX1 + 36
SOURCE BUFFER
DELAY LINE BUFFER

Figure 39-9: Read From Delay Line Buffer

Pre-Modified Read Index

Note that TL[N] is the first tap list entry in memory pointed to by the tap list pointer register (`EMDMA_TPTR`). Tap list entries are 27-bit signed integers. Therefore, for each read-block, the DMA state machine fetches the offset external modifier from the tap list. The reads are circular buffered if circular buffering is enabled.

NOTE: The channel 1 DMA address generation follows pre-modify addressing for reads in delay line DMA and therefore the `EMDMA_INDX1` register values are not updated. Also the `EMDMA_MOD1` register does not have any effect during these delay line reads. Once the read count completes, the EMDMA_CNT0 register decrements to zero (both EMDMA_CNT0 and EMDMA_TCNT are zero) for the final tap. Finally, the delay line DMA access completes and the DMA completion interrupt is generated. If chaining is enabled, the interrupt is dependent on the EMDMA_CHNPTR.PCI bit setting. The delay line DMA can only be initialized using the TCB. In order to use the delay line DMA for a single DMA sequence, initialize the EMDMA_CHNPTR register to zero in the TCB.

For each 32-bit tap read, the channel 1 read index is shown in the ***Read/Write Index Pre-Modify (Scatter/Gather DMA)*** table. Note that one tap list entry starts multiple reads.

Table 39-12: Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation **EMDMA_INDX0** + **EMDMA_TPTR**[**EMDMA_TCNT**] + (**EMDMA_MOD1** × **EMDMA_CNT0**)	Result	
	Block Size	Tap
EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 × 1	N	0
EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 × 2		
EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 × 3		
EMDMA_INDX0 + EMDMA_TPTR[0] + EMDMA_MOD1 × N		
EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 × 1	N	1
EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 × 2		
EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 × 3		
EMDMA_INDX0 + EMDMA_TPTR[1] + EMDMA_MOD1 × N		
EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 × 1	N	M
EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 × 2		
EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 × 3		
EMDMA_INDX0 + EMDMA_TPTR[M] + EMDMA_MOD1 × N		

ADSP-SC58x EMDMA Register Descriptions

Extended Memory DMA (EMDMA) contains the following registers.

Table 39-13: ADSP-SC58x EMDMA Register List

Name	Description
EMDMA_BASE	External Base Address Register
EMDMA_BUFLEN	Circular Buffer Length Register
EMDMA_CHNPTR	Chain Pointer Register
EMDMA_CNT0	Internal Count Register
EMDMA_CNT1	External Count Register

Table 39-13: ADSP-SC58x EMDMA Register List (Continued)

Name	Description
EMDMA_CTL	External Memory DMA Control Register
EMDMA_INDX0	Internal Index Register
EMDMA_INDX1	External Index Register
EMDMA_MOD0	Internal Modifier Register
EMDMA_MOD1	External Modifier Register
EMDMA_TCNT	Delay Line Tap Count Register
EMDMA_TPTR	Tap List Pointer Register

External Base Address Register

The `EMDMA_BASE` register contains the external base address of the Delay Line buffer. This is used for maintaining circular buffered read/writes to the delay line.

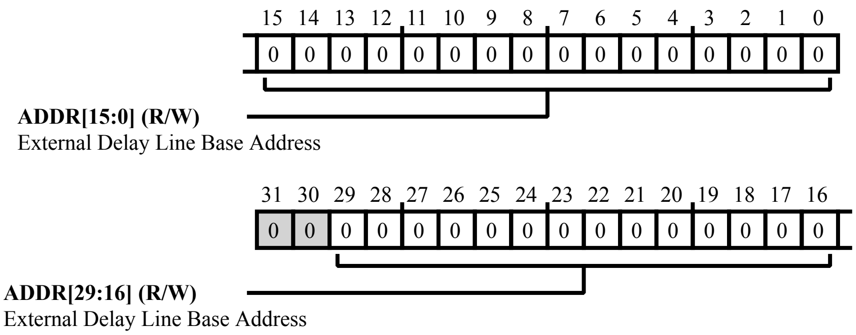

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R/W)
External Delay Line Base Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[29:16] (R/W)
External Delay Line Base Address

Figure 39-10: EMDMA_BASE Register Diagram

Table 39-14: EMDMA_BASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	ADDR	External Delay Line Base Address. The `EMDMA_BASE.ADDR` bit field contains the external base address of the Delay Line buffer.

Circular Buffer Length Register

The `EMDMA_BUFLEN` register holds the circular buffer length for the Delay-line DMA.

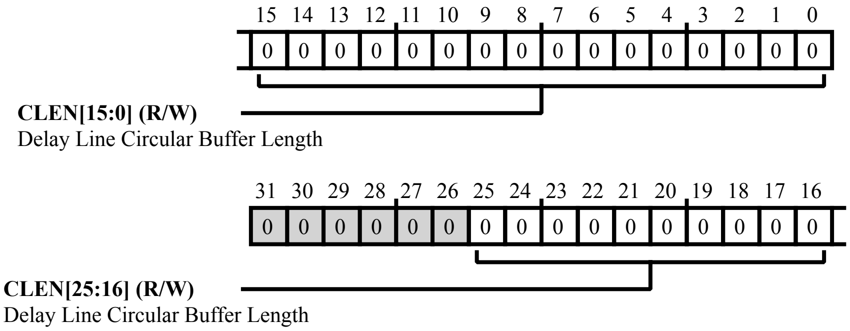

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLEN[15:0] (R/W)
Delay Line Circular Buffer Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CLEN[25:16] (R/W)
Delay Line Circular Buffer Length

Figure 39-11: EMDMA_BUFLEN Register Diagram

Table 39-15: EMDMA_BUFLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:0 (R/W)	CLEN	Delay Line Circular Buffer Length. The `EMDMA_BUFLEN.CLEN` bit field holds the circular buffer length for the Delay-line DMA.

Chain Pointer Register

The `EMDMA_CHNPTR` register contains the address of the next descriptor in memory when the `EMDMA_CTL.CHEN` bit =1. This register also has bits to change DMA directions for each descriptor and the PCI bit. Note that the lower 30-bits of this register are to be written with 30 MSB's of the Word Aligned Byte addresses of the corresponding next descriptor address.

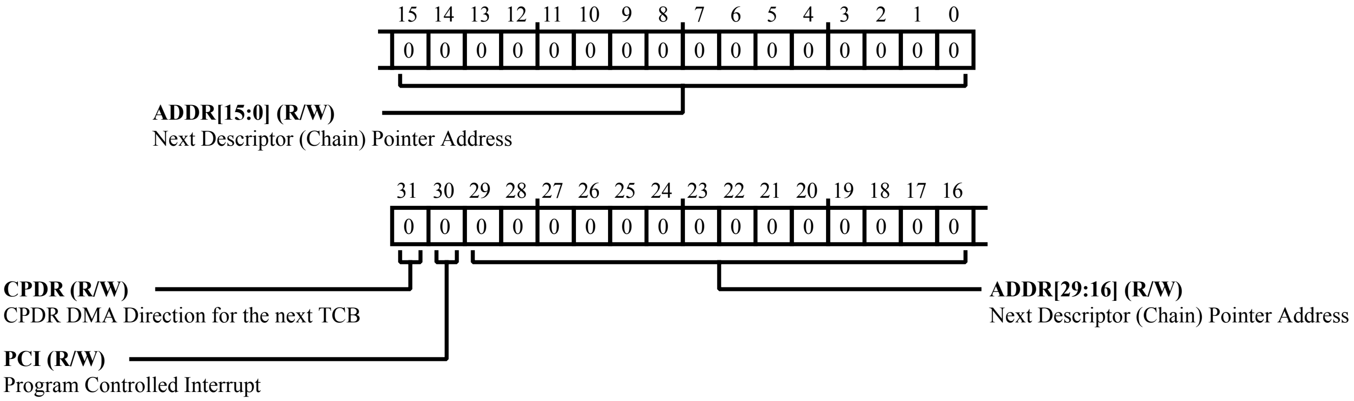

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R/W)
Next Descriptor (Chain) Pointer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPDR (R/W)
CPDR DMA Direction for the next TCB
PCI (R/W)
Program Controlled Interrupt
ADDR[29:16] (R/W)
Next Descriptor (Chain) Pointer Address

Figure 39-12: EMDMA_CHNPTR Register Diagram

Table 39-16: EMDMA_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CPDR	CPDR DMA Direction for the next TCB. The `EMDMA_CHNPTR.CPDR` bit configures whether the DMA is a write to internal memory or a read from internal memory. Note that this setting is applicable only if `EMDMA_CTL.OFCEN` =1 and is not applicable for Delay Line DMA.	
		0	Write to Channel 0 (Channel 1 reads)
		1	Read from Channel 0 (Channel 1 Writes)
30 (R/W)	PCI	Program Controlled Interrupt. The `EMDMA_CHNPTR.PCI` bit PCI sets whether an interrupt is generated after the current TCB or if no interrupt is generated. (Only affects DMA if chaining is enabled)	
		0	Enable DMA Channel interrupt to occur at the completion of the entire DMA chained transfer
		1	Enable DMA Channel interrupt to occur at the completion of current DMA sequence
29:0 (R/W)	ADDR	Next Descriptor (Chain) Pointer Address. The `EMDMA_CHNPTR.ADDR` bit field provides the address of the next descriptor in memory.	

Internal Count Register

The `EMDMA_CNT0` register contains the number of words to be transferred for channel 0 DMA. Note: If Delay Line DMA is enabled then the `EMDMA_CNT0` register serves as the count register for the Delay Line writes.

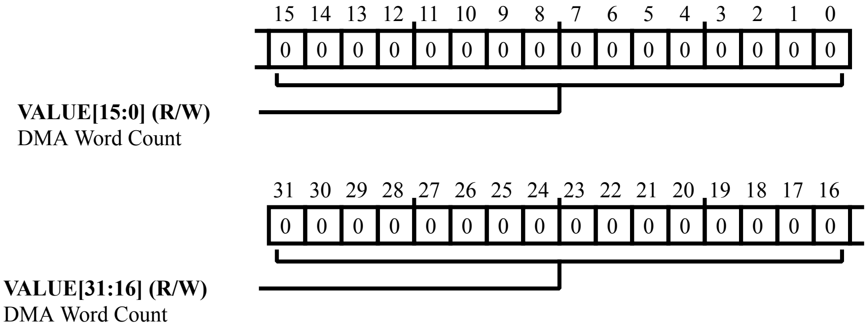

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA Word Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
DMA Word Count

Figure 39-13: EMDMA_CNT0 Register Diagram

Table 39-17: EMDMA_CNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Word Count. The `EMDMA_CNT0.VALUE` bit field is the DMA word count.

External Count Register

The `EMDMA_CNT1` register contains the number of words to be transferred for channel 1 DMA. Note: If Delay Line DMA is enabled then the `EMDMA_CNT1` register serves as the count register for the Delay Line writes.

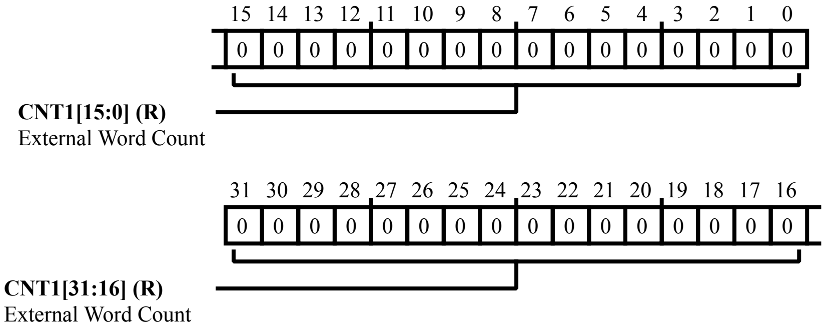

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT1[15:0] (R)
External Word Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT1[31:16] (R)
External Word Count

Figure 39-14: EMDMA_CNT1 Register Diagram

Table 39-18: EMDMA_CNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT1	External Word Count. The `EMDMA_CNT1.CNT1` bit field is the external word count.

External Memory DMA Control Register

The EMDMA_CTL register contains bits that enable and configure EMDMA and indicate DMA transfer status.

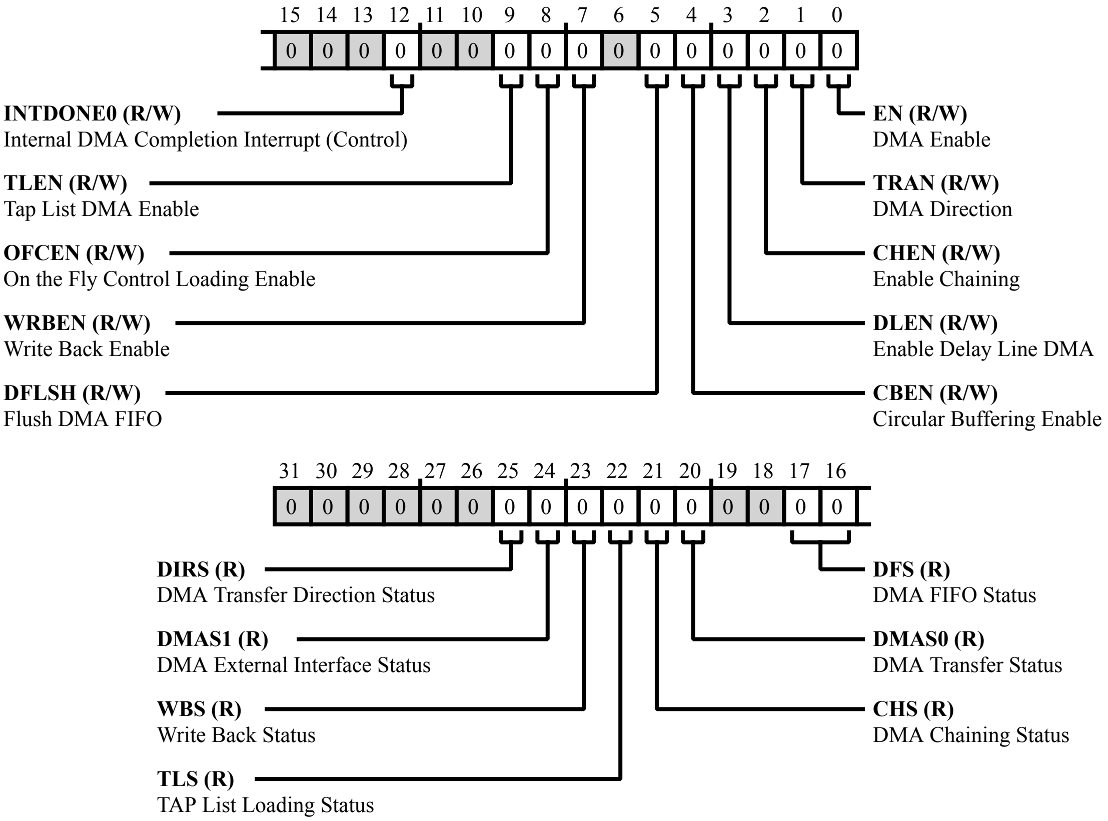

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INTDONE0 (R/W)
Internal DMA Completion Interrupt (Control)
TLEN (R/W)
Tap List DMA Enable
OFCEN (R/W)
On the Fly Control Loading Enable
WRBEN (R/W)
Write Back Enable
DFLSH (R/W)
Flush DMA FIFO
EN (R/W)
DMA Enable
TRAN (R/W)
DMA Direction
CHEN (R/W)
Enable Chaining
DLEN (R/W)
Enable Delay Line DMA
CBEN (R/W)
Circular Buffering Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIRS (R)
DMA Transfer Direction Status
DMAS1 (R)
DMA External Interface Status
WBS (R)
Write Back Status
TLS (R)
TAP List Loading Status
DFS (R)
DMA FIFO Status
DMAS0 (R)
DMA Transfer Status
CHS (R)
DMA Chaining Status

Figure 39-15: EMDMA_CTL Register Diagram

Table 39-19: EMDMA_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/NW)	DIRS	DMA Transfer Direction Status. The `EMDMA_CTL.DIRS` bit provides the DMA transfer status direction. This is useful for delay line DMA where the transfer direction changes with the state of the DMA state machine. For standard DMA, the `EMDMA_CTL.DIRS` bit reflects the state of the `EMDMA_CTL.TRAN` bit.	
		0	DMA direction is Channel 1 Reads
		1	DMA direction is Channel 1 Writes

Table 39-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/NW)	DMAS1	DMA External Interface Status. The `EMDMA_CTL.DMAS1` bit provides the DMA channel 1 transfer status.	
		0	Channel 1 DMA does not have any access pending
		1	Channel 1 DMA access are pending
23 (R/NW)	WBS	Write Back Status. The `EMDMA_CTL.WBS` bit provides the delay line write pointer write back status.	
		0	Write pointer write back is not active
		1	Write pointer write back is active
22 (R/NW)	TLS	TAP List Loading Status. The `EMDMA_CTL.TLS` bit provides the DMA tap list loading status.	
		0	TAP list loading is not active
		1	TAP list loading is active
21 (R/NW)	CHS	DMA Chaining Status. The `EMDMA_CTL.CHS` bit provides the DMA chaining status.	
		0	DMA chain loading is not active
		1	DMA chain loading is active
20 (R/NW)	DMAS0	DMA Transfer Status. The `EMDMA_CTL.DMAS0` bit provides the DMA channel 0 transfer status.	
		0	DMA idle
		1	DMA in progress
17:16 (R/NW)	DFS	DMA FIFO Status. The `EMDMA_CTL.DFS` bit field provides the DMA FIFO status.	
		0	FIFO EMPTY
		1	FIFO Partially Full
		2	Reserved
		3	FIFO Full

Table 39-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	INTDONE0	Internal DMA Completion Interrupt (Control). The `EMDMA_CTL.INTDONE0` bit configures when the DMA complete interrupt is generated. The `EMDMA_CTL.INTDONE0` =1 setting is provided for backward compatibility with older SHARC processors.	
		0	Interrupt on access completion (Channel 0 or Channel 1 DMA completion)
		1	Interrupt on Channel 0 DMA completion.
9 (R/W)	TLEN	Tap List DMA Enable. The `EMDMA_CTL.TLEN` bit enables scatter/gather tap list DMA.	
		0	Disables the tap list based scatter/gather DMA
		1	Enables the tap list based scatter/gather DMA
8 (R/W)	OFCEN	On the Fly Control Loading Enable. The control bits in `EMDMA_CHNPTR` register are used to describe the next TCB behavior if the `EMDMA_CTL.OFCEN` bit is set and therefore the DMA controls can be changed from TCB to TCB. 0 = disables the control bits in the `EMDMA_CHNPTR` register 1 = Enables the control bits in the `EMDMA_CHNPTR` register. If chaining is enabled with `EMDMA_CTL.OFCEN` bit set then the `EMDMA_CTL.TRAN` bit has no effect, and direction is determined by `EMDMA_CHNPTR.CPDR` bit.	
7 (R/W)	WRBEN	Write Back Enable. The `EMDMA_CTL.WRBEN` bit enables write back of the EIEP register after reads and or writes.Write back is automatically enabled for delay line DMA. WRBEN is applicable only if chaining is enabled (`EMDMA_CTL.CHEN` =1).	
5 (R/W)	DFLSH	Flush DMA FIFO. The `EMDMA_CTL.DFLSH` bit flushes the DMA FIFO. The buffer is only flushed if this bit is set. It can be set with the enable bit. Setting this bit also clears the `EMDMA_CTL.DFS` bit.	
4 (R/W)	CBEN	Circular Buffering Enable. The `EMDMA_CTL.CBEN` bit enables circular buffering. Circular buffering can be used with normal DMA as well, if circular buffering is enabled with chaining for normal DMA then ELEP and EBEP should be part of the TCB.	
		0	Disables circular buffering with delay line DMA
		1	Enables circular buffering with delay line DMA

Table 39-19: EMDMA_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DLEN	Enable Delay Line DMA. The `EMDMA_CTL.DLEN` bit enables delay line DMA. This bit is applicable only if the `EMDMA_CTL.CHEN` bit =1.	
		0	Delay-line DMA disabled
		1	Delay-line DMA enabled
2 (R/W)	CHEN	Enable Chaining. The `EMDMA_CTL.CHEN` bit enables DMA chaining.	
		0	Chaining disabled
		1	Chaining enabled
1 (R/W)	TRAN	DMA Direction. The `EMDMA_CTL.TRAN` bit determines the DMA data direction. Note: If delay line DMA is enabled then this bit does not have any effect. For delay line DMA, transfer direction depends on the state of delay line transfers. For Internal-Internal or External-External DMA this bit has to be set.	
		0	Write through Channel 0 (Channel 1 reads)
		1	Read from Channel 0(Channel 1 writes)
0 (R/W)	EN	DMA Enable. The `EMDMA_CTL.EN` bit enables DMA.	
		0	Disable DMA
		1	Enable DMA

Internal Index Register

The `EMDMA_INDX0` register contains the start address of the buffer for Channel 0 DMA. Note: For delay line DMA the `EMDMA_INDX0` register serves as the delay line write index which is the start address of the channel 0 DMA buffer for the channel 1 write data.

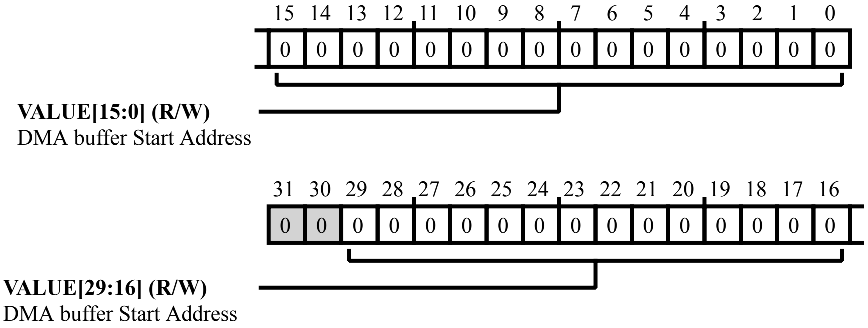

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA buffer Start Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
DMA buffer Start Address

Figure 39-16: EMDMA_INDX0 Register Diagram

Table 39-20: EMDMA_INDX0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	DMA buffer Start Address. The `EMDMA_INDX0.VALUE` bit field is written with the 30 MSBs of the Word Aligned Byte addresses.

External Index Register

The EMDMA_INDX1 register contains the start address of the buffer for channel 1 DMA.

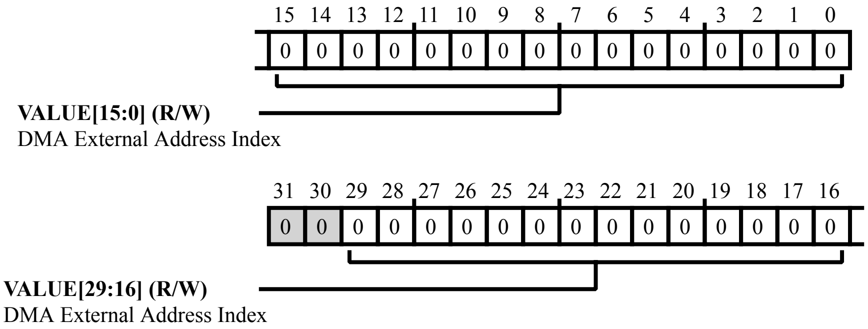

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA External Address Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
DMA External Address Index

Figure 39-17: EMDMA_INDX1 Register Diagram

Table 39-21: EMDMA_INDX1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	DMA External Address Index. The `EMDMA_INDX1.VALUE` bit field is the DMA external address index.

Internal Modifier Register

The `EMDMA_MOD0` register contains the channel 0 DMA address modifier.

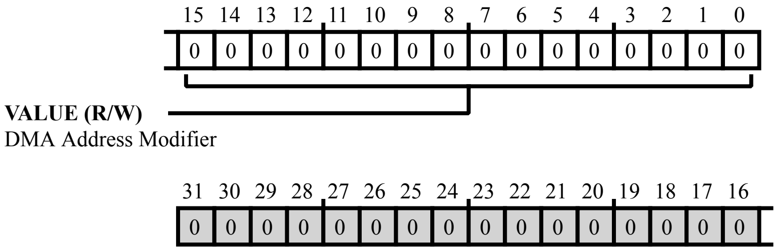

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
DMA Address Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 39-18: EMDMA_MOD0 Register Diagram

Table 39-22: EMDMA_MOD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	DMA Address Modifier. The `EMDMA_MOD0.VALUE` bit field is the DMA address modifier.

External Modifier Register

The `EMDMA_MOD1` register contains the external (channel 1 DMA) address modifier.

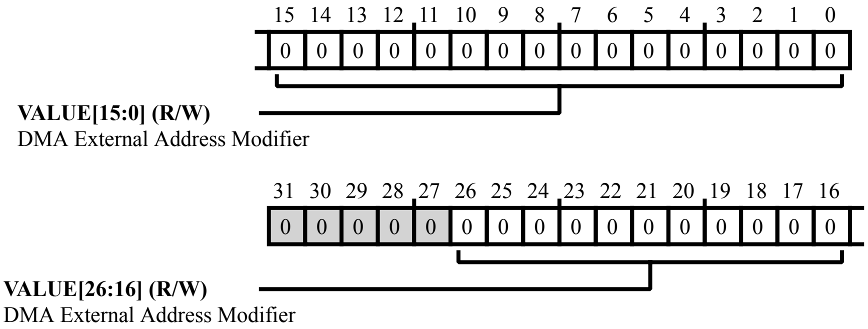

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
DMA External Address Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[26:16] (R/W)
DMA External Address Modifier

Figure 39-19: EMDMA_MOD1 Register Diagram

Table 39-23: EMDMA_MOD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26:0 (R/W)	VALUE	DMA External Address Modifier. The `EMDMA_MOD1.VALUE` bit field is the DMA external address modifier.

Delay Line Tap Count Register

The `EMDMA_TCNT` register is the tap count register for Delay Line DMA. This register holds the length of the tap list (the number of taps). The total number of words read from the delay line is equal to the `EMDMA_TCNT` (tap count) multiplied by the `EMDMA_CNT1` (read block size).

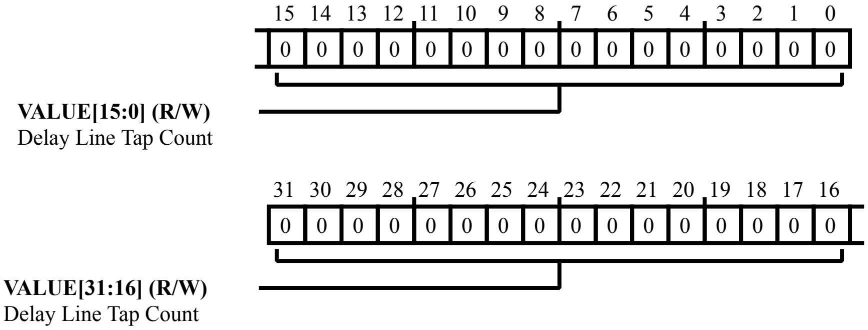

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Delay Line Tap Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Delay Line Tap Count

Figure 39-20: EMDMA_TCNT Register Diagram

Table 39-24: EMDMA_TCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Line Tap Count. The `EMDMA_TCNT.VALUE` bit field is the tap count register for Delay Line DMA.

Tap List Pointer Register

The `EMDMA_TPTR` register holds the address of an array in memory which holds offsets to be used when accessing a Delay-line in system memory. The offset represents the first address of each read block. Note that the lower 30-bits of this register are to be written with 30 MSBs of the Word Aligned Byte address of the Array.

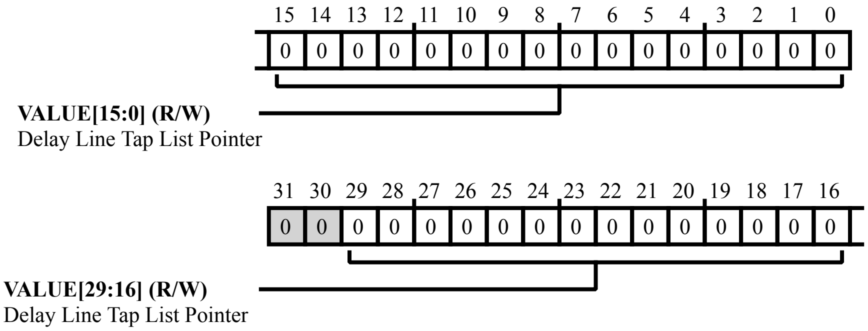

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Delay Line Tap List Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Delay Line Tap List Pointer

Figure 39-21: EMDMA_TPTR Register Diagram

Table 39-25: EMDMA_TPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Delay Line Tap List Pointer. The `EMDMA_TPTR.VALUE` bit field contains the offsets to be used when accessing a Delay-line in system memory.

40 Cyclic Redundancy Check (CRC)

The CRC peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error.

The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by any qualified master of the CRC peripheral bus.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the look-up table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

NOTE: CRC is supported by standard bandwidth MDMA only.

CRC Features

The CRC peripheral supports a number of key features.

- Memory scan modes for memory verification
- Memory transfer modes for on-the-fly CRC calculations while transferring data from one memory to another
- A programmable 32-bit CRC polynomial with automatic LUT generation
- Data mirroring options

The CRC module also includes the following features.

- CRC checksum computation and comparison modes
- 32-bit programmable CRC polynomial with bit reverse option
- Automatic look-up table (LUT) generation

- Data mirroring options for endian and reflected polynomial cases
- Automatic clear and preset of results
- Fault and error interrupt reporting
- DMA and MMR based operation

Because the CRC module is closely tied to memory-to-memory DMA (MDMA) channel pairs, the use cases include the following features.

- Memory scan mode with CRC compute or compare
- Memory transfer mode with CRC compute or compare
- Memory fill operation with 32-bit data patterns
- Memory verify operation
- MMR write access to FIFO of destination DMA
- MMR read access to FIFO of source DMA
- Profiting from advanced DMA features, like descriptor mode and bandwidth control or monitor

CRC Functional Description

The CRC peripheral supports a number of modes of operation that allow for the initialization and verification of regions of memory. The peripheral supports efficient memory-fill and verification operations on regions of memory with or against a constant value. These modes of operation do not require the CRC engine to calculate a signature. Other modes of operation allow for the calculation of CRC signature and verification for a memory region. The modes allow for on-the-fly CRC calculation when performing memory-to-memory DMA transfers from one memory region to another.

To minimize the need for core accesses, the peripheral interfaces with one or more (depending on processor features) memory-to memory DMA (MDMA) channels. This connectivity permits flexible configuration, in which data can be written-to or read-from the peripheral using DMA transactions, core transactions, or a combination of both.

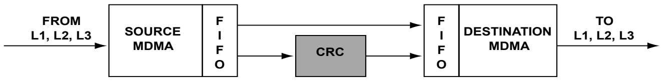


Figure 40-1: Memory Flow

ADSP-SC58x CRC Register List

The Cyclic Redundancy Check (CRC) unit includes the data comparison, polynomial operation, and look up table generation features needed for CRC operation. The CRC provides CRC protection as specified by many functional safety requirements. This unit facilitates the system software's ability to periodically check the correctness of the code/data available in memory. A set of registers govern CRC operations. For more information on CRC functionality, see the CRC register descriptions.

Table 40-1: ADSP-SC58x CRC Register List

Name	Description
`CRC_COMP`	Data Compare Register
`CRC_CTL`	Control Register
`CRC_DCNT`	Data Word Count Register
`CRC_DCNTCAP`	Data Count Capture Register
`CRC_DCNTRLD`	Data Word Count Reload Register
`CRC_DFIFO`	Data FIFO Register
`CRC_FILLVAL`	Fill Value Register
`CRC_INEN`	Interrupt Enable Register
`CRC_INEN_CLR`	Interrupt Enable Clear Register
`CRC_INEN_SET`	Interrupt Enable Set Register
`CRC_POLY`	Polynomial Register
`CRC_RESULT_CUR`	CRC Current Result Register
`CRC_RESULT_FIN`	CRC Final Result Register
`CRC_STAT`	Status Register

ADSP-SC58x CRC Interrupt List

Table 40-2: ADSP-SC58x CRC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
176	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
177	CRC1_DCNTEXP	CRC1 Datacount expiration	Level	
178	CRC0_ERR	CRC0 Error	Level	
179	CRC1_ERR	CRC1 Error	Level	

CRC Definitions

To make the best use of the CRC, it is useful to understand the following terms.

CRC

Acronym for Cyclic Redundancy Check. An error detection code that can detect changes within a block of data.

CRC Polynomial

The 32-bit polynomial used by the CRC engine to generate the look-up table required for the CRC implementation

LUT

Acronym for the Look-up Table. The look-up table is automatically generated from the supplied 32-bit CRC polynomial.

DMA

Acronym for Direct Memory Access. Used to describe a data transfer that takes place through a DMA channel allowing data distribution around a system without intervention from the core.

MDMA

Acronym for Memory-To-Memory DMA transfer that often requires the use of two DMA channels to transfer data from one memory region to another memory region. One DMA channel is configured as a source channel and the second as a destination channel.

CRC Block Diagram

The ***CRC Block Diagram*** shows the functional block diagram of the CRC. The following sections describe the blocks.

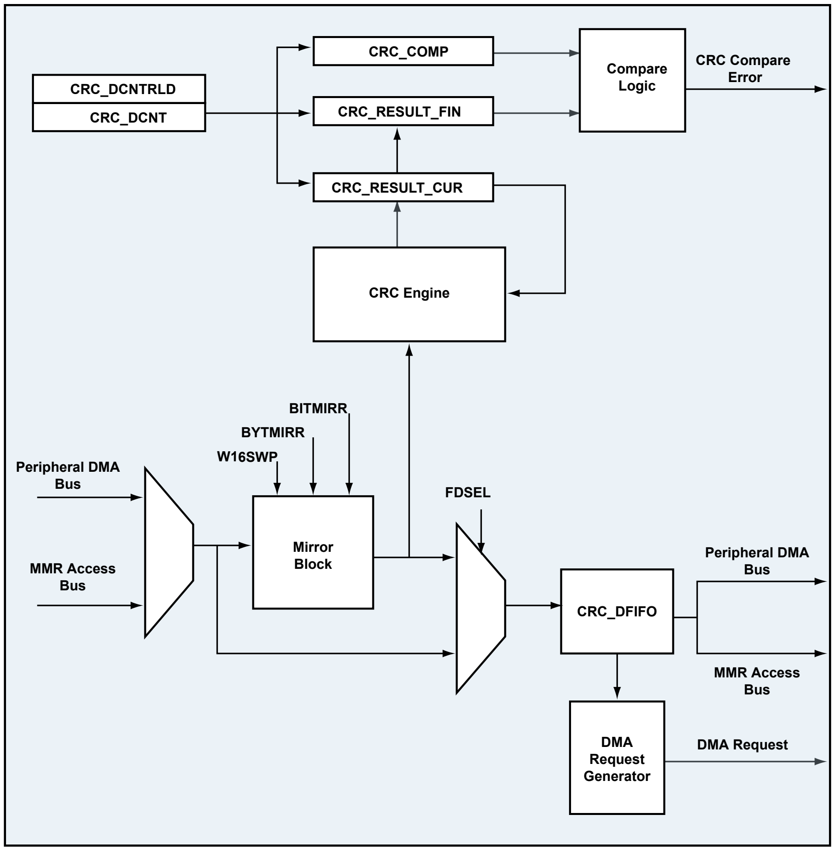

CRC_DCNTRLD
CRC_DCNT
CRC_COMP
CRC_RESULT_FIN
CRC_RESULT_CUR
Compare Logic
CRC Compare Error
CRC Engine
BITMIRR
BYTMIRR
W16SWP
Peripheral DMA Bus
MMR Access Bus
Mirror Block
FDSEL
CRC_DFIFO
Peripheral DMA Bus
MMR Access Bus
DMA Request Generator
DMA Request

Figure 40-2: CRC Block Diagram

Peripheral DMA Bus

The CRC peripheral provides both an incoming and outgoing datapath to the peripheral DMA bus. The MDMA source channel is interfaced to the incoming datapath providing data to the CRC peripheral. For memory transfer and data fill modes, the CRC uses the MDMA destination channel to either output the data from the CRC FIFO or use the data for the fill operation.

MMR Access Bus

The core uses the MMR access bus to access all the memory-mapped registers of the peripheral for configuration, status, and debug purposes. The core can also use the MMR access bus to feed data to the CRC peripheral or read data from the FIFO of the CRC peripheral. The CRC operation is an alternative to the DMA channel operation to read data from the FIFO.

Data received by MMR writes can transfer to destination DMA. Similarly, data received by source DMA can be output through the MMR interface. Optionally, intermediate results can be made available to the MMR interface.

Mirror Block

The mirror block individually controls bit-reversing of the polynomial, the computation results, and the expected result. Bit mirroring, byte mirroring, word swapping, and any combination of these operations can control endian and the reflection of processed data.

Data FIFO

The CRC data FIFO is a 32-bit-wide 4-entry FIFO. The FIFO is accessible to both the peripheral DMA bus and the MMR access bus. The FIFO status is accessible from the `CRC_STAT` register.

DMA Request Generator

The DMA request generator is responsible for granting incoming DMA requests from the source DMA channel and issuing outgoing DMA requests to the destination DMA channel.

CRC Engine

The CRC engine is a 32-bit CRC engine that implements the reduced table look-up scheme. The CRC engine provides support for a user-programmable 32-bit polynomial that the CRC uses to load the look-up table parameters required for the CRC calculation. The CRC engine is a two-cycle implementation operating on 16 bits of data per cycle.

Compare Logic

The compare logic takes the final CRC signature and compares it to the expected CRC signature, generating a CRC compare error when the signatures do not match. A compare error can cause an error interrupt request to the Cortex-M4 system or to SEC1.

CRC Architectural Concepts

A 32-bit polynomial is required before calculation of the CRC signature can occur. The CRC uses the polynomial to generate the contents of an internal look-up table that the reduced table look-up implementation requires. The look-up table is automatically generated when the polynomial is written. It must be initialized prior to any operation that requires the use of the CRC engine.

The mirror block logic can manipulate the data presented to the CRC engine before the CRC uses the data in the calculation of the CRC signature. The data mirror operation is configurable to allow for bit reversing, byte reversing, and 16-bit word swapping operations on the incoming data. For memory transfer compute-and-compare operations, programs can configure the peripheral to output the data in the same form in which it was received. Or, the operation can output the mirrored data in the same manner that it is presented to the CRC engine.

While the CRC peripheral is in operation, the status of the FIFO is continually updated and reflected in the `CRC_STAT` register. The FIFO status is required for core-based accesses to the CRC peripheral. The status indicates when:

- The CRC peripheral can receive data
- Data is available for reading from the FIFO
- The result of the `CRC_RESULT_CUR` register has been updated

The status of the `CRC_RESULT_CUR` register indicates that the current CRC calculation has completed and the result is available.

Look-up Table

The look-up table consists of a set of sixteen 32-bit registers that hardware populates automatically when a write access takes place to the `CRC_POLY` register. 16 clock cycles are required to generate all 16 look-up table entries. The status of the process for generating the look-up table is reflected in `CRC_STAT.LUTDONE` allowing for software to poll on the completion of the event or for generation of an interrupt.

NOTE: Hardware must populate the look-up table before any operation using the CRC peripheral can take place, even if the operation does not use the CRC engine. The peripheral does not issue any data requests until the table generation process is complete. In addition, the `CRC_STAT.IBR` field, that indicates the input buffer status as required for core-based transfers, is only valid upon completion of the process for generating the look-up table.

Data Mirroring

The data mirror block can be configured to manipulate the incoming data before the data passes to the CRC engine and, optionally, to the FIFO. This configuration allows the peripheral to handle various forms of endianness and to function with reflected polynomials.

There are three configuration bits that control the data mirroring process: `CRC_CTL.BITMIRR`, `CRC_CTL.BYTMIRR`, and `CRC_CTL.W16SWP`. The ***Data Mirroring Options*** table details how these options affect the incoming data and the output generated by the mirror block.

Table 40-3: Data Mirroring Options

W16SWP	BYTMIRR	BITMIRR	Output Data
0	0	0	Dout[31:0] = Din[31:0]
0	0	1	Dout[31:0] = Din[24:31],Din[16:23],Din[8:15],Din[0:7]
0	1	0	Dout[31:0] = Din[7:0],Din[15:8],Din[23:16],Din[31:24]
0	1	1	Dout[31:0] = Din[0:7],Din[8:15],Din[16:23],Din[24:31]
1	0	0	Dout[31:0] = Din[15:0], D[31:16]
1	0	1	Dout[31:0] = Din[8:15],Din[0:7], Din[24:31],Din[16:23]
1	1	0	Dout[31:0] = Din[23:16],Din[31:24], Din[7:0],Din[15:8]
1	1	1	Dout[31:0] = Din[16:23],Din[24:31], Din[0:7],Din[8:15]

When the CRC is configured to operate in the memory transfer compute-and-compare mode, the bit-reversed output data can be written to the FIFO. This feature is controlled through the `CRC_CTL.FDSEL` field.

In addition to providing bit swapping and mirror options to the incoming data, the CRC peripheral also supports bit mirroring on the following registers.

- `CRC_RESULT_CUR` and `CRC_RESULT_FIN`, controlled through the `CRC_CTL.RSLTMIRR` field. When mirroring is enabled, the values to be written to these registers are fully bit-reversed before the write operation occurs.
- `CRC_POLY`, controlled through the `CRC_CTL.POLYMIRR` field. When mirroring is enabled, the 32-bit polynomial is fully bit-reversed before the write operation to the register occurs.
- `CRC_COMP`, controlled through the `CRC_CTL.CMPMIRR` field. When mirroring is enabled, the contents to be loaded to this register are fully bit-reversed before the write operation occurs.

FIFO Status and Data Requests

The CRC peripheral provides indication of the input and output buffer status through `CRC_STAT.IBR` and `CRC_STAT.OBR` respectively. For core-based operations, software must monitor these status fields prior to writing to or reading from the CRC FIFO. No write to the CRC FIFO can occur while `CRC_STAT.IBR` indicates that the buffer is not ready to accept data. Similarly, the CRC FIFO cannot be read until `CRC_STAT.OBR` indicates that data is available.

The memory scan modes of operation only require the monitoring of the input buffer status. The memory transfer, compute-and-compare mode uses both input and output buffer status. If the current result of the CRC computation is required, then software must verify that the current operation has completed and that the intermediate result is ready. The `CRC_STAT.IRR` indicates the status.

NOTE: The memory transfer fill mode of operation requires the use of a DMA channel. The CRC does not support core reads from the CRC FIFO for this mode of operation.

Memory transfer, compute-and-compare mode uses burst transactions to make the most efficient use of the available resources. In this mode, when the FIFO is initially empty and the peripheral is enabled, the `CRC_STAT.IBR` bit indicates that the CRC is ready to accept data. The peripheral generates data requests to the source DMA channel (if the CRC uses DMA). While the number of words remaining in the `CRC_DCNT` register is greater than the FIFO depth, the peripheral issues data requests or accepts incoming data in bursts. The peripheral continues until the CRC FIFO becomes full.

Once full, the `CRC_STAT.IBR` and `CRC_STAT.OBR` bits are updated, and then the CRC issues outgoing data requests. Only when the FIFO is empty can the peripheral accept further incoming data, and the `CRC_STAT.IBR` and `CRC_STAT.OBR` bits are updated once again.

Once `CRC_DCNT` is decremented such that the number of words waiting for processing is less than the number required to fill the FIFO, the burst mode of operation is disabled. Incoming data is accepted as long as the FIFO is not full. Outgoing data is available as long the FIFO is not empty. Therefore, there are no restrictions requiring the word count to be a multiple of the FIFO depth.

All other CRC modes of operation indicate that incoming data can be accepted as long as the FIFO is not full. Outgoing data is available as long as the FIFO is not empty.

The `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit configurations also influence how the CRC generates data requests and status bits. The following list describes the bits.

- The `CRC_CTL.OBRSTALL` bit can be configured such that the CRC peripheral stalls as soon as there is output data available in the FIFO. Use this mode of operation only in memory transfer, compute-and-compare mode. This mode results in the processing of one 32-bit word at a time. The peripheral does not request or accept incoming data until the current value being processed is read from the peripheral.
- The `CRC_CTL.IRRSTALL` bit can be configured so that the CRC peripheral stalls all further incoming data requests until the `CRC_RESULT_CUR` register is read after being updated. Use this mode of operation for CRC signature generation. It is not applicable to memory transfer data-fill mode or memory scan data-verify mode of operation.

CRC Operating Modes

The following sections describe the various operating modes of the CRC interface.

Data Transfer Modes

The CRC peripheral supports two main categories of operation involving data transfers:

- Memory scan mode
- Memory transfer mode

Memory scan modes are read-only operations that allow the contents of memory to be read into the peripheral and verified for correctness. There are two forms of memory scan mode:

- CRC compute-and-compare performs a CRC calculation on data presented to the peripheral and compares the CRC result with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data verify compares each 32-bit data word presented to the CRC peripheral to a pre-loaded 32-bit value and generates an error when the data differs.

Both of these modes of operation require, at the most, a single DMA channel to read the data from memory into the peripheral. No data is forwarded to the data output or destination DMA. The CRC can also use core-driven transfers for either of these modes of operation.

The memory transfer modes involve memory write or memory read-and-write operations allowing for memory to be initialized or transferred from one region of memory to another. There are two forms of memory transfer mode:

- CRC compute-and-compare performs a full data transfer from one memory region to another memory region. The CRC generates a signature on the data presented and compares it with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data fill initializes a region of memory with a pre-loaded 32-bit constant value.

The CRC compute-and-compare mode of operation requires both incoming and outgoing data channels. The operation occurs either using DMA channels, using core-driven write or read operations to and from the FIFO or using a combination of both. The data fill mode of operation requires only a memory write DMA destination channel—this mode does not support core driven operations.

Memory Scan Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the `CRC_CTL.OPMODE` field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the `CRC_DCNT` register. The accumulated result of the CRC operation is contained in the `CRC_RESULT_CUR` register. As the CRC engine processes each 32-bit word, the `CRC_DCNT` register is decremented and `CRC_RESULT_CUR` is updated.

Once `CRC_DCNT` decrements to zero, the contents of the `CRC_RESULT_CUR` register are copied to `CRC_RESULT_FIN` and `CRC_STAT.DCNTEXP` is updated accordingly. The CRC uses the `CRC_COMP` register to store the expected result of the operation. After the CRC calculation, `CRC_COMP` is compared with `CRC_RESULT_FIN` and `CRC_STAT.CMPERR` is updated to reflect the status of the compare operation. `CRC_STAT.CMPERR` must be cleared before the next CRC operation is performed.

The CRC peripheral also contains the `CRC_DCNTRLD` register. The CRC uses this register to reload `CRC_DCNT` upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRF`. This configuration provides a way to reset `CRC_RESULT_CUR` to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Scan Data Verify

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the `CRC_CTL.OPMODE` field. Each 32-bit word of the data stream is compared with a constant value that is stored in the `CRC_COMP` register. The `CRC_DCNT` register contains the number of words for comparison. The `CRC_DCNT` register is decremented upon receiving a new 32-bit word from the data stream. If the compare operation fails, the `CRC_STAT.CMPERR` bit is updated and the contents of `CRC_DCNT` are captured in the `CRC_DCNTCAP` register. This information can be used to identify the location in the data stream where the error occurred. Clear the `CRC_STAT.CMPERR` field to reenable capturing of further errors.

Once `CRC_DCNT` decrements to zero, `CRC_STAT.DCNTEXP` is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the `CRC_CTL.OPMODE` field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the `CRC_DCNT` register. The accumulated result of the CRC operation is contained in the `CRC_RESULT_CUR` register. As the CRC engine processes each 32-bit word, the `CRC_DCNT` register is decremented and `CRC_RESULT_CUR` is updated.

Once `CRC_DCNT` decrements to zero, the contents of the `CRC_RESULT_CUR` register are copied to `CRC_RESULT_FIN` and `CRC_STAT.DCNTEXP` is updated accordingly. The CRC uses the `CRC_COMP` register to store the expected result of the operation. Upon completion of the CRC calculation, `CRC_COMP` is compared with `CRC_RESULT_FIN` and `CRC_STAT.CMPERR` is updated to reflect the status of the compare operation. Clear `CRC_STAT.CMPERR` before the next CRC operation is performed.

The CRC peripheral also contains `CRC_DCNTRLD` register. The CRC uses this register to reload `CRC_DCNT` upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRF`. This configuration provides a means to reset `CRC_RESULT_CUR` to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Data Fill Mode

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the `CRC_CTL.OPMODE` field. The `CRC_FILLVAL` register is written with a 32-bit value. The CRC uses this value to initialize a block memory through the memory-to-memory DMA destination channel. When the CRC peripheral and the DMA destination channel are enabled, the contents of the `CRC_FILLVAL` register is written to the DMA channel to initialize the memory region. The `CRC_DCNT` register contains the number of words for the write operation.

Once `CRC_DCNT` decrements to zero, `CRC_STAT.DCNTEXP` is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the data expiration event to generate an interrupt.

CRC Event Control

The CRC peripheral can enable certain CRC status operations to generate an interrupt event to the Cortex-M4 system event controller. There, a CRC error can be qualified as a system fault.

Interrupt Signals

The CRC peripheral can generate two interrupt requests that are optionally enabled as interrupts with the Cortex-M4 core NVIC, or as events to SEC1 for fault operations. One is a CRC status interrupt and the other is a CRC error interrupt.

The `CRC_STAT.CMPERR` status bit can be configured as an interrupt and is signaled through the CRC error interrupt signal. The `CRC_STAT.CMPERR` status field is set whenever the CRC peripheral performs a compare operation that fails. This status can be the result of a failed memory scan data-verify operation that compares the contents of a memory range with a constant 32-bit value. Or, it can be the result of a CRC signature calculated for a memory region that does not match the expected pre-programmed result for a memory-compare operation.

The `CRC_STAT.DCNTEXP` status bit is set when the `CRC_DCNT` register has decremented to zero. The status indicates that the CRC peripheral has now processed all the data requested for the current CRC operation. The CRC can also use this signal to generate an interrupt. The interrupt is signaled on the CRC status interrupt signal.

Both these status bits can be configured to generate and interrupt through the `CRC_INEN` register. The `CRC_INEN` register also has bit set, `CRC_INEN_SET`, and bit clear `CRC_INEN_CLR` equivalent registers that the CRC uses for the enabling and disabling of these interrupt sources.

The `CRC_STAT` register has two write one to clear (W1C) fields for clearing the two interrupt sources.

NOTE: Disabling the CRC peripheral through the `CRC_CTL.BLKEN` bit does not result in the clearing of interrupt sources. Clear the interrupt sources using a W1C operation to the `CRC_STAT` register.

CRC Programming Model

It is important to note the following restrictions when using the CRC peripheral with the DMA channels:

1. When enabling the CRC peripheral and the DMA channels, enable the CRC peripheral prior to enabling the DMA channels.
2. When disabling the CRC peripheral and the DMA channels, disable the DMA channels prior to disabling the CRC peripheral.

CRC Mode Configuration

Describes a number of tasks showing the various operation modes of the CRC peripheral.

- Look-up Table Generation
- Core Driven Memory Scan Compute-and-Compare Mode
- DMA Driven Memory Scan Compute-and-Compare Mode
- Core Driven Memory Scan Data Verify Mode
- DMA Driven Memory Scan Data Verify Mode
- Core Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Data Fill Mode

Look-up Table Generation

Describes the steps required to initialize the CRC peripheral LUT.

1. Write the 32-bit CRC polynomial of choice to the `CRC_POLY` register.

   ADDITIONAL INFORMATION: This operation results in the CRC peripheral starting the LUT initialization process. The `CRC_STAT.LUTDONE` bit is updated to reflect the operation is in progress.

2. Poll the `CRC_STAT.LUTDONE` bit until the status bit indicates that the operation is completed.

The CRC peripheral has completed initialization of all the LUT registers and is now ready for data operations. The `CRC_STAT.LUTDONE` bit remains in the current state until the `CRC_POLY` register is written again, or the peripheral or processor are reset.

Core Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using core transactions. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling the `CRC_CTL` register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per `CRC_CTL.BLKEN`

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute-and-compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

   - Disable the `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit options for this task example.
   - Configure all mirroring and bit reversal options.
   - Configure CRC auto-clear options.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral.

   a. While `CRC_STAT.IBR` bit indicates that the input buffer is ready, write the `CRC_DFIFO` register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat this step until all data has been written.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write to the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear of these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed. The final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing more CRC operations.

DMA Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using DMA transactions. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling `CRC_CTL`.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit.

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

   ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final operation.

5. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

   - Disable the `CRC_CTL.OBRSTALL` and `CRC_CTL.IRRSTALL` bit options for this task example.
   - Configure all mirroring and bit reversal options.

- Configure all CRC auto clear options.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed and the final result indicated is through `CRC_STAT.CMPERR` and the corresponding interrupt, when enabled.

Clear any W1C CRC status bits before performing a further CRC operation. Clear any W1C status bits of the memory-to-memory source DMA channel before the next CRC operation.

Core Driven Memory Scan Data Verify Mode

Reads a region of memory using core transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per `CRC_CTL.BLKEN`

The interrupt service routine for the compare error interrupt reads and stores the contents of `CRC_DCNTCAP` register to a buffer before clearing the compare error interrupt.

1. Initialize the `CRC_DCNT` register.

ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

   ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan data verify mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

6. Write memory region data to the CRC peripheral.

   a. Poll the `CRC_STAT.IBR` bit until input buffer is ready.

   b. Write the `CRC_DFIFO` register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat these two steps until the entire memory region has been written to the CRC peripheral.

7. Poll the `CRC_INEN_SET.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the `CRC_DCNTCAP` register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write to the `CRC_STAT` to clear both the `CRC_INEN_SET.DCNTEXP` and `CRC_INEN.CMPERR` bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, the clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the `CRC_INEN.CMPERR` bit and the corresponding interrupt, when enabled. Each comparison error is traceable due to the logging of `CRC_DCNTCAP` from within the compare error interrupt handler.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Scan Data Verify Mode

The memory scan data verify mode reads a region of memory using DMA transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

The interrupt service routine for the compare error interrupt reads and stores the contents of the `CRC_DCNTCAP` register to a buffer before clearing the compare error interrupt.

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan data verify mode and `CRC_CTL.BLKEN` configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write the data.

6. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

7. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the `CRC_DCNTCAP` register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled. Each comparison error is traceable due to the logging of the `CRC_DCNTCAP` register from within the compare error interrupt handler.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

Core Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using core transactions while copying the contents to another memory region. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling the `CRC_CTL` register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of the current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute-and-compare mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

   a. Disable the `CRC_CTL.OBRSTALL` bit and the `CRC_CTL.IRRSTALL` bit options for this task example.

   b. Configure all mirroring and bit reversal options.

   c. Configure CRC auto clear options

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral and read it back to the new destination.

   a. While the `CRC_STAT.IBR` bit indicates that the input buffer is ready, write the `CRC_DFIFO` register with 32-bit data.

   b. While the `CRC_STAT.OBR` bit indicates that the output buffer is ready, read the `CRC_DFIFO` register and store data to new destination.

      ADDITIONAL INFORMATION: Repeat these two steps until all required data has been processed through the CRC peripheral and copied to the new destination.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both `CRC_STAT.DCNTEXP` and `CRC_STAT.CMPERR` bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The memory region has been copied to a new location and an integrity check of the memory through the expected CRC signature has also completed. The final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using DMA transactions. The memory region is also copied to another memory region using memory-to-memory DMA transfers. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling `CRC_CTL`.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` register.

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

   ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_RESULT_CUR` register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the `CRC_COMP` register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the `CRC_INEN` register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory scan compute compare mode and `CRC_CTL.BLKEN` configured to enable the CRC peripheral.
    a. Disable the `CRC_CTL.OBRSTALL` and the `CRC_CTL.IRRSTALL` bit options for this task example.
    b. Configure all mirroring and bit reversal options
    c. Configure CRC auto clear options

    The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode and destination DMA channel for memory write STOP mode.

    ADDITIONAL INFORMATION: This step starts the data transfer from one memory region to another through the memory-to-memory DMA channels and the CRC peripheral.

8. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

    ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the `CRC_STAT.CMPERR` bit if the interrupt was disabled to check for a compare error.

    ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the `CRC_STAT` register to clear both the `CRC_STAT.DCNTEXP` and the `CRC_STAT.CMPERR` bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The integrity check of the memory through the expected CRC signature has completed and the final result is indicated through the `CRC_STAT.CMPERR` bit and the corresponding interrupt when enabled. The memory region has also been copied to its final destination.

Clear any W1C CRC status bits before performing a further CRC operation. Also, clear any W1C status bits of the memory-to-memory source and destination DMA channels before the next CRC operation.

DMA Driven Memory Transfer Data Fill Mode

This mode initializes a region of memory to a constant 32-bit value using DMA transactions.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the `CRC_CTL.BLKEN` bit

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the `CRC_DCNTRLD` register.

   ADDITIONAL INFORMATION: This value is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the `CRC_FILLVAL` register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the CRC module uses to fill the memory region.

4. Initialize the `CRC_INEN` register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of block completion. Configure these interrupts as required. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the `CRC_CTL` register with the `CRC_CTL.OPMODE` bit set to memory transfer fill mode and the `CRC_CTL.BLKEN` bit configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and is ready for the DMA channel to write data.

6. Configure and enable the memory-to-memory destination DMA channel for memory write STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer taking the constant 32-bit value from the CRC peripheral and writing the data to the DMA channel.

7. Poll the `CRC_STAT.DCNTEXP` bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

8. Write the `CRC_STAT` register to clear the `CRC_STAT.DCNTEXP` bit.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear this status bit within the interrupt handlers for the respective interrupts.

   The CRC memory transfer fill operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

The memory region is now filled with the constant data and the CRC peripheral is ready to be configured for a new operation.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

ADSP-SC58x CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 40-4: ADSP-SC58x CRC Register List

Name	Description
`CRC_COMP`	Data Compare Register
`CRC_CTL`	Control Register
`CRC_DCNT`	Data Word Count Register
`CRC_DCNTCAP`	Data Count Capture Register
`CRC_DCNTRLD`	Data Word Count Reload Register
`CRC_DFIFO`	Data FIFO Register
`CRC_FILLVAL`	Fill Value Register
`CRC_INEN`	Interrupt Enable Register
`CRC_INEN_CLR`	Interrupt Enable Clear Register
`CRC_INEN_SET`	Interrupt Enable Set Register
`CRC_POLY`	Polynomial Register
`CRC_RESULT_CUR`	CRC Current Result Register
`CRC_RESULT_FIN`	CRC Final Result Register
`CRC_STAT`	Status Register

Data Compare Register

The `CRC_COMP` register contains the value corresponding to the expected CRC result or signature for the current data stream. At the end of the operation, the content of this register is used to compare against the result produced by the CRC operation. In data verify mode, each incoming data value is compared with the content of this register.

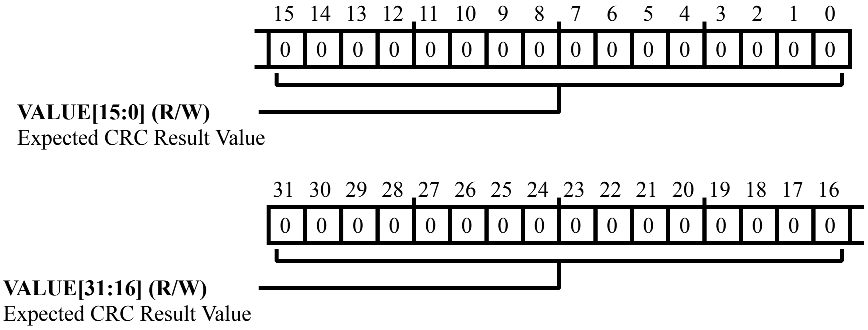

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Expected CRC Result Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Expected CRC Result Value

Figure 40-3: CRC_COMP Register Diagram

Table 40-5: CRC_COMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Expected CRC Result Value. The `CRC_COMP.VALUE` bit field contains the value corresponding to the expected CRC result or signature for the current data stream.

Control Register

The CRC_CTL register configures the operation modes and settings for the CRC.

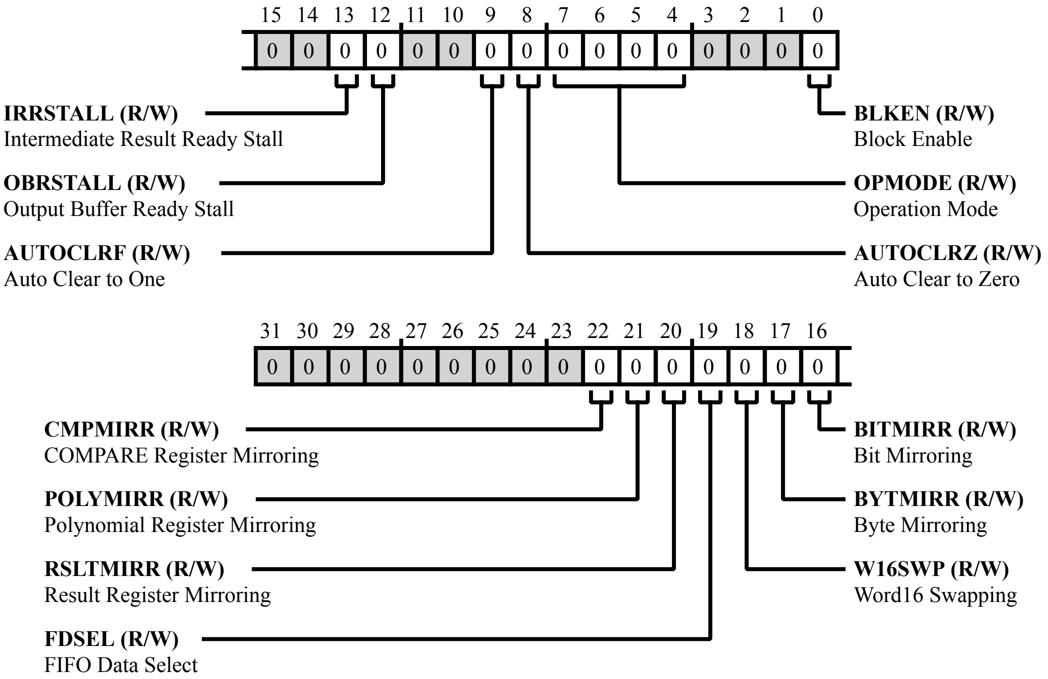

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IRRSTALL (R/W)
Intermediate Result Ready Stall
OBRSTALL (R/W)
Output Buffer Ready Stall
AUTOCLRF (R/W)
Auto Clear to One
BLKEN (R/W)
Block Enable
OPMODE (R/W)
Operation Mode
AUTOCLRZ (R/W)
Auto Clear to Zero
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CMPMIRR (R/W)
COMPARE Register Mirroring
POLYMIRR (R/W)
Polynomial Register Mirroring
RSLTMIRR (R/W)
Result Register Mirroring
FDSEL (R/W)
FIFO Data Select
BITMIRR (R/W)
Bit Mirroring
BYTMIRR (R/W)
Byte Mirroring
W16SWP (R/W)
Word16 Swapping

Figure 40-4: CRC_CTL Register Diagram

Table 40-6: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	CMPMIRR	COMPARE Register Mirroring. The CRC_CTL.CMPMIRR bit enables data mirroring for the CRC_COMP compare register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for comparison with the CRC_RESULT_FIN register.	
		0	Disable compare mirroring
		1	Enable compare mirroring
21 (R/W)	POLYMIRR	Polynomial Register Mirroring. The CRC_CTL.POLYMIRR bit enables data mirroring for the CRC_POLY polynomial register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for CRC computations.	
		0	Disable polynomial mirroring
		1	Enable polynomial mirroring

Table 40-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	RSLTMIRR	Result Register Mirroring. The `CRC_CTL.RSLTMIRR` bit enables data mirroring for the `CRC_RESULT_CUR` and `CRC_RESULT_FIN` result registers. When enabled, the 32-bit values in these registers are fully bit mirrored (reversed).	
		0	Disable result mirroring
		1	Enable result mirroring
19 (R/W)	FDSEL	FIFO Data Select. The `CRC_CTL.FDSEL` bit selects whether the CRC writes modified or unmodified data to the FIFO in memory transfer mode. If enabled, the data written is affected by the state of the data mirroring selections (`CRC_CTL.BITMIRR`, `CRC_CTL.BYTMIRR`, and `CRC_CTL.W16SWP`) before being written to the FIFO.	
		0	Write unmodified data to FIFO
		1	Write modified data to FIFO
18 (R/W)	W16SWP	Word16 Swapping. The `CRC_CTL.W16SWP` bit enables the CRC's data mirror block to swap the upper and lower 16-bit words within the 32-bit input data, before further processing.	
		0	Disable word16 swapping
		1	Enable word16 swapping
17 (R/W)	BYTMIRR	Byte Mirroring. The `CRC_CTL.BYTMIRR` bit enables the CRC's data mirror block to mirror the bytes within the 32-bit input data, before further processing.	
		0	Disable byte mirroring
		1	Enable byte mirroring
16 (R/W)	BITMIRR	Bit Mirroring. The `CRC_CTL.BITMIRR` bit enables the CRC's data mirror block to mirror the bits within each byte of the 32-bit input data, before further processing.	
		0	Disable bit mirroring
		1	Enable bit mirroring
13 (R/W)	IRRSTALL	Intermediate Result Ready Stall. The `CRC_CTL.IRRSTALL` bit enables stalling the state machine for input data when there is a valid intermediate result to be read in the `CRC_RESULT_CUR` register. This feature should be used only in CRC computation modes (for example, `CRC_CTL.OPMODE` =1 or =3).	
		0	Do not stall
		1	Stall on IRR

Table 40-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	OBRSTALL	Output Buffer Ready Stall. The `CRC_CTL.OBRSTALL` bit enables stalling the state machine for input data when there is valid data in the output buffer. This feature should be used only in memory-to-memory transfer modes (for example, `CRC_CTL.OPMODE` =1).	
		0	Do not stall
		1	Stall on OBR
9 (R/W)	AUTOCLRF	Auto Clear to One. The `CRC_CTL.AUTOCLRF` bit enables auto clear to one when the CRC is in intermediate results ready stall mode (`CRC_CTL.IRRSTALL`=1) and the CRC data count expires (`CRC_DCNT`=0). Note that the `CRC_CTL.AUTOCLRZ` bit must be disabled, or the `CRC_CTL.AUTOCLRF` bit has no effect.	
		0	No auto clear
		1	Auto clear
8 (R/W)	AUTOCLRZ	Auto Clear to Zero. The `CRC_CTL.AUTOCLRZ` bit enables auto clear to zero when the CRC is in intermediate results ready stall mode (`CRC_CTL.IRRSTALL`=1) and the CRC data count expires (`CRC_DCNT`=0). Note that `CRC_CTL.AUTOCLRF` must be disabled, or the `CRC_CTL.AUTOCLRZ` has no effect.	
		0	No auto clear
		1	Auto clear
7:4 (R/W)	OPMODE	Operation Mode. The `CRC_CTL.OPMODE` bit field selects the memory transfer or scan mode.	
		0	Reserved
		1	CRC compute/compare memory transfer
		2	Data fill memory transfer
		3	CRC compute/compare memory scan
		4	Data verify memory scan
0 (R/W)	BLKEN	Block Enable. The `CRC_CTL.BLKEN` bit enables and disables the CRC operation.	
		0	Disable
		1	Enable

Data Word Count Register

The `CRC_DCNT` register holds the word count that is used for the CRC operation. On transfer of every 32-bit word, the CRC decrements by 1 the content of this register. When the count decrements to zero, this event triggers a CRC compare action, and the `CRC_DCNT` register is automatically loaded from the `CRC_DCNTRLD` register for the next CRC operation.

Note that the initial value programmed into the `CRC_DCNT` register may be different from what is programmed in the `CRC_DCNTRLD` register.

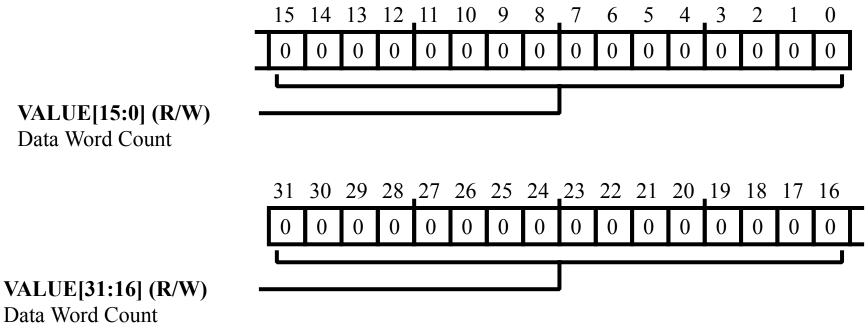

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Data Word Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Data Word Count

Figure 40-5: CRC_DCNT Register Diagram

Table 40-7: CRC_DCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Word Count. The `CRC_DCNT.VALUE` bit field holds the word count that is used for the CRC operation.

Data Count Capture Register

The `CRC_DCNTCAP` register captures the `CRC_DCNT` value when a compare operation fails in data verify mode. This capture can be used to track the position of an error in the data stream. The capture operation is enabled only if the `CRC_STAT.CMPERR` bit indicates no compare error. After an error occurs and the data count is captured, no further errors are logged until the `CRC_STAT.CMPERR` bit is cleared. To obtain the position of an error in the data stream, subtract the `CRC_DCNTCAP` register value from the initial `CRC_DCNT`.

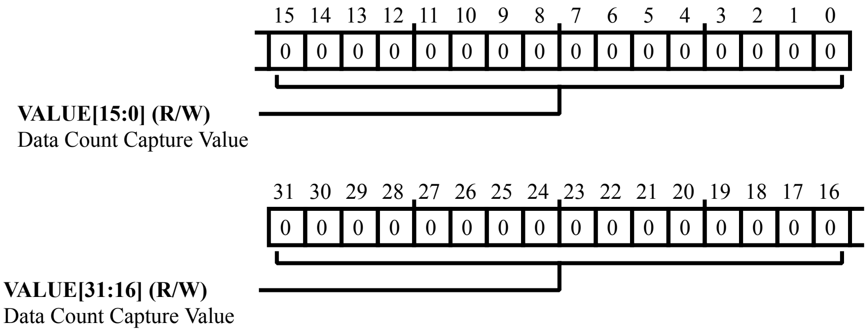

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Data Count Capture Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Data Count Capture Value

Figure 40-6: CRC_DCNTCAP Register Diagram

Table 40-8: CRC_DCNTCAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Count Capture Value. The `CRC_DCNTCAP.VALUE` bit field contains the `CRC_DCNT` value when a compare operation fails in data verify mode.

Data Word Count Reload Register

The CRC_DCNTRLD register holds the value that the CRC automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0. At startup, the value programmed in CRC_DCNT and the CRC_DCNTRLD register could be different. So, for the first iteration, the CRC operation happens for the count initially programmed in the CRC_DCNT register. While for subsequent CRC operations, the count is taken from the CRC_DCNTRLD register.

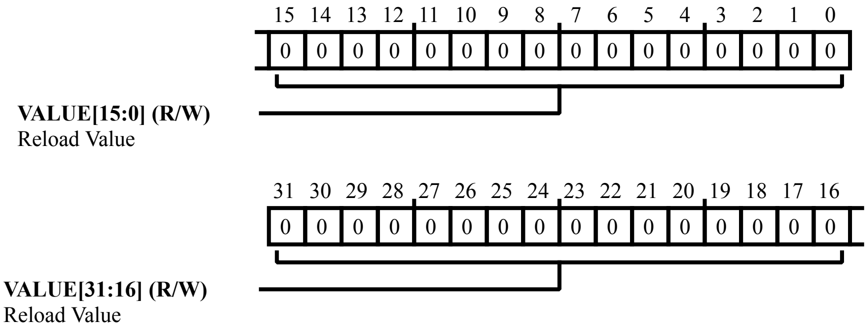

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Reload Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Reload Value

Figure 40-7: CRC_DCNTRLD Register Diagram

Table 40-9: CRC_DCNTRLD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reload Value. The CRC_DCNTRLD.VALUE bit field holds the value that automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0.

Data FIFO Register

In memory transfer mode (non-data fill mode), the data from the DMA or processor core buses is written into the `CRC_DFIFO` on each input data grant (DMA grant or core write). Data is read from this FIFO on each output data grant (DMA grant or core read). FIFO status information is available in the `CRC_STAT` register. Whenever, the FIFO has valid data, output data requests are generated.

Note that in non-memory transfer mode and in data fill mode, the input data does not get written into this FIFO. So, this register should not be read in these modes.

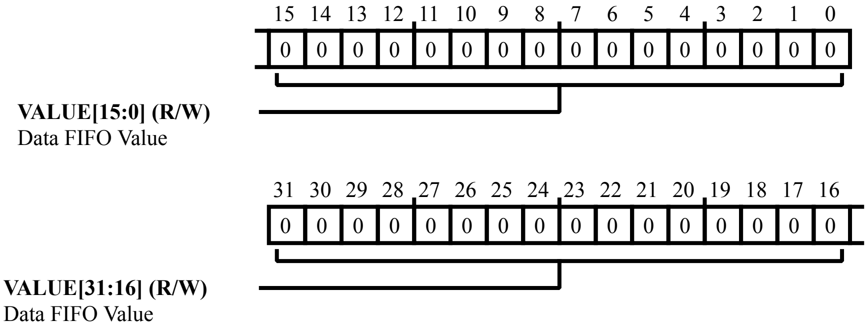

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Data FIFO Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Data FIFO Value

Figure 40-8: CRC_DFIFO Register Diagram

Table 40-10: CRC_DFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data FIFO Value. The `CRC_DFIFO.VALUE` bit field is the data from the DMA or processor core buses.

Fill Value Register

The `CRC_FILLVAL` register holds the value that the CRC uses for the memory fill operation. In data fill mode, the value programmed in this register is used for the memory fill operation.

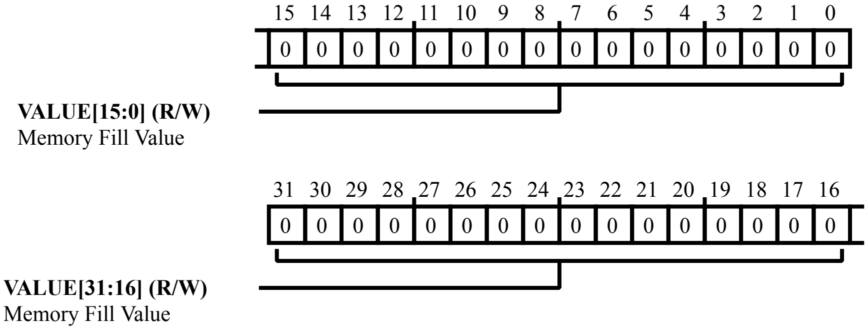

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Memory Fill Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Memory Fill Value

Figure 40-9: CRC_FILLVAL Register Diagram

Table 40-11: CRC_FILLVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Memory Fill Value. The `CRC_FILLVAL.VALUE` bit field holds the value that the CRC uses for the memory fill operation.

Interrupt Enable Register

The `CRC_INEN` register unmasks (enables) or masks (disables) interrupt requests generated in the CRC from going to the processor core.

Note that CRC interrupts are not disabled when the CRC is disabled (`CRC_CTL.BLKEN` =0).

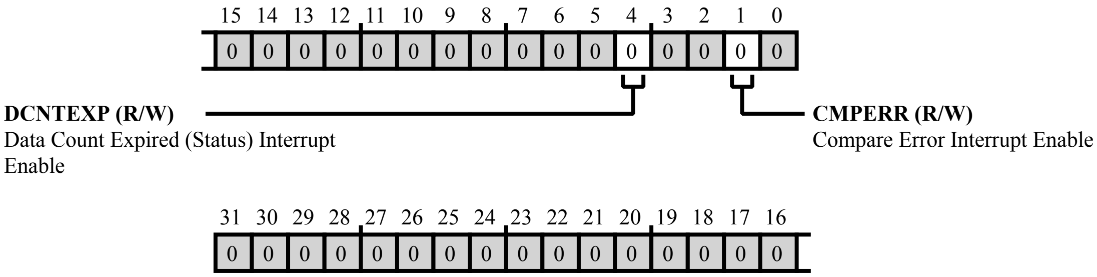

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DCNTEXP (R/W)
Data Count Expired (Status) Interrupt Enable
CMPERR (R/W)
Compare Error Interrupt Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 40-10: CRC_INEN Register Diagram

Table 40-12: CRC_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DCNTEXP	Data Count Expired (Status) Interrupt Enable. The `CRC_INEN.DCNTEXP` enables (unmasks) the data count expired (CRC status) interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/W)	CMPERR	Compare Error Interrupt Enable. The `CRC_INEN.CMPERR` enables (unmasks) the data compare interrupt, which is generated when CRC data comparison fails.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Interrupt Enable Clear Register

The `CRC_INEN_CLR` register permits clearing individual bits in the `CRC_INEN` register without affecting other bits in the register.

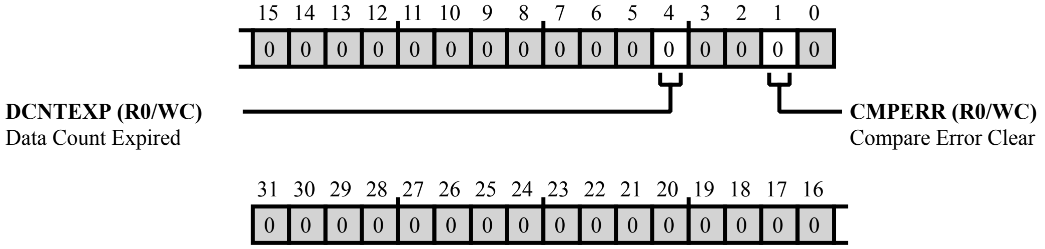

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DCNTEXP (R0/WC)
Data Count Expired
CMPERR (R0/WC)
Compare Error Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 40-11: CRC_INEN_CLR Register Diagram

Table 40-13: CRC_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WC)	DCNTEXP	Data Count Expired. The `CRC_INEN_CLR.DCNTEXP` bit clears the data count expired (status) interrupt.	
		0	No effect
		1	Clear bit
1 (R0/WC)	CMPERR	Compare Error Clear. The `CRC_INEN_CLR.CMPERR` bit clears the compare error interrupt.	
		0	No effect
		1	Clear bit

Interrupt Enable Set Register

The `CRC_INEN_SET` register permits setting individual bits in the `CRC_INEN` register without affecting other bits in the register.

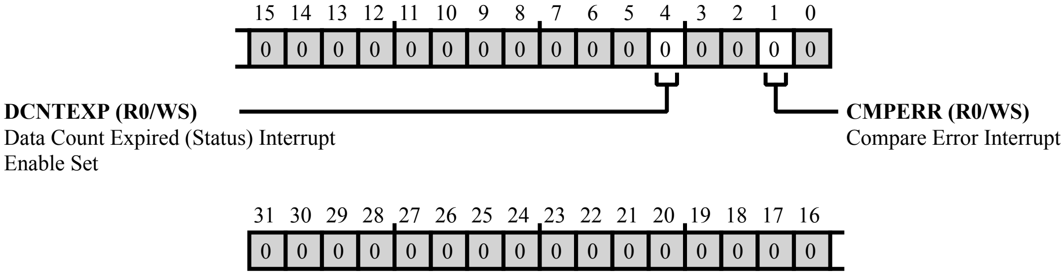

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DCNTEXP (R0/WS)
Data Count Expired (Status) Interrupt Enable Set
CMPERR (R0/WS)
Compare Error Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 40-12: CRC_INEN_SET Register Diagram

Table 40-14: CRC_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WS)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Set. The `CRC_INEN_SET.DCNTEXP` bit sets the data count expired (status) interrupt.	
		0	No effect
		1	Set bit
1 (R0/WS)	CMPERR	Compare Error Interrupt. The `CRC_INEN_SET.CMPERR` bit sets the compare error interrupt.	
		0	No effect
		1	Set bit

Polynomial Register

The `CRC_POLY` register holds a 32-bit polynomial for CRC operations. Bit 31 corresponds to the coefficient of x31 of the CRC polynomial, bit 30 corresponds to the coefficient of x30, and so on through bit 0. A coefficient of x32 is assumed to be "1" for any polynomial that is selected. Based on the polynomial in the `CRC_POLY` register, the CRC generates a look-up table (LUT), which is used to compute the CRC of the incoming data stream.

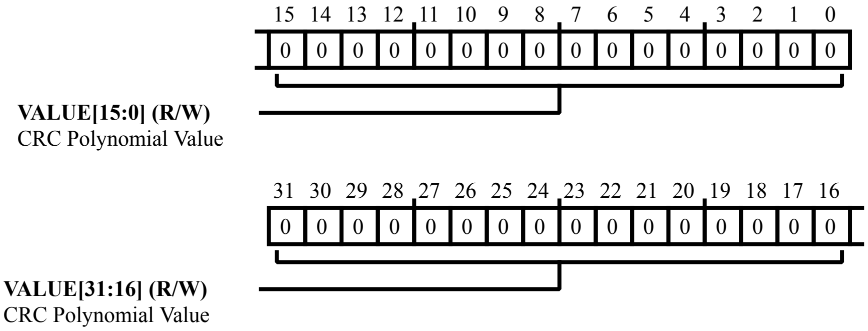

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
CRC Polynomial Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
CRC Polynomial Value

Figure 40-13: CRC_POLY Register Diagram

Table 40-15: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Polynomial Value. The `CRC_POLY.VALUE` bit field holds the 32-bit polynomial for CRC operations.

CRC Current Result Register

The `CRC_RESULT_CUR` register holds the current or intermediate CRC result. It is updated when new data is written into the CRC. Each time the `CRC_DCNT` expires, the CRC loads the value from this register into the `CRC_RESULT_FIN` register. The `CRC_RESULT_CUR` register may be set to auto clear to zero or auto clear to ones when `CRC_DCNT` expires by configuring the `CRC_CTL.AUTOCLRZ` and `CRC_CTL.AUTOCLRF` bits. Before starting a CRC operation, the `CRC_RESULT_CUR` register should be programmed to the desired value.

Note that this register can be read by the processor core at any time.

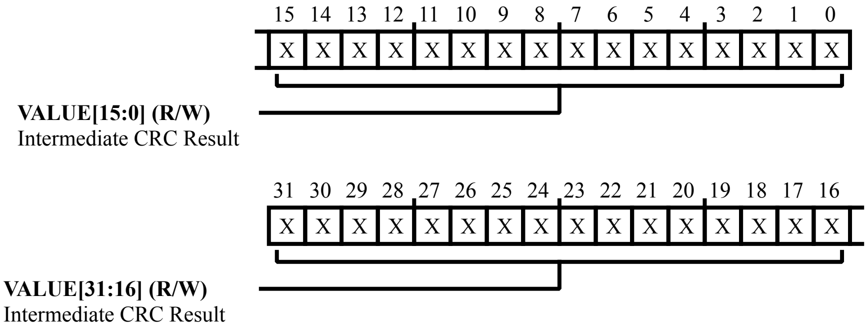

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
X X X X X X X X X X X X X X X X
VALUE[15:0] (R/W)
Intermediate CRC Result
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
X X X X X X X X X X X X X X X X
VALUE[31:16] (R/W)
Intermediate CRC Result

Figure 40-14: CRC_RESULT_CUR Register Diagram

Table 40-16: CRC_RESULT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Intermediate CRC Result. The `CRC_RESULT_CUR.VALUE` bit field holds the current or intermediate CRC result.

CRC Final Result Register

The `CRC_RESULT_FIN` register holds the final CRC computed for a data stream. A data stream is a DMA of `CRC_DCNT` number of words into the CRC. When `CRC_DCNT` decrements to zero for each data stream, the CRC loads the `CRC_RESULT_FIN` register with the value from the `CRC_RESULT_CUR` register.

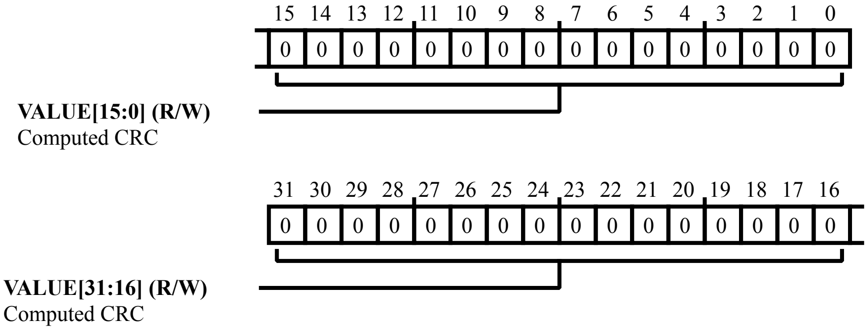

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Computed CRC
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Computed CRC

Figure 40-15: CRC_RESULT_FIN Register Diagram

Table 40-17: CRC_RESULT_FIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Computed CRC. The `CRC_RESULT_FIN.VALUE` bit field holds the final CRC computed for a data stream.

Status Register

The CRC_STAT register indicates the status for CRC operations and interrupt generation.

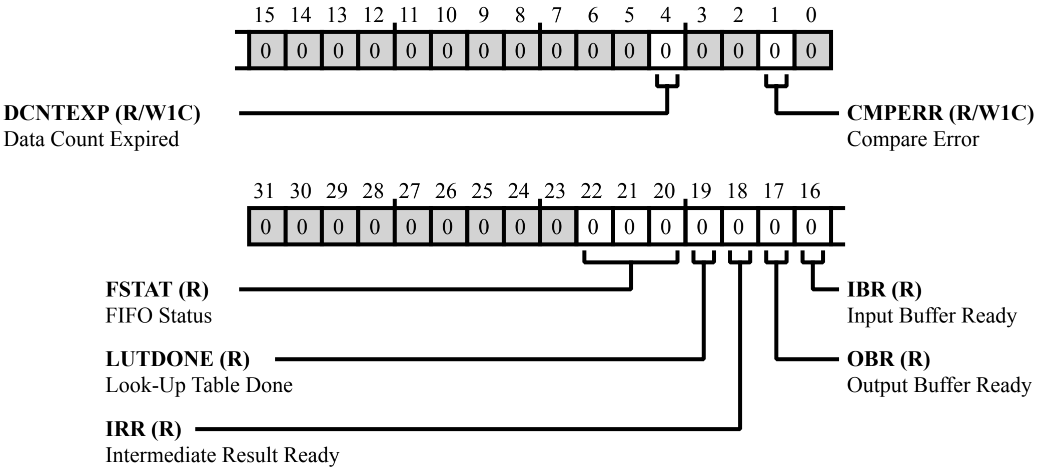

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DCNTEXP (R/W1C)
Data Count Expired
CMPERR (R/W1C)
Compare Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FSTAT (R)
FIFO Status
IBR (R)
Input Buffer Ready
LUTDONE (R)
Look-Up Table Done
OBR (R)
Output Buffer Ready
IRR (R)
Intermediate Result Ready

Figure 40-16: CRC_STAT Register Diagram

Table 40-18: CRC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22:20 (R/NW)	FSTAT	FIFO Status. The CRC_STAT.FSTAT indicates the current FIFO status. This field is read-only.	
		0	FIFO empty
		1	FIFO has 1 data
		2	FIFO has 2 data
		3	FIFO has 3 data
		4	FIFO has 4 data (full)
19 (R/NW)	LUTDONE	Look-Up Table Done. The CRC_STAT.LUTDONE bit indicates that the CRC has generated the look-up table for the current polynomial. This read-only bit is cleared at reset and cleared when the CRC_POLY is written.	
		0	No status
		1	LUT generation done

Table 40-18: CRC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/NW)	IRR	Intermediate Result Ready. The `CRC_STAT.IRR` bit indicates that the CRC has updated the `CRC_RESULT_CUR` register with intermediate CRC results for the new data written to the CRC. The processor core should read from the `CRC_RESULT_CUR` register only after detecting `CRC_STAT.IRR` =1. This read-only bit is cleared by CRC hardware and is valid when the `CRC_CTL.IRRSTALL` bit is enabled.	
		0	No status
		1	Intermediate results ready
17 (R/NW)	OBR	Output Buffer Ready. The `CRC_STAT.OBR` bit indicates that the CRC has data ready for the processor core to read. The processor core should read from the CRC only after detecting `CRC_STAT.OBR` =1. This read-only bit is cleared by CRC hardware.	
		0	No status
		1	Output buffer ready
16 (R/NW)	IBR	Input Buffer Ready. The `CRC_STAT.IBR` bit indicates that the CRC is ready to accept a processor core write. The processor core should write to the input register only after detecting that `CRC_STAT.IBR` =1. This read-only bit is cleared by CRC hardware.	
		0	No status
		1	Input buffer ready
4 (R/W1C)	DCNTEXP	Data Count Expired. The `CRC_STAT.DCNTEXP` bit indicates that the `CRC_DCNT` has expired. This W1C bit is not automatically cleared when the CRC is disabled (`CRC_CTL.BLKEN` =0). When the CRC sets this bit on `CRC_DCNT` expiry, the CRC generates the `CRC_INEN.DCNTEXP` interrupt.	
		0	No status
		1	Data counter expired
1 (R/W1C)	CMPERR	Compare Error. The `CRC_STAT.CMPERR` bit indicates that a CRC mismatch or data mismatch has been detected. This W1C bit is not automatically cleared when the CRC is disabled (`CRC_CTL.BLKEN` =0). When the CRC sets this bit on detecting a mismatch, the CRC generates the `CRC_INEN.CMPERR` interrupt. While this bit is set, the `CRC_DCNTCAP` register is disabled from capturing the data count values.	
		0	No status
		1	Compare error

41 Housekeeping ADC (HADC)

The Housekeeping ADC is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters like voltages, temperatures in the system or for any general-purpose use as well.

HADC Features

The HADC supports following features:

- 12-bit ADC core with built-in sample and hold
- ENOB = 10 bit
- 8 input channels which can be extended to 15 channels by putting an external channel multiplexer
- Throughput rates up to 1MSPS
- Single ended operation
- External reference 2.5 V to 3.3 V
- Analog input 0 V to 3.3 V.
- Selectable ADC clock frequency through a pre-scaler
- Conversion type adaptable to each application: allows single or continuous conversion with option of auto-scan
- 16 data registers (individually addressable) to store conversion values
- Auto sequencing capability provides up to 15 *auto-conversions* in a single session. Each conversion can be programmed to select any of the available input channels.

HADC Functional Description

The HADC provides the analog to digital conversion capability for general-purpose housekeeping tasks, such as voltage and temperature monitoring. The core of HADC is a 12-bit SAR ADC, providing multiple analog input channels.

The HADC has the following functionality:

Fixed and continuous conversion modes

ADC converts the input channel sequence for a fixed number of times or continuously converts an input channel sequence.

Auto scanning

All the input channels can be sampled in a sequential manner.

Channel sequence programming

The sequence of a channel can be selected by programming the channel mask register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain.

ADSP-SC58x HADC Register List

The Housekeeping ADC (HADC) provides a general purpose, multi-channel successive approximation A-to-D converter. A set of registers governs HADC operations. For more information on HADC functionality, see the HADC register descriptions.

Table 41-1: ADSP-SC58x HADC Register List

Name	Description
`HADC_CHAN_MSK`	Channel Mask Register
`HADC_CTL`	Control Register
`HADC_DATA[nn]`	Channel Data Registers
`HADC_IMSK`	Interrupt Mask Register
`HADC_STAT`	Status Register

ADSP-SC58x HADC Interrupt List

Table 41-2: ADSP-SC58x HADC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
157	HADC0_EVT	HADC0 Event	Edge	

ADSP-SC58x HADC Trigger List

Table 41-3: ADSP-SC58x HADC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
133	`HADC0_EOC`	HADC0 End of Conversion	Edge

Table 41-4: ADSP-SC58x HADC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

HADC Definitions

The following definitions are helpful when using the HADC module.

Auto-scan

Auto-scan is a feature which allows the multiple channels to be scanned and converted in sequence one after the other.

HADC Wakeup Time

It is the time required by the module after coming out of power down before it can start converting.

Signal-to-Noise and Distortion Ratio (SINAD)

The measured ratio of signal-to-noise and distortion at the output of the ADC. The signal is the rms amplitude of the fundamental. Noise is the sum of all non-fundamental signals up to half the sampling frequency ($f_S/2$), excluding dc. The ratio depends on the number of quantization levels in the digitization process; the more levels, the smaller the quantization noise. The theoretical signal-to-noise and distortion ratio for an ideal N-bit converter with a sine wave input is given by:

Signal-to-(Noise + Distortion) = (6.02 N + 1.76) dB

Total Harmonic Distortion (THD)

The ratio of the rms sum to the harmonics to the fundamental.

Peak Harmonic or Spurious Noise

The ratio of the rms value of the next largest component in the ADC output spectrum (up to $f_S/2$ and excluding dc) to the rms value of the fundamental. Typically, the value of this specification is determined by the largest harmonic in the spectrum, but for ADCs where the harmonics are buried in the noise floor, it is a noise peak.

Integral Nonlinearity

The maximum deviation from a straight line passing through the endpoints of the ADC transfer function. The endpoints are zero scale, a point 1 LSB below the first code transition, and full scale, a point 1 LSB above the last code transition.

Differential Nonlinearity

The difference between the measured and the ideal 1 LSB change between any two adjacent codes in the ADC.

Offset Error

The deviation of the first code transition (00...000) to (00...001) from the ideal—that is, GND1 + 1 LSB.

Offset Error Match

The difference in offset error between any two channels.

Gain Error

The deviation of the last code transition (111...110) to (111...111) from the ideal (that is, REFIN – 1 LSB) after the offset error has been adjusted out.

Gain Error Matching

The difference in gain error between any two channels.

Power Supply Rejection Ratio (PSRR)

PSRR is defined as the ratio of the power in the ADC output at full-scale frequency, f, to the power of a 100 mV p-p sine wave applied to the ADC VDD supply of frequency, f_S. The frequency of the input varies from 5 kHz to 25 MHz. PSRR (dB) = 10 log(Pf/Pf_S) where: Pf is the power at frequency, f, in the ADC output. Pf_S is the power at frequency, f_S, in the ADC output.

HADC Block Diagram

The ***HADC Block Diagram*** figure shows the functional blocks within the HADC and the interface to the processor core and the peripherals. The HADC has an internal 8 channel multiplexer that is controlled by a programmable sequencer which selects the desired channel or sequence of channels. The eight channels of the internal multiplexer can be expanded by providing an external multiplexer, increasing the total number of channels to 15.

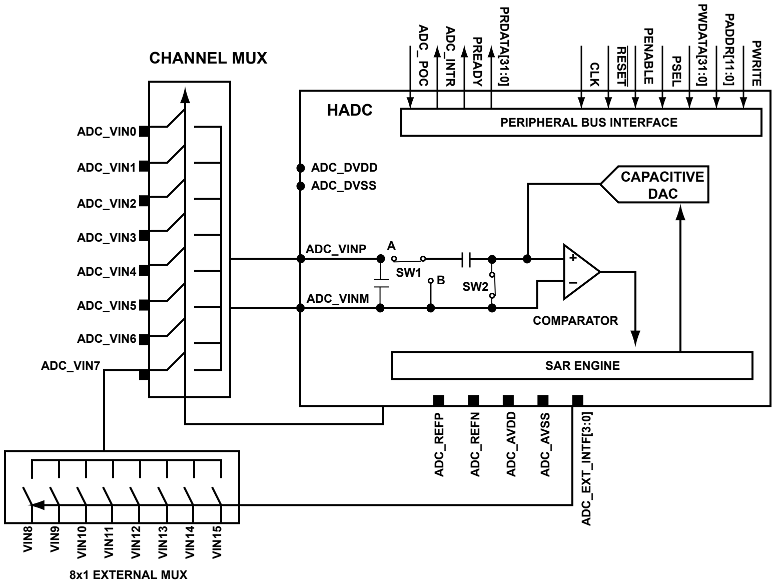

CHANNEL MUX
ADC_VIN0
ADC_VIN1
ADC_VIN2
ADC_VIN3
ADC_VIN4
ADC_VIN5
ADC_VIN6
ADC_VIN7
ADC_POC
ADC_INTR
PREADY
PRDATA[31:0]
CLK
RESET
PENABLE
PSEL
PWDATA[31:0]
PADDR[11:0]
PWRITE
HADC
PERIPHERAL BUS INTERFACE
ADC_DVDD
ADC_DVSS
CAPACITIVE DAC
ADC_VINP
A
SW1
B
SW2
ADC_VINM
+
–
COMPARATOR
SAR ENGINE
ADC_REFP
ADC_REFN
ADC_AVDD
ADC_AVSS
ADC_EXT_INTF[3:0]
VIN8
VIN9
VIN10
VIN11
VIN12
VIN13
VIN14
VIN15
8x1 EXTERNAL MUX

Figure 41-1: HADC Block Diagram

HADC Signal Descriptions

The ***HADC Signal Descriptions*** table provides descriptions of the signals used by the HADC.

Table 41-5: HADC Signal Descriptions

Signal Name	Signal Description
AVDD	ADC I/O supply
AVSS	I/O ground for analog blocks
VREFP	External reference for ADC
VREFN	Ground reference for ADC
VINn	Analog input at channel n

HADC Architectural Concepts

The HADC is based on a 12-bit SAR ADC that provides a simple register-based access model to obtain the results of conversion. The digital front end of the HADC provides a set of registers to configure the mode of operation, sampling frequency, and input channel selection control. The ADC supports multiple input analog channels which can be individually selected or deselected for conversion. The results of each analog channel are stored in a register. The core can access the register to read the conversion results once the conversion is complete. The HADC also provides the interrupts on completion of each channel conversion to avoid polling by the core. The following sections provide more details about the architecture of the HADC.

Converter Operation

The housekeeping ADC is a 12-bit successive approximation ADC based around a capacitive DAC. The ***ADC Acquisition Phase*** figure and the ***ADC Conversion Phase*** figure show simplified schematics of the ADC. The ADC is comprised of control logic, SAR, and a capacitive DAC. The components are used to add and subtract fixed amounts of charge from the sampling capacitor to bring the comparator back into a balanced condition. The ***ADC Conversion Phase*** figure shows the ADC during its acquisition phase. SW2 is closed and SW1 is in Position A. The comparator is held in a balanced condition and the sampling capacitor acquires the signal on the selected V_{IN} channel.

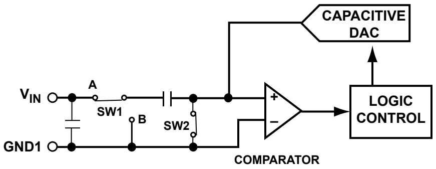


Figure 41-2: ADC Acquisition Phase

When the ADC starts a conversion (see the ***ADC Conversion Phase*** figure), SW2 opens and SW1 moves to Position B, causing the comparator to become unbalanced. The control logic and the capacitive DAC are used to add and subtract fixed amounts of charge to bring the comparator back into a balanced condition. When the comparator is rebalanced, the conversion is complete. The control logic generates the ADC output code.

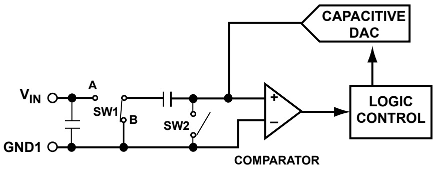


Figure 41-3: ADC Conversion Phase

Auto-Scan

The HADC features auto-scan mode where all the input channels can be sampled in a sequential manner. The number of channels enabled in auto-scan mode can be selected by programming the `HADC_CHAN_MSK` register. If the bit corresponding to the channel is set high, that particular channel is masked, and is not included in the auto-scan chain. In this way programs can sample all, none, or a selected set of channels by writing a high or a low for the individual channel. Auto sequencing allows the system to convert the same channel multiple times, allowing programs to perform oversampling algorithms.

For example, if the `HADC_CHAN_MSK` register bits [3:0] are set to 1101, then channel 0, channel 2 and channel 3 are not included in the auto-scan chain. Whether the conversion is a single or fixed number or continuous depends on the status of `HADC_CTL.CONT` bit. If this bit is low, the `HADC_CTL.FIXEDCNV` bits determine the number of sequence conversions.

The maximum number of fixed sequence conversions is 15. By default, the first eight channels of the HADC are enabled and extended channels are masked. An extended channel is the increased number of channels when a single channel of ADC is externally multiplexed, effectively increasing the total number of available channels. The total number of channels supported is 15. The program must configure the `HADC_CHAN_MSK` register to enable any desired channel.

Channel Sequence Programming

The sequence of a channel can be selected by programming the `HADC_CHAN_MSK` register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain. If the program must get the conversion results from a particular channel, then the bit corresponding to that channel should be zero. Channels 8-15 are virtual channels, in that they can connect an external multiplexer and increase the effective number of channels. As the output of the external multiplexer is routed to channel 7, this channel is not available (or converted) separately when the virtual channels are not masked in the `HADC_CHAN_MSK` register. The auto-scan section has more details.

ADC Transfer Function

The output coding is straight binary for the analog input channel conversion. The designed code transitions occur at successive LSB values (that is, 1 LSB, 2 LSBs, and so forth). The LSB size is $V_{REF}/4096$ for the HADC. The *ADC Transfer Function* figure shows the ideal transfer characteristic for straight binary coding.

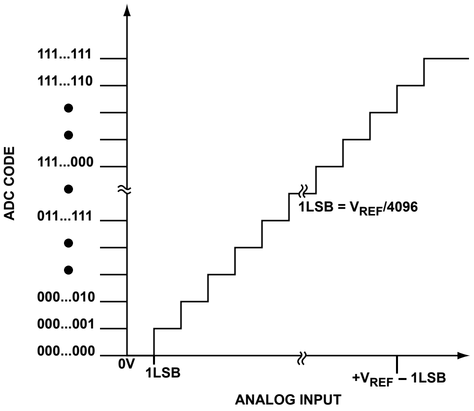


Figure 41-4: ADC Transfer Function

Results

The HADC takes 20 cycles of f_{SAMPLE} for one channel conversion. (The value of the `HADC_CTL.FDIV` bit field determines f_{SAMPLE}). The time taken to complete one sequence depends on the number of channels in the auto-scan chain. There is a latency of 1 cycle from the time the channel is selected internally to sample to the time the data is ready. After the end of each channel conversion, the data is written into the corresponding data register. An interrupt is generated (if the interrupt mask is not enabled) to signal that the data is ready for that particular channel.

HADC Operating Modes

The HADC has two modes of operation described in the following sections.

Fixed Conversion Mode

In this mode, the ADC converts the input channel sequence for a fixed number of times. The frequency is configured in the `HADC_CTL.FIXEDCNV` bit field. To use this mode, clear the `HADC_CTL.CONT` bit.

Continuous Conversion Mode

In this mode ADC continuously converts an input channel sequence, as long as `HADC_CTL.STARTCNV` bit is held high. To use this mode, set the `HADC_CTL.CONT` bit.

HADC Event Control

The HADC generates different events depending on the state of the ADC and the status of channel conversions. It can generate an event for each of the following conditions:

- When ADC is ready for conversion
- At the end of sequence conversion
- At the end of each individual channel conversion

Each of these events can generate an interrupt. To generate an interrupt on any desired event, clear the respective bit in the `HADC_IMSK` register.

HADC Programming Model

Following sections provide some guidelines for HADC programming.

Powering Up the HADC

To power-up the HADC, program the following bits in the `HADC_CTL` register.

- Deassert the `HADC_CTL.PD` bit (HADC power down)
- Set the `HADC_CTL.NRST` bit (Reset)
- Set the `HADC_CTL.ENLS` bit (Enable level shifters)

After deasserting `HADC_CTL.PD`, the HADC requires a finite wake-up time (t_{WAKEUP}) before it can start converting. The HADC requires only two f_{SAMPLE} clocks from the assertion of the `HADC_CTL.NRST` bit before the module is ready to convert. (`HADC_CTL.PD` is low). Poll the `HADC_STAT.RDY` bit. A 1 on this bit indicates that the HADC is ready to convert data.

Enabling the HADC

Setting the `HADC_CTL.STARTCNV` bit enables the HADC. When this bit is kept high, the HADC can work in either continuous or fixed conversion mode. After the `HADC_CTL.STARTCNV` bit is set =1, the `HADC_CHAN_MSK` can still be re-programmed, but the new sequence only comes into effect after the current sequence conversion is complete.

ADSP-SC58x HADC Register Descriptions

Housekeeping ADC (HADC) contains the following registers.

Table 41-6: ADSP-SC58x HADC Register List

Name	Description
`HADC_CHAN_MSK`	Channel Mask Register
`HADC_CTL`	Control Register
`HADC_DATA[nn]`	Channel Data Registers
`HADC_IMSK`	Interrupt Mask Register
`HADC_STAT`	Status Register

Channel Mask Register

The `HADC_CHAN_MSK` register provides bits that mask each channel. The first MSB corresponds to channel 15, the second MSB to channel 14 and so on. If the mask is set for a particular channel, that channel is not converted. By default, channels 0-7 are not masked and the extended channels 8-15 are masked.

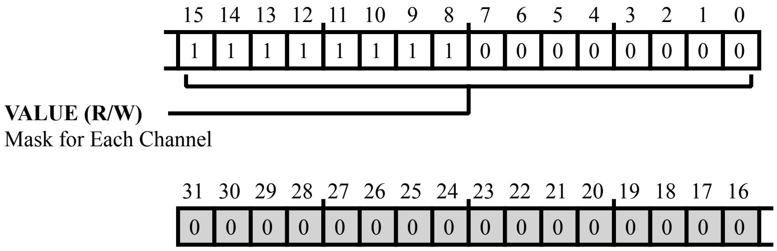

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
VALUE (R/W)
Mask for Each Channel
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 41-5: HADC_CHAN_MSK Register Diagram

Table 41-7: HADC_CHAN_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Mask for Each Channel. The `HADC_CHAN_MSK.VALUE` bit field is the mask bit for each channel. The first MSB corresponds to channel 15, the second MSB to channel 14 and so on. If the mask is set for a particular channel, that channel is not converted. By default, channels 0-7 are not masked and the extended channels 8-15 are masked.

Control Register

The HADC_CTL register contains control bits that configure various module settings start or reset the module.

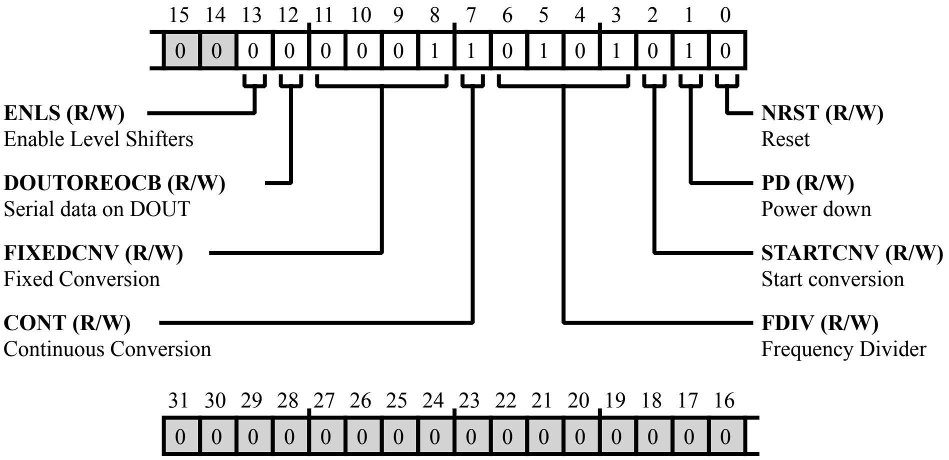

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0
ENLS (R/W)
Enable Level Shifters
DOUTOREOCB (R/W)
Serial data on DOUT
FIXEDCNV (R/W)
Fixed Conversion
CONT (R/W)
Continuous Conversion
NRST (R/W)
Reset
PD (R/W)
Power down
STARTCNV (R/W)
Start conversion
FDIV (R/W)
Frequency Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 41-6: HADC_CTL Register Diagram

Table 41-8: HADC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	ENLS	Enable Level Shifters. Setting the `HADC_CTL.ENLS` bit enables the level shifters, allowing the HADC analog side (which works in the VDD_EXT domain) to work with the digital core and interface is (in the VDD_INT domain).	
		0	Disable down level shifters
		1	Enable down level shifters
12 (R/W)	DOUTOREOCB	Serial data on DOUT. If the `HADC_CTL.DOUTOREOCB` bit =1, serial data arrives on the EOC_DOUT pin. If this bit =0 (default) it acts as an EOC only if the external multiplexer is connected.	
		0	Reserved
		1	Reserved
11:8 (R/W)	FIXEDCNV	Fixed Conversion. The `HADC_CTL.FIXEDCNV` bit configures the number of conversions = FIXEDCNV. This value determines how many times a sequence is converted when the HADC is in fixed conversion mode. This only applies when the `HADC_CTL.CONT` bit =0. Before changing the `HADC_CTL.FIXEDCNV` bit, clear the `HADC_CTL.NRST` bit (=0).	

Table 41-8: HADC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CONT	Continuous Conversion. When the `HADC_CTL.CONT` bit =0, the ADC converts a sequence for a fixed number of times. This number is configured using the `HADC_CTL.FIXEDCNV` bit field. When the `HADC_CTL.CONT` bit =1, the ADC continuously converts a given sequence, provided the `HADC_CTL.STARTCNV` is held high.	
		0	ADC converts sequence for fixed number of times
		1	ADC continuously converts given sequence
6:3 (R/W)	FDIV	Frequency Divider. The `HADC_CTL.FDIV` bit field configures f_{SAMPLE}=f_{CLK}=/(FDIV+1). Select f_{CLK} and `HADC_CTL.FDIV` values so that f_{SAMPLE} is in the range of 50 kHz to 22.5 MHz. The minimum value for `HADC_CTL.FDIV` is 1. Before changing the `HADC_CTL.FDIV` bits, clear the `HADC_CTL.NRST` bit.	
2 (R/W)	STARTCNV	Start conversion. The `HADC_CTL.STARTCNV` bit needs to be set for the ADC to start converting data. If the ADC is running in non continuous mode, it is reset by hardware after the desired number of conversions is completed.	
		0	No action
		1	Start converting
1 (R/W)	PD	Power down. The `HADC_CTL.PD` bit powers down the analog circuitry of the ADC. After this bit returns to 0 a finite power-up time is required before the ADC can start converting data.	
		0	No action
		1	Power down the analog circuitry of the ADC
0 (R/W)	NRST	Reset. The `HADC_CTL.NRST` bit resets the ADC.	
		0	Reset the ADC
		1	No action

Channel Data Registers

The `HADC_DATA[nn]` registers N ranges from 0-15. Each corresponding to an ADC channel. ADC_DATA_0 corresponds to channel 0, ADC_DATA_1 to channel 1 and so on.

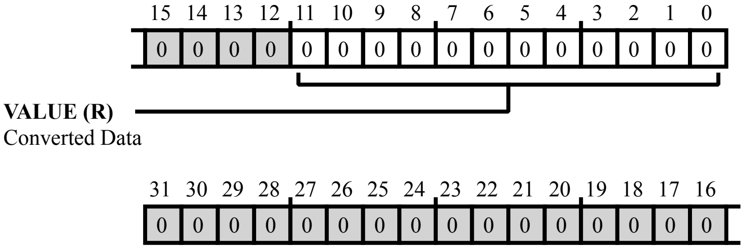

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Converted Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 41-7: HADC_DATA[nn] Register Diagram

Table 41-9: HADC_DATA[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/NW)	VALUE	Converted Data. The `HADC_DATA[nn].VALUE` bit field contains the digital code for the sampled analog value. Each channel has its own data register.

Interrupt Mask Register

The HADC_IMSK register masks (disables) or unmasks (enables) the interrupts as programmed. The reset value of the HADC_IMSK register is 0x08, masking these interrupts after reset.

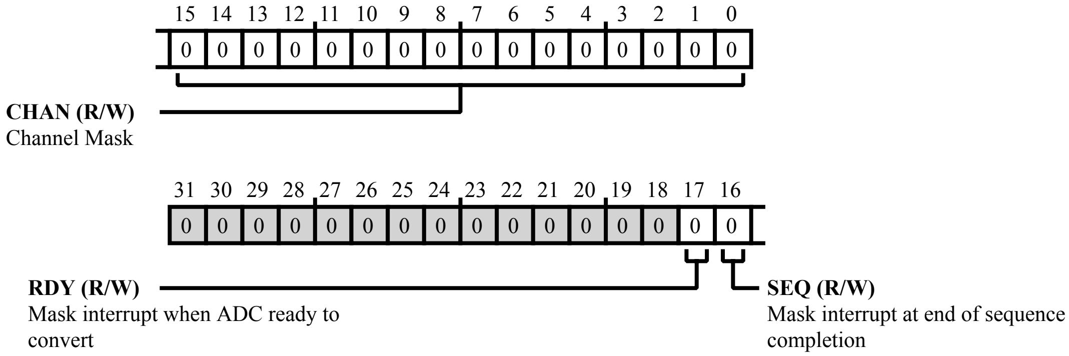

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHAN (R/W)
Channel Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDY (R/W)
Mask interrupt when ADC ready to convert
SEQ (R/W)
Mask interrupt at end of sequence completion

Figure 41-8: HADC_IMSK Register Diagram

Table 41-10: HADC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	RDY	Mask interrupt when ADC ready to convert. The HADC_IMSK.RDY bit masks the interrupt generated when ADC is ready to convert.	
16 (R/W)	SEQ	Mask interrupt at end of sequence completion. The HADC_IMSK.SEQ bit masks the interrupt which is generated at the end of sequence completion.	
		0	Interrupt is unmasked
		1	Interrupt is masked
15:0 (R/W)	CHAN	Channel Mask. The HADC_IMSK.CHAN bit field provides the interrupt mask bit for each channel. N ranges from 0-15. The MSB corresponds to channel 15, the second MSB to channel 14 and so on. If the interrupt mask for a particular channel is high, no interrupt is generated when that channel finishes the end of data conversion.	

Status Register

The `HADC_STAT` register contains bits that provide status information on the HADC module.

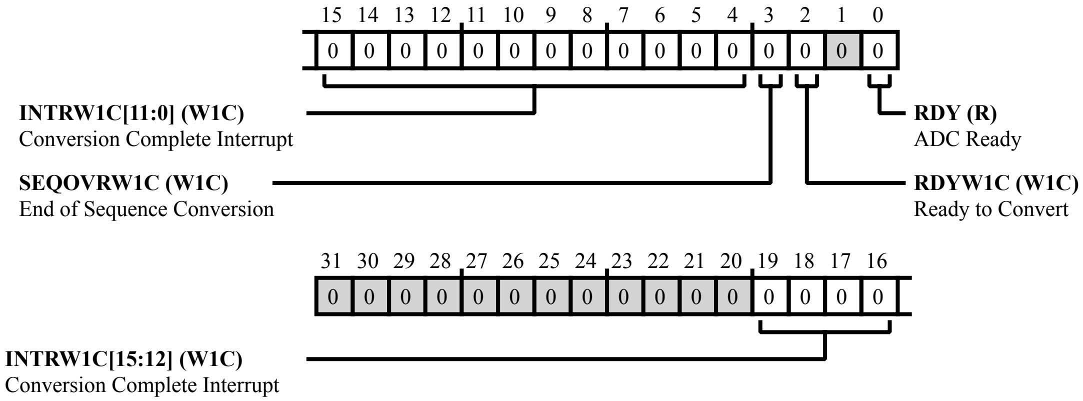

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INTRW1C[11:0] (W1C)
Conversion Complete Interrupt
SEQOVRW1C (W1C)
End of Sequence Conversion
RDY (R)
ADC Ready
RDYW1C (W1C)
Ready to Convert
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INTRW1C[15:12] (W1C)
Conversion Complete Interrupt

Figure 41-9: HADC_STAT Register Diagram

Table 41-11: HADC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:4 (RX/W1C)	INTRW1C	Conversion Complete Interrupt. The `HADC_STAT.INTRW1C` bit field indicates when the corresponding ADC channel completes conversion. N ranges from 0-15 and the MSB corresponds to channel 15, the second MSB to channel 14 and so on. These bits are sticky and are W1C.
3 (RX/W1C)	SEQOVRW1C	End of Sequence Conversion. The `HADC_STAT.SEQOVRW1C` bit indicates the end of a sequence conversion and is a sticky status bit which is W1C.
2 (RX/W1C)	RDYW1C	Ready to Convert. The `HADC_STAT.RDYW1C` bit is the stick version of the `HADC_STAT.RDY` bit.
0 (R/NW)	RDY	ADC Ready. The `HADC_STAT.RDY` bit is set (=1) when the ADC is ready to convert data.

42 System Security

The requirement to protect content, keys, IP and other sensitive information have become increasingly prevalent. The processor contains several modules and system elements that contribute to creating a secure operating environment for trusted code to execute. The modules and system elements that can be used are:

- Boot Kernel
- Secure Core
- System Protection Unit (SPU)
- System Memory Protection Unit (SMPU)
- DEBUG through the Test Access Port Controller (TAPC)
- One-Time Programmable (OTP) memory
- Cryptographic Accelerators (optionally)

NOTE: This product includes security features that can be used to protect embedded non-volatile memory contents and prevent execution of non-authorized code. When security is enabled on this device (either by the ordering party or the subsequent receiving parties), Analog Devices' ability to conduct Failure Analysis on returned devices will be limited. Please contact Analog Devices, Inc. for details on the Failure Analysis limitations for this device.

Security Features

The security infrastructure items in the system provide the following features:

- Secure operating environment for secure code execution
- Protect sensitive IP from theft by malicious users or competitors
- Protect sensitive data (for example cipher keys)
- Allow debugging while still maintaining security

Security Functional Description

In order to provide secure operating environment in a system, it requires the involvement of multiple elements.

Boot Kernel

The Boot Kernel is the root of trust for a secure system. Since the boot kernel is developed by Analog Devices, Inc and is stored in ROM and can't be changed, it can be trusted. The boot kernel validates the authenticity of the application binary image that needs to be booted in. It also handles decrypting the binary image if it's encrypted.

Verifying the authenticity of the application binary asserts that

1. The image hasn't been tampered with or altered
2. The image came from a trusted developer

If the image is encrypted, it ensures confidentiality since the boot image is stored on an external storage device and can be more easily read or stolen than in the part.

Once the boot kernel can verify and optionally decrypt the boot image, it can be loaded into the processor for execution. At this point, the *chain of trust* is continued with the verified application.

Secure Booting

Secure booting is when the boot kernel uses cryptographic algorithms to perform checks on the application binary and to decrypt it. Secure booting is done when security is enabled. If security is not enabled the boot kernel does not verify any signatures nor does it perform any decryption on the application binary. See Security Mode Configuration.

Secure Core

In a system with multiple master and slave resources, not everything is considered secure. There are secure and non-secure peripherals and secure and non-secure segments of memory.

For a system to be considerd secure, a secure core that can access and execute instructions (verified application) from secure memory must be configured. Typically, in a single core processor, either the core is hardwired to be secure or the core can switch between secure and non-secure modes.

System Protection Unit (SPU)

The SPU in the system serves two functions. First, it acts as a gatekeeper, guarding against non-secure accesses to secure resources (peripherals). Second, it is used to define which resources in the system are secure or non-secure masters and which resources in the system are secure or non-secure slaves.

NOTE: Even thought the SPU can be configured by secure or non-secure master, the first steps that the verified secure application should perform are:

1. Configure the SPU as a secure slave itself so only other secure masters can configure it

2. Define and configure the secure masters and slaves using the SPU

This way, once the SPU is secure and other secure masters and slaves are configured, non-secure masters cannot tamper with the security privileges of secure masters and slaves nor can non-secure resources be changed to a secure resource.

System Memory Protection Unit (SMPU)

Similar to the SPU protecting the MMR address range for peripherals, the SMPU can guard memory ranges or pages. Memory pages can be configured as secure or non-secure. Again, like in the case of the SPU, the SMPU can guard against non-secure transaction attempts to secure memory.

NOTE: A verified application should reside in memory configured as secure. By default the SMPU has all memory configured as secure. If the program needs to update the memory protections via the SMPU, the application and it's data should remain in secure memory.

Debug

The typical way of accessing a system is through the debug port via a JTAG or serial wire interface. If these access points are not secure sensitive IP such as cipher keys can be exposed and code can be changed to disable other security settings. To guard against this type of attack or security hole and still provide debugging capabilities for a developer, a debug unit with security features is used.

When the developer first receives the part, they can define a JTAG/DEBUG key that is programmed into OTP memory. Once security is enabled, the debug unit compares the key sent from the host debugger with the key inside the system. If a match occurs, debug access to secure resources are allowed.

One-Time-Programmable (OTP) Memory

Customer programmable OTP memory is used to safely and securely store sensitive information such as cipher keys.

In a public key algorithm (for example ECDSA which is used in secure boot), the public key is used to verify the digital signature that accompanies the application binary. The private key is used by the developer on the host development machine to create the digital signature. The public and private key pair is unique and the public key needs to be stored in non-volatile, one-time-programmable memory. If the public key is allowed to be changed than users can generate their own public/private key pairs and successfully boot in malicious code. This can either re-purpose the part or change security configurations to allow easier access to sensitive information stored elsewhere in OTP memory.

When security is not enabled, OTP can be accessed freely. When a user decides to set the LOCK bit in OTP memory to enable security, portions of the OTP, specifically the first 6K bits and another 1K bits for the customer boot information are will also be locked. From then on, only secure masters will be allowed to access those memory regions.

Cryptographic Accelerators

Cryptographic algorithms are mathematical tools to help provide security. Hardware engines provide some advantages but are not necessarily required. For computationally expensive operations like those used in Elliptic Curve Cryptography, like ECDSA used in secure boot, the operations can be accelerated while the core performs other tasks. Also, it's less likely that the hardware engine can be hacked to change the results.

Security Mode Configuration

Because the processor is unsecure by default, if security is not required then no steps need to be done.

To enable security features, use the following procedure.

- Generate the public/private key pair on the host development machine.[1]
- Program OTP memory with the public key.[2]
- Program OTP memory with the decryption key if the application binary needs to be encrypted.[2]
- Program OTP memory with the debug/JTAG key.[2]
- Develop the application and sign it, creating the digital signature with the private key.[1]
- If confidentiality is required, encrypt the application binary before signing it.
- Set the LOCK bit in OTP memory to enable security. After this, subsequent boots are secure boots.[2, 3]

[1] Software tools are provided with developments tools to generate keys, sign boot streams and also perform encryption. Refer to the development tools manuals for information on usage.

[2] Refer to the OTP Chapter for programming the OTP and other related information.

[3] Refer to the Booting Chapter for more information on Secure Booting and other related information.

Status and Error Signals

In a fully functional secure system, non-secure resources should not even attempt to access a secure resource. If this does occur, then either the code has been altered or replaced with malicious code or the system contains a bug.

Errors or error events are dependant on the configuration of the SPU and SMPU, the protection units which guard against security violations. In the case of the SMPU, the error can simply be captured, captured and interrupt generated, or the access prevented without capturing any error. It is the developer's responsibility to:

1. Determine if there was an error due to a blocked access.
2. Determine how to handle a blocked access (for example fix the bug (offline), or try to use a different resource (run time)).

43 System Protection Unit (SPU)

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The SPU restricts access to certain MMRs, similar to the functionality of a semaphore.

The SPU also protects peripherals based on security settings. It is part of the overall security infrastructure of the processor.

SPU Features

The SPU has the following features:

- Write-protect system MMR from certain system masters and core masters.
- Simultaneously lock multiple peripheral configuration registers through a global lock mechanism.
- Write-protect and block access to its own write-protection registers from other system masters.
- Defined security privileges to peripherals and system resources.
- Security protection to guard secure peripheral MMRs against non-secure accesses.

SPU Functional Description

The following sections provide information on the function of the SPU.

ADSP-SC58x SPU Register List

The System Protection Unit (SPU) provides a set of registers that can protect system resources from errant writes. The protection categories are global lock (protects configuration registers) and write protect register lock (protects the write protect register). For more information on SPU functionality, see the SPU register descriptions.

Table 43-1: ADSP-SC58x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_SECURECHK	Secure Check Register

Table 43-1: ADSP-SC58x SPU Register List (Continued)

Name	Description
`SPU_SECURECTL`	Secure Control Register
`SPU_SECUREC[n]`	Secure Core Registers
`SPU_SECUREP[n]`	Secure Peripheral Register
`SPU_STAT`	Status Register
`SPU_WP[n]`	Write Protect Register n

ADSP-SC58x SPU Interrupt List

Table 43-2: ADSP-SC58x SPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
241	SPU0_INT	SPU0 Interrupt	Level	

Peripheral Register Write Protection

The SPU has a write-protection register (`SPU_WP[n]`) associated with each peripheral. Each of these write-protection registers has the exact same bits that correspond to a particular SMMR master (Core 0, Core 1, MDMA, for example). When the bits are set, the SPU locks the corresponding SMMR masters from accessing the register address space of the associated peripheral. The bits in the register can be cleared to allow access to the registers of the peripheral again. When the SPU intiates the write-protection register, any writes that are in-progress complete before the SPU blocks subsequent writes.

In the ***SPU Write Protect Registers*** figure, each write-protect register in the SPU is associated with a particular peripheral.

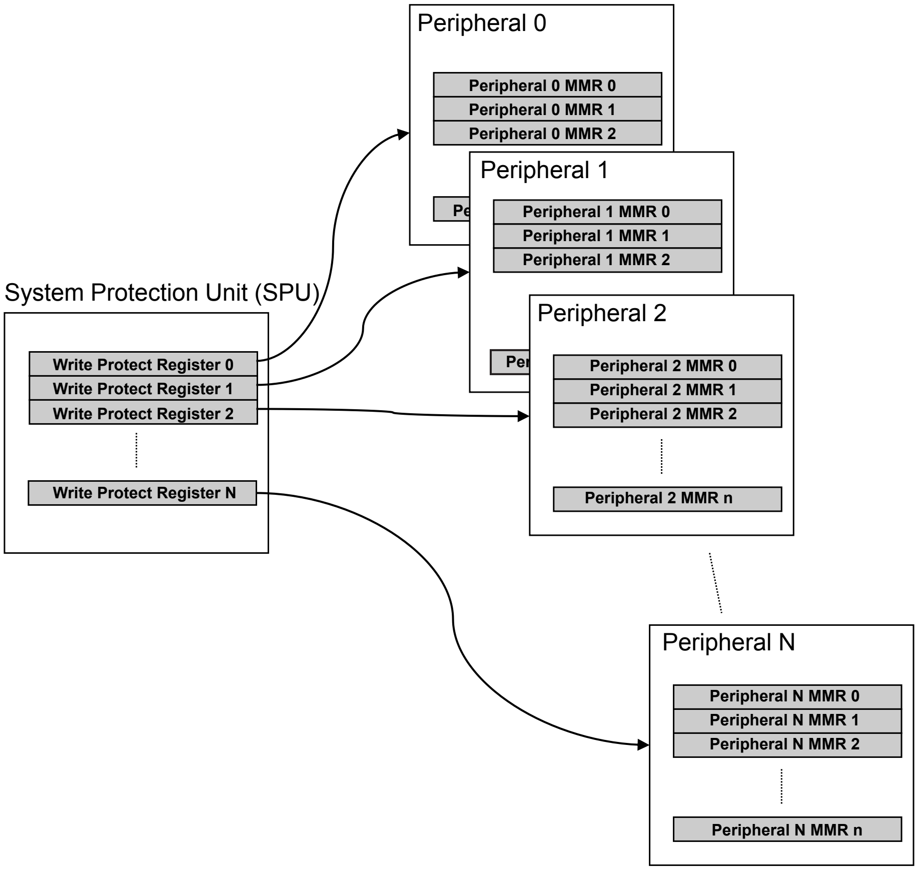

Peripheral 0
Peripheral 0 MMR 0
Peripheral 0 MMR 1
Peripheral 0 MMR 2
Pe
Peripheral 1
Peripheral 1 MMR 0
Peripheral 1 MMR 1
Peripheral 1 MMR 2
Per
Peripheral 2
Peripheral 2 MMR 0
Peripheral 2 MMR 1
Peripheral 2 MMR 2
Peripheral 2 MMR n
System Protection Unit (SPU)
Write Protect Register 0
Write Protect Register 1
Write Protect Register 2
Write Protect Register N
Peripheral N
Peripheral N MMR 0
Peripheral N MMR 1
Peripheral N MMR 2
Peripheral N MMR n

Figure 43-1: SPU Write Protect Registers

In the figure, a write-protect register in the SPU module blocks write-attempts to the MMR space of the associated peripheral. The bits in the write-protect register specify from which masters to block write-access.

NOTE: A SPU write protection register (`SPU_WP[n]`) exists for the SPU alone. If all defined bits are set in this register for the SPU, any configurations in the SPU are locked and cannot be changed. Only a system reset can restore access to the SPU.

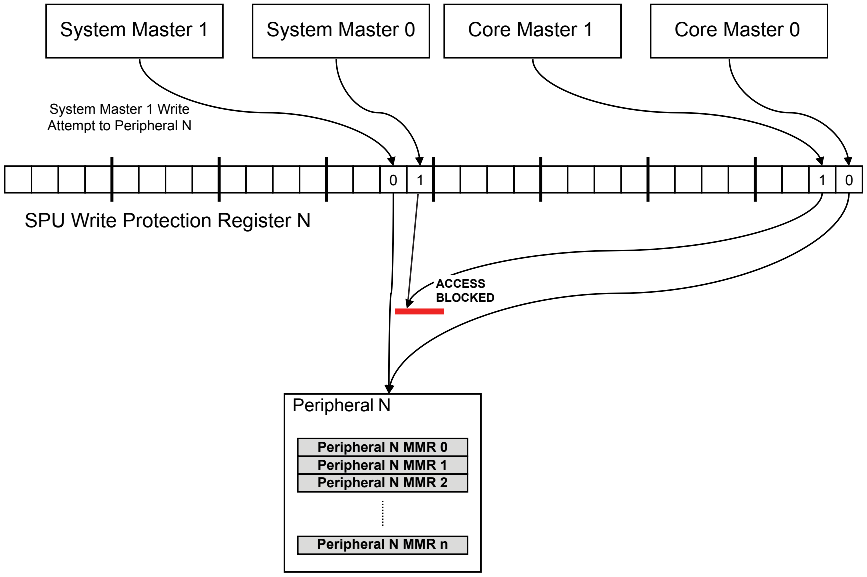

System Master 1
System Master 0
Core Master 1
Core Master 0
System Master 1 Write Attempt to Peripheral N
0
1
1
0
SPU Write Protection Register N
ACCESS BLOCKED
Peripheral N
Peripheral N MMR 0
Peripheral N MMR 1
Peripheral N MMR 2
Peripheral N MMR n

Figure 43-2: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

Global Locking

The SPU also has global locking capability. When enabled by setting `SPU_CTL.GLCK` bit field to a value other than 0xAD, a system-wide global lock signal is active. Some peripherals have a lock enable bit in their control register. When this bit is set, the peripheral recognizes the global lock signal and blocks further write-accesses to its own control register. Access to the configuration register of the peripheral is enabled when the global lock is turned off in the SPU.

The ***Global Locking*** figure is a conceptual diagram. The diagram shows how the SPU module (or any peripheral) blocks any write attempts to its control register when:

- The global lock signal from the SPU is active, and
- The global lock enable bit is set in the control register of the peripheral

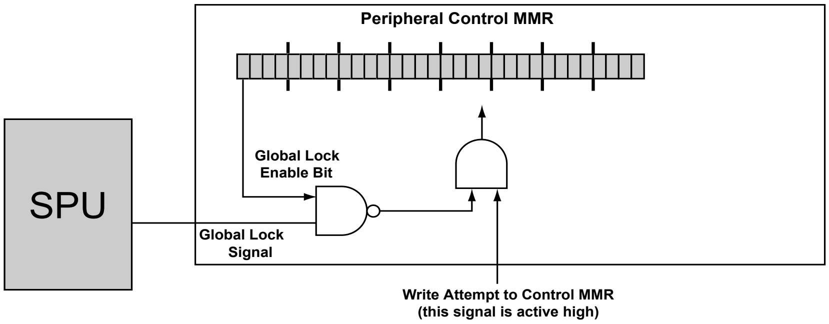

Peripheral Control MMR
SPU
Global Lock Enable Bit
Global Lock Signal
Write Attempt to Control MMR (this signal is active high)

Figure 43-3: Global Locking

The SPU can write-protect its own registers. When the `SPU_CTL.WPLCK` bit is set and global locking is enabled, the SPU blocks accesses to the SPU write-protection registers. To enable write access to the write-protection registers in the SPU, disable the global locking.

SPU Block Diagram

The *SPU System-Level Block Diagram* shows a system-level block diagram of where the SPU is located in the system. It resides between the SMMR interface and the system crossbar. Depending on the configuration of the SPU write-protect registers, it can block access to some peripherals from certain SMMR masters.

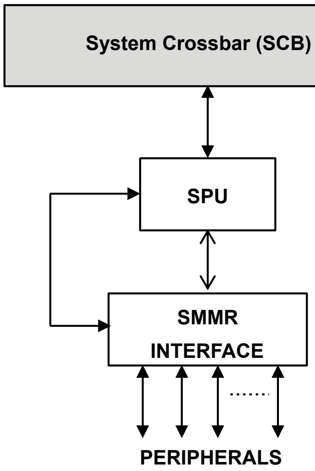


Figure 43-4: SPU System-Level Block Diagram

SPU Architectural Concepts

As shown in the block diagram, the SPU sits between the system crossbar (SCB) and the SMMR interface to the peripherals. The SPU gates any MMR access to any peripheral from any master that comes through the SCB. Depending on the configuration of the write-protection registers in the SPU, the SPU does or does not allow the MMR write to go through.

The SPU also checks whether the transaction is a secure or non-secure transaction and blocks it according to the configured security setting for the target destination. A secure master can generate secure read or secure write transactions which can access secure or non-secure slaves. A non-secure master can generate non-secure read or non-secure write transactions and can only access non-secure slaves.

SPU Event Control

The system protection unit provides write-protection against MMRs peripherals and its own write-protect registers. If a write attempt is made to any locked MMR peripheral the SPU has write-protected, it blocks the write. The SPU generates a bus error to the master that attempted the write. That master does or does not generate an event, based on the returned error.

The SPU can be configured to generate an interrupt for the write-protection violation by setting the `SPU_CTL.PINTEN` bit. The SPU can also be configured to generate an interrupt for a security violation by setting the `SPU_SECURECTL.SINTEN` bit. If either one or both bits is triggered, the `SPU_STAT.VIRQ` bit is set.

The SPU can also lock its own registers from write attempts. If a write-attempt is made to a locked register in the SPU, the SPU blocks it and records it as an error in the `SPU_STAT.LWERR` bit. Again, the SPU generates a bus error to the master that attempted the write.

The master does or does not generate an event, based on the returned error.

The SPU does not generate an event for a blocked write access to an SPU register. If the `SPU_CTL.PINTEN` bit is set, the SPU triggers an interrupt for this blocked access attempt.

The global lock is enabled by setting the `SPU_CTL.GLCK` bit to something other than 0xAD. If the lock bit is set in that same configuration register, a peripheral can block write access to its configuration register. When the SPU blocks a write attempt, the peripheral logs and reports the failed attempt. The SPU is unaware and therefore does not provide any indication of a failed write attempt to the configuration register of the peripheral.

SPU Programming Model

The system protection unit (SPU) consists of write-protect registers. Each one corresponds to a different peripheral instance. Bits in the write-protect registers correspond to system masters that can modify the MMR contents of the peripherals. By writing to these write-protect registers, the corresponding memory-mapped registers of the peripheral are write-protected against masters whose bits in the write-protect register are set.

The SPU globally locks the control register of the peripheral. Peripherals that support this feature have a lock enable bit in their control register. The peripheral blocks any additional write attempts to its control register from any master when:

- The global lock signal is active from the SPU, and
- The lock enable bit of the peripheral is set

If the lock enable bit of a peripheral is not set and the global lock signal is active, access to that control register of the peripheral is still allowed. To grant access again, disable the global lock signal from the SPU by writing the value 0xAD into the `SPU_CTL.GLCK` bit field.

Another protection mechanism that the SPU offers is write-protection against the write-protection registers. If the write protect register lock bit (`SPU_CTL.WPLCK`) is set and the global lock signal is active, writes to the write-protect registers of the SPU are blocked. To reenable access to the write-protect registers in the SPU, deactivate the global lock signal by writing 0xAD into the `SPU_CTL.GLCK` bit field.

For security, the SPU provides a set of `SPU_SECUREC[n]` registers (one for each processor core from Analog Devices) to configure their security settings. The SPU also provides a set of `SPU_SECUREP[n]` registers (one for each peripheral instance) to configure their security settings.

Enabling and Disabling the SPU

The SPU is always operating. There are no bits to enable or disable the SPU. The SPU configuration can be updated at any time. Any ongoing transactions finish before a new configuration is in effect. By default, the SPU does not write-protect any of the MMRs.

Write-Protecting the SPU

The SPU is treated like any other peripheral in the system. As such, the SPU also has an associated write-protection register. If this write-protection register is configured to block all writes from all masters, any SPU configuration remains the same until the next system reset.

Checking the Security State

In some cases while running a peripheral, an application system master does not know whether they are a secure master generating secure transactions or not. The SPU provides a means for checking the security state of the master through the `SPU_SECURECHK` register. When read by a secure master, the register reads 0xFFFFFFFF and when read by a non-secure master, the value is 0x00000000.

SPU Mode Configuration

The SPU can provide address range-wide protection by write-protecting the peripherals MMR address range from system MMR masters. It can also provide register wide protection using global locking. Peripherals that support this feature can enable it in their respective configuration register. When the SPU enables the global lock signal, all subsequent writes to the configuration register of the peripheral are blocked until the global lock signal is deasserted. Similarly, the write-protection registers of the SPU can be write-protected using the global lock signal as well. The SPU uses all these modes of operation together.

Locking Write-Protect Registers

Use the following steps to lock (write-protect) a register.

1. Set the `SPU_CTL.WPLCK` bit and configure the `SPU_CTL.GLCK` field to something other than 0xAD.

The SPU write-protect registers are blocked from further write accesses.

Protecting a Peripheral

Use the following procedure to protect a peripheral.

1. Determine which peripheral needs protection and locate the corresponding write-protect register (`SPU_WP[n]`) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. Determine the SMMR masters from which the peripheral needs protection. Then, set the corresponding bit or bits in the write-protect register (`SPU_WP[n]`) for the peripheral. See the "Write-Protect and Secure Peripheral Registers" section.

After setting the write-protect register for the particular peripheral, the identified SMMR masters are blocked from writing to any MMR in the address space of the peripheral. This block remains in place until the bits in the write-protect register are cleared.

Configuring Security Privileges of a Peripheral

Use the following procedure to configure the security privileges of a peripheral.

1. Determine the peripheral and its corresponding secure peripheral register (`SPU_SECUREP[n]`) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. If the peripheral is to be a secure slave only accepting secure transactions, set bit 0 (`SPU_SECUREP[n].SSEC`).
3. If the peripheral is to be a secure master that generates secure transactions (keeping in mind not all peripherals can be masters), set bit 1 (`SPU_SECUREP[n].MSEC`).

This procedure sets the security privileges of a peripheral.

> **NOTE:** Only a secure master can set security privileges, keeping the chain of trust intact. If a non-secure master configures the security privileges, it can undermine security protection.

ADSP-SC58x SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 43-3: ADSP-SC58x SPU Register List

Name	Description
`SPU_CTL`	Control Register
`SPU_SECURECHK`	Secure Check Register
`SPU_SECURECTL`	Secure Control Register
`SPU_SECUREC[n]`	Secure Core Registers
`SPU_SECUREP[n]`	Secure Peripheral Register
`SPU_STAT`	Status Register
`SPU_WP[n]`	Write Protect Register n

Control Register

The SPU control register (SPU_CTL) provides a global lock for configuration registers as well as control for locking the write protect (SPU_WP[n]) registers. It also controls the generation of an interrupt to report blocked accesses.

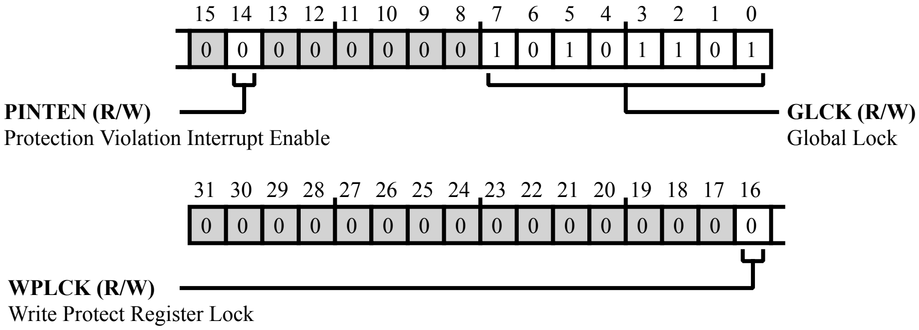

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1
PINTEN (R/W)
Protection Violation Interrupt Enable
GLCK (R/W)
Global Lock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WPLCK (R/W)
Write Protect Register Lock

Figure 43-5: SPU_CTL Register Diagram

Table 43-4: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	WPLCK	Write Protect Register Lock. When the SPU_CTL.WPLCK bit is set in combination with the SPU_CTL.GLCK bit, writes to the SPU's write protect registers are blocked and return an error.	
		0	Disable
		1	Enable
14 (R/W)	PINTEN	Protection Violation Interrupt Enable. When the SPU_CTL.PINTEN bit is set (=1), a block of any transaction according to the configured settings produces an interrupt.	
		0	Disable
		1	Enable
7:0 (R/W)	GLCK	Global Lock. The SPU_CTL.GLCK controls the global lock signal. The global lock signal provides register-based write protection. Writing 0xAD to this field disables the lock, and writing any other value enables the lock.	

Secure Check Register

The `SPU_SECURECHK` register reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

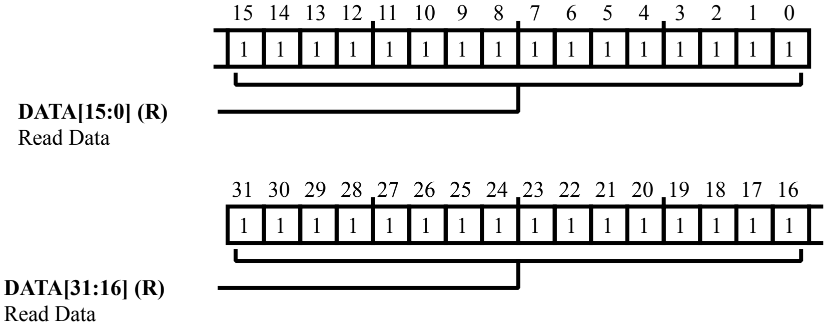

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
DATA[15:0] (R)
Read Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
DATA[31:16] (R)
Read Data

Figure 43-6: SPU_SECURECHK Register Diagram

Table 43-5: SPU_SECURECHK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Read Data. The `SPU_SECURECHK.DATA` bit field performs reads. Reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

Secure Control Register

The SPU Secure Control Register (`SPU_SECURECTL`) allows the user to lock write access to all the `SPU_SECUREC[n]` and `SPU_SECUREP[n]` registers as well as configure the interrupt generation in an event of a security error. It also allows bulk clear of the SSEC bits and/or MSEC bits in the `SPU_SECUREP[n]` registers.

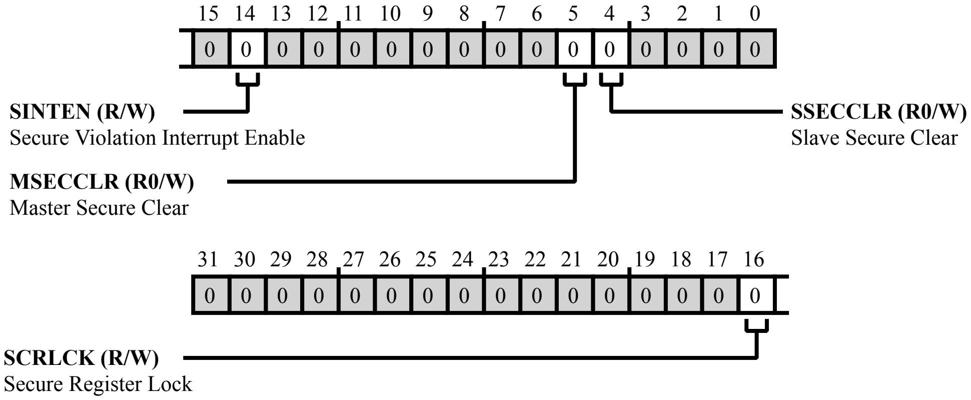

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SINTEN (R/W)
Secure Violation Interrupt Enable
MSECCLR (R0/W)
Master Secure Clear
SSECCLR (R0/W)
Slave Secure Clear
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCRLCK (R/W)
Secure Register Lock

Figure 43-7: SPU_SECURECTL Register Diagram

Table 43-6: SPU_SECURECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	SCRLCK	Secure Register Lock. When the `SPU_SECURECTL.SCRLCK` bit is set in combination with the `SPU_CTL.GLCK` bit, writes to the Security Configuration registers (`SPU_SECUREC[n]` and `SPU_SECUREP[n]`) are blocked and return an error which is captured in the `SPU_STAT.LWERR` bit.	
		0	Disable
		1	Enable
14 (R/W)	SINTEN	Secure Violation Interrupt Enable. The `SPU_SECURECTL.SINTEN` bit generates an interrupt if a security violation was captured. Interrupt status is provided in the `SPU_STAT.VIRQ` bit.	
		0	Disable
		1	Enable
5 (R0/W)	MSECCLR	Master Secure Clear. When the `SPU_SECURECTL.MSECCLR` bit is set, the `SPU_SECUREP[n].MSEC` bits in all `SPU_SECUREP[n]` registers are cleared. The `SPU_SECURECTL.MSECCLR` bit always reads back as a 0.	
		0	No Action
		1	Clear All Master Secure Control Bits

Table 43-6: SPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/W)	SSECCLR	Slave Secure Clear. When the `SPU_SECURECTL.SSECCLR` bit is set, the `SPU_SECUREP[n].SSEC` bits in all `SPU_SECUREP[n]` registers are cleared. The `SPU_SECURECTL.SSECCLR` bit always reads back as a 0.	
		0	No Action
		1	Clear All Slave Secure Control Bits

Secure Core Registers

A SPU register exists for every DSP core in the system. The bits enable or disable security for features in the core.

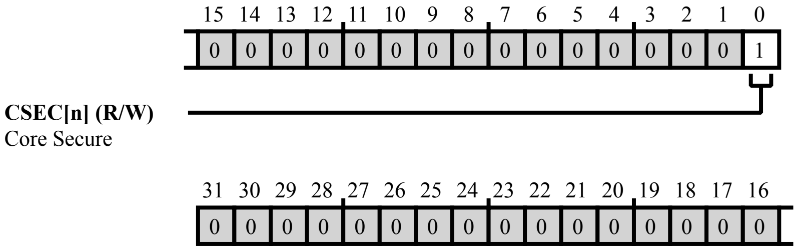

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
CSEC[n] (R/W)
Core Secure
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 43-8: SPU_SECUREC[n] Register Diagram

Table 43-7: SPU_SECUREC[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	CSEC[n]	Core Secure. The `SPU_SECUREC[n].CSEC[n]` bit controls whether non-secure accesses are allowed to L1 memory of the processor core. When =1, the core (as a slave) is set as secure meaning only secure transactions are allowed to L1.	
		0	Disable
		1	Enable

Secure Peripheral Register

In the system, each SPU_SECUREP[n] register is assigned to a specific MMR address range associated with one peripheral. Each SPU_SECUREP[n] has a Slave Secure (SSEC) bit and a Master Secure (MSEC) bit. When the Slave Secure (SSEC) bit is set, the SPU will only allow Secure Masters generating secure transactions to access the peripheral's MMR address space. When the Master Secure (MSEC) bit is set, the associated peripheral will be secure and will generate secure transactions.

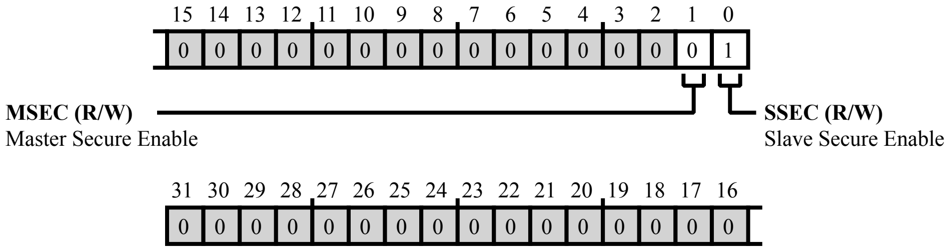

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
MSEC (R/W)
Master Secure Enable
SSEC (R/W)
Slave Secure Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 43-9: SPU_SECUREP[n] Register Diagram

Table 43-8: SPU_SECUREP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	MSEC	Master Secure Enable. The SPU_SECUREP[n].MSEC bit controls whether the peripheral generates secure transactions as a master. When clear (=0), the peripheral generates non-secure transactions as a master (if applicable). When set (=1), the peripheral generates secure transactions as a master. 0 Disable 1 Enable
0 (R/W)	SSEC	Slave Secure Enable. The SPU_SECUREP[n].SSEC bit controls whether the peripheral is protected from non-secure transactions. When clear (=0), the security status of the transaction is ignored. When set (=1), only secure transactions are allowed to access the address space of the peripheral and non-secure transactions are blocked. 0 Disable 1 Enable

Status Register

The SPU_STAT register indicates if there have been any errors, active interrupts and global lock status.

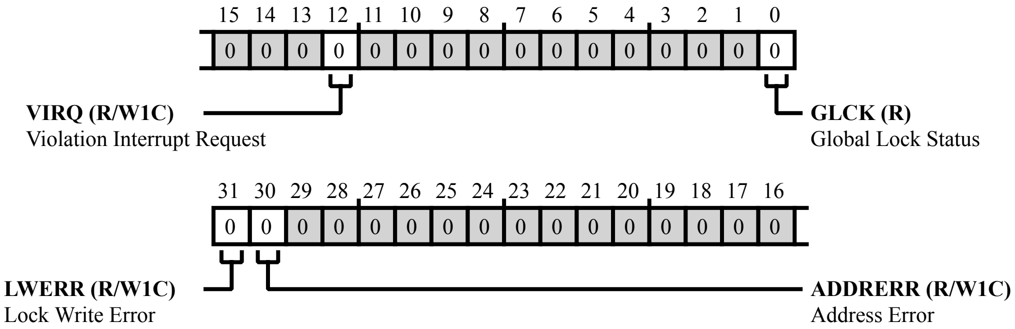

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VIRQ (R/W1C)
Violation Interrupt Request
GLCK (R)
Global Lock Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
LWERR (R/W1C)
Lock Write Error
ADDRERR (R/W1C)
Address Error

Figure 43-10: SPU_STAT Register Diagram

Table 43-9: SPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The `SPU_STAT.LWERR` indicates whether there has been an attempted write to a register in the SPU with its lock bit (`SPU_CTL.WPLCK` or SCRLCK) set while `SPU_CTL.GLCK` was asserted. This bit is W1C.	
		0	Inactive
		1	Active
30 (R/W1C)	ADDRERR	Address Error. The `SPU_STAT.ADDRERR` indicates whether there has been an attempted write to a read-only register or an access an invalid address in the SPU MMR address range. This bit is W1C.	
		0	Inactive
		1	Active
12 (R/W1C)	VIRQ	Violation Interrupt Request. The `SPU_STAT.VIRQ` bit indicates that a security and/or protection violation has been detected and interrupt asserted. This is a W1C bit.	
		0	Inactive
		1	Active
0 (R/NW)	GLCK	Global Lock Status. The `SPU_STAT.GLCK` indicates whether the global lock is enabled or disabled.	
		0	Disabled (global_lock=0)
		1	Enabled (global_lock=1)

Write Protect Register n

In the system, each `SPU_WP[n]` register is assigned to a specific MMR address range associated with one peripheral. When the appropriate bits are set, writes to the peripheral from a specific master are blocked and an error is returned to the master. For more information, see the processor specific additional information for the `SPU_WP[n]` register.

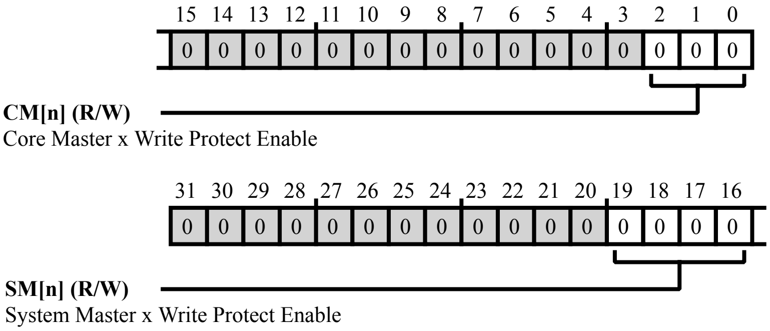


Figure 43-11: SPU_WP[n] Register Diagram

Table 43-10: SPU_WP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:16 (R/W)	SM[n]	System Master x Write Protect Enable. The `SPU_WP[n].SM[n]` bits correspond to different system masters in the system. When a particular bit is set in this field, the corresponding system master cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU.
2:0 (R/W)	CM[n]	Core Master x Write Protect Enable. The `SPU_WP[n].CM[n]` bits correspond to different cores in the system. When a particular bit is set in this field, the corresponding core cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU. Core ID 0 = M0 Supervisor and Core ID 1 = M4 Controller.

ADSP-SC5xx Write-Protect, Secure Peripheral and Secure Core Registers

The SPU consists of a collection of write-protect registers each of which are associated with a specific peripheral or slave. The SPU also has a collection of secure peripheral registers which are also associated with specific peripherals. The table gives the write-protect register and secure peripheral number for each of the peripherals that are provided with write protection and security through the SPU. The SPU for ADSP-SC5xx is configured with 188 write-protect registers and also 188 secure peripheral registers. The number corresponding to a peripheral correlates to both the Write-Protect register and the Secure Peripheral register.

For each processor, there are different numbers of masters that are able to access the SMMR space. The ***SPU_WPn.CMn and SPU_WPn.SMn Bits*** table shows which bits enable the protection against which master.

Table 43-11: SPU_WPn.CMn and SPU_WPn.SMn Bits

Bit Number	Bit Name	Description
0	CM_WP[0]	ARM Core 0
1	CM_WP[1]	SHARC Core 1
2	CM_WP[2]	SHARC Core 2
15	SM_WP[0]	PCIe
16	SM_WP[1]	DBG
17	SM_WP[2]	Embeded Trace Router (ETR)
18	SM_WP[3]	Enhanced Bandwidth MDMA

For each peripheral, there is a corresponding write-protect register, `SPU_WP[n]`, and secure peripheral register, `SPU_SECUREP[n]`. The table shows the write-protect register and secure peripheral number for each peripheral.

Table 43-12: Write-Protect Register and Secure Peripheral Number (n)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
0	Reserved	WP0	SECUREP0	N/A
1	Reserved	WP1	SECUREP1	N/A
2	Reserved	WP2	SECUREP2	N/A
3	LP0	WP3	SECUREP3	N/A
4	LP1	WP4	SECUREP4	N/A
5	LP0 DMA30	WP5	SECUREP5	Yes
6	LP1 DMA36	WP6	SECUREP6	Yes
7	CAN0	WP7	SECUREP7	N/A
8	CAN1	WP8	SECUREP8	N/A
9	TIMER0	WP9	SECUREP9	N/A
10	CRC0	WP10	SECUREP10	N/A
11	CRC1	WP11	SECUREP11	N/A
12	TWI0	WP12	SECUREP12	N/A
13	TWI1	WP13	SECUREP13	N/A
14	TWI2	WP14	SECUREP14	N/A
15	SPORT 0A	WP15	SECUREP15	N/A
16	SPORT 0B	WP16	SECUREP16	N/A
17	SPORT 1A	WP17	SECUREP17	N/A

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
18	SPORT 1B	WP18	SECUREP18	N/A
19	SPORT 2A	WP19	SECUREP19	N/A
20	SPORT 2B	WP20	SECUREP20	N/A
21	SPORT 3A	WP21	SECUREP21	N/A
22	SPORT 3B	WP22	SECUREP22	N/A
23	SPORT 4A	WP23	SECUREP23	N/A
24	SPORT 4B	WP24	SECUREP24	N/A
25	SPORT 5A	WP25	SECUREP25	N/A
26	SPORT 5B	WP26	SECUREP26	N/A
27	SPORT 6A	WP27	SECUREP27	N/A
28	SPORT 6B	WP28	SECUREP28	N/A
29	SPORT 7A	WP29	SECUREP29	N/A
30	SPORT 7B	WP30	SECUREP30	N/A
31	UART0	WP31	SECUREP31	N/A
32	UART1	WP32	SECUREP32	N/A
33	UART2	WP33	SECUREP33	N/A
34	PORTA	WP34	SECUREP34	N/A
35	PORTB	WP35	SECUREP35	N/A
36	PORTC	WP36	SECUREP36	N/A
37	PORTD	WP37	SECUREP37	N/A
38	PORTE	WP38	SECUREP38	N/A
39	PORTF	WP39	SECUREP39	N/A
40	PORTG	WP40	SECUREP40	N/A
41	PADS	WP41	SECUREP41	N/A
42	PINT0	WP42	SECUREP42	N/A
43	PINT1	WP43	SECUREP43	N/A
44	PINT2	WP44	SECUREP44	N/A
45	PINT3	WP45	SECUREP45	N/A
46	PINT4	WP46	SECUREP46	N/A
47	PINT5	WP47	SECUREP47	N/A
48	SMC0	WP48	SECUREP48	N/A

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
49	SMPU-SMC	WP49	SECUREP49	N/A
50	WDT0	WP50	SECUREP50	N/A
51	WDT1	WP51	SECUREP51	N/A
52	EPWM0	WP52	SECUREP52	N/A
53	EPWM1	WP53	SECUREP53	N/A
54	EPWM2	WP54	SECUREP54	N/A
55	EMAC0	WP55	SECUREP55	Yes
56	EMAC1	WP56	SECUREP56	Yes
57	CNT0	WP57	SECUREP57	N/A
58	MSI0	WP58	SECUREP58	Yes
59	OTP-MMR	WP59	SECUREP59	N/A
60	SINC0	WP60	SECUREP60	Yes
61	SWU-SMC	WP61	SECUREP61	N/A
62	TMU0	WP62	SECUREP62	N/A
63	HADC0	WP63	SECUREP63	N/A
64	HAE0	WP64	SECUREP64	N/A
65	ACM0	WP65	SECUREP65	N/A
66	SPORT 0A DMA0	WP66	SECUREP66	Yes
67	SPORT 0B DMA1	WP67	SECUREP67	Yes
68	SPORT 1A DMA2	WP68	SECUREP68	Yes
69	SPORT 1B DMA3	WP69	SECUREP69	Yes
70	SPORT 2A DMA4	WP70	SECUREP70	Yes
71	SPORT 2B DMA5	WP71	SECUREP71	Yes
72	SPORT 3A DMA6	WP72	SECUREP72	Yes
73	SPORT 3B DMA7	WP73	SECUREP73	Yes
74	SPORT 4A DMA10	WP74	SECUREP74	Yes
75	SPORT 4B DMA11	WP75	SECUREP75	Yes
76	SPORT 5A DMA12	WP76	SECUREP76	Yes
77	SPORT 5B DMA13	WP77	SECUREP77	Yes
78	SPORT 6A DMA14	WP78	SECUREP78	Yes
79	SPORT 6B DMA15	WP79	SECUREP79	Yes

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
80	SPORT 7A DMA16	WP80	SECUREP80	Yes
81	SPORT 7B DMA17	WP81	SECUREP81	Yes
82	UART0 RX DMA20	WP82	SECUREP82	Yes
83	UART0 TX DMA21	WP83	SECUREP83	Yes
84	UART1 RX DMA34	WP84	SECUREP84	Yes
85	UART1 TX DMA35	WP85	SECUREP85	Yes
86	UART2 RX DMA37	WP86	SECUREP86	Yes
87	UART2 TX DMA38	WP87	SECUREP87	Yes
88	MDMA0 SRC CRC0 DMA8	WP88	SECUREP88	Yes
89	MDMA0 DST CRC0 DMA9	WP89	SECUREP89	Yes
90	MDMA1 SRC CRC1 DMA18	WP90	SECUREP90	Yes
91	MDMA1 DST CRC1 DMA19	WP91	SECUREP91	Yes
92	HAE RX0 DMA32	WP92	SECUREP92	Yes
93	HAE RX1 DMA33	WP93	SECUREP93	Yes
94	HAE TX0 DMA31	WP94	SECUREP94	Yes
95	EPPI0	WP95	SECUREP95	N/A
96	SWU-SPIF(SPI2)	WP96	SECUREP96	N/A
97	SPI0	WP97	SECUREP97	N/A
98	SPI1	WP98	SECUREP98	N/A
99	SPI2	WP99	SECUREP99	N/A
100	SWU-PCIE-SLV	WP100	SECUREP100	N/A
101	SPI0 TX DMA22	WP101	SECUREP101	Yes
102	SPI0 RX DMA23	WP102	SECUREP102	Yes
103	SPI1 TX DMA24	WP103	SECUREP103	Yes
104	SPI1 RX DMA25	WP104	SECUREP104	Yes
105	SPI2 TX DMA26	WP105	SECUREP105	Yes
106	SPI2 RX DMA27	WP106	SECUREP106	Yes
107	EPPI0 Channel 0 DMA28	WP107	SECUREP107	Yes
108	EPPI0 Channel 1 DMA29	WP108	SECUREP108	Yes
109	DMC0	WP109	SECUREP109	N/A
110	DMC0-PHY	WP110	SECUREP110	N/A

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
111	DMC0-DFT	WP111	SECUREP111	N/A
112	DMC1	WP112	SECUREP112	N/A
113	DMC1-PHY	WP113	SECUREP113	N/A
114	DMC1-DFT	WP114	SECUREP114	N/A
115	L2CTLport0	WP115	SECUREP115	N/A
116	L2CTL port 1	WP116	SECUREP116	N/A
117	L2CTL port 2	WP117	SECUREP117	N/A
118	SMPU-L2CTL, Core Port 0	WP118	SECUREP118	N/A
119	SMPU-L2CTL, DMA Port 0	WP119	SECUREP119	N/A
120	SMPU-L2CTL, Core Port 1	WP120	SECUREP120	N/A
121	SMPU-L2CTL, DMA Port1	WP121	SECUREP121	N/A
122	SMPU-L2CTL, Core Port 2	WP122	SECUREP122	N/A
123	SMPU-L2CTL, DMA Port 2	WP123	SECUREP123	N/A
124	SEC0	WP124	SECUREP124	N/A
125	TRU0	WP125	SECUREP125	N/A
126	RCU0	WP126	SECUREP126	N/A
127	SPU0	WP127	SECUREP127	N/A
128	CGU0(PLL-Dig)	WP128	SECUREP128	N/A
129	CGU1(PLL-Dig)	WP129	SECUREP129	N/A
130	CDU0	WP130	SECUREP130	N/A
131	DPM0	WP131	SECUREP131	N/A
132	SWU_L2CTL,Core Port 0	WP132	SECUREP132	N/A
133	SWU_L2CTL, DMA Port 0	WP133	SECUREP133	N/A
134	SWU_L2CTL, Core Port 1	WP134	SECUREP134	N/A
135	SWU_L2CTL, DMA Port 1	WP135	SECUREP135	N/A
136	SWU_L2CTL, Core Port 2	WP136	SECUREP136	N/A
137	SWU_L2CTL, DMA Port 2	WP137	SECUREP137	N/A
138	SWU-SMMR	WP138	SECUREP138	N/A
139	SMPU-PCIE-SLV	WP139	SECUREP139	N/A
140	Medium BW MDMA	WP140	SECUREP140	Yes
141	Maximum BW MDMA	WP141	SECUREP141	Yes

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
142	FFTA0	WP142	SECUREP142	Yes
143	MLB0	WP143	SECUREP143	Yes
144	SWU-DMC0	WP144	SECUREP144	N/A
145	SWU-DMC1	WP145	SECUREP145	N/A
146	SMPU-DMC0	WP146	SECUREP146	N/A
147	SMPU-DMC1	WP147	SECUREP147	N/A
148	STM0	WP148	SECUREP148	N/A
149	GIC-Port 0 (Cortex A5 Distributor)	WP149	SECUREP149	N/A
150	GIC-Port 1 (Cortex A5 Core)	WP150	SECUREP150	N/A
151	PCIE-DBI+PCIE-RSCKPHY	WP151	SECUREP151	Yes
153	USB0	WP153	SECUREP153	Yes
154	USB1	WP154	SECUREP154	Yes
155	FIR0	WP155	SECUREP155	Yes
156	IIR0	WP156	SECUREP156	Yes
158	EMDMA0 (2 channels)	WP158	SECUREP158	Yes
159	RTC0	WP160	SECUREP160	N/A
160	DAI0	WP161	SECUREP161	N/A
161	DAI1	WP162	SECUREP162	N/A
162	PKTE	WP163	SECUREP163	Yes
163	PKA	WP164	SECUREP164	N/A
164	DAPROM	WP165	SECUREP165	N/A
165	SHARC1 DBG	WP166	SECUREP166	N/A
166	SHARC1 CTI	WP167	SECUREP167	N/A
167	SHARC1 PTM	WP168	SECUREP168	N/A
168	STM	WP169	SECUREP169	N/A
169	SHARC2 DBG	WP170	SECUREP170	N/A
170	SHARC2 CTI	WP171	SECUREP171	N/A
171	SHARC2 PTM	WP172	SECUREP172	N/A
172	CSTF	WP173	SECUREP173	N/A
173	ETF	WP174	SECUREP174	N/A
174	ETR	WP175	SECUREP175	N/A

Table 43-12: Write-Protect Register and Secure Peripheral Number (n) (Continued)

No	Peripheral/Block Name	SWPn Mapping	SECUREPn Mapping	Master Capable
175	TPIU	WP176	SECUREP176	N/A
176	CTI Trace	WP177	SECUREP177	N/A
177	CTI System	WP178	SECUREP178	N/A
178	A5 Integration ROM	WP179	SECUREP179	N/A
179	A5 DBG	WP180	SECUREP180	N/A
180	A5 PMU	WP181	SECUREP181	N/A
181	A5 CTI	WP182	SECUREP182	N/A
182	A5 ETM	WP183	SECUREP183	N/A
183	TAPC MMR	WP184	SECUREP184	N/A
184	DebugControl	WP185	SECUREP185	N/A
185	SWU Core 1 Slave Port 1	WP186	SECUREP186	N/A
186	SWU Core 1 Slave Port 2	WP187	SECUREP187	N/A
187	SWU Core 2 Slave Port 1	WP188	SECUREP188	N/A
188	SWU Core 2 Slave Port 2	WP189	SECUREP189	N/A

ADSP-SC5xx Specific Information

The information in this section applies specifically to the ADSP-SC5xx processor models

Global Locking

The global lock signal from the SPU along with the peripheral lock bit can be used to provide lock functionality for the control MMR of the peripheral. The Global Lock (`SPU_CTL.GLCK`) field determines whether global lock is enabled or not. Global Lock is disabled if the `SPU_CTL.GLCK` field is 0xAD (default value), otherwise it is enabled. The following is a list of peripherals that have the global lock bit in their control MMR.

- General-Purpose IO (GPIO)
- System Event Controller (SEC0)
- Trigger Routing Unit (TRU0
- Clock Generation Unit (CGU0)
- Clock Generation Unit1 (CGU1)
- Clock Distribution Unit (CDU0)
- Dynamic Power Management (DPM)
- Reset Control Unit (RCU0)

- System Protection Unit (SPU0)
- L2 Memory Controller (L2CTL0)
- L2 Memory Controller (L2CTL1)
- L2 Memory Controller (L2CTL2)

44 Security Packet Engine (PKTE)

The PKTE is a security packet engine designed to off-load the host processor to improve the speed of applications requiring cryptographic processing. The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation.

PKTE Features

The PKTE has the following features.

- Hardware assisted processing for the cryptographic ciphers, hashes, and pseudo-random number generation
- Header and trailer processing for Internet security protocols
- DMA capability to move data in and out of the engine efficiently and to allow the engine to run autonomously while moving data
- Interrupt controller to signal module status and errors
- Clock manager for enabling or disabling different features to save power

NOTE: Not all algorithms, decrypt, and hash functions and extra features are available on all product models. For complete information on included features, see the product-specific data sheet.

NOTE: This packet engine provides support for various network security protocols by processing headers and trailers as well as accelerating cryptographic functions. Not all processors have direct support for Ethernet. As such, the packet engine can still be used if Ethernet is indirectly used.

PKTE Functional Description

The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation. The following sections describe these functional blocks.

ADSP-SC58x PKTE Register List

The Security Packet Engine (PKTE) provides security-related features. A set of registers governs PKTE operations. For more information on PKTE functionality, see the PKTE register descriptions.

Table 44-1: ADSP-SC58x PKTE Register List

Name	Description
PKTE_ARC4STATE_ADDR	Packet Engine ARC4 State Record Address
PKTE_ARC4STATE_BUF	Starting Entry of 256-byte ARC4 State Buffer
PKTE_BUF_PTR	Packet Engine Buffer Pointer Register
PKTE_BUF_THRESH	Packet Engine Buffer Threshold Register
PKTE_CDRBASE_ADDR	Packet Engine Command Descriptor Ring Base Address
PKTE_CDSC_CNT	Packet Engine Command Descriptor Count Register
PKTE_CDSC_INCR	Packet Engine Command Descriptor Count Increment Register
PKTE_CFG	Packet Engine Configuration Register
PKTE_CLK_CTL	PE Clock Control Register
PKTE_CONT	PKTE Continue Register
PKTE_CTL_STAT	Packet Engine Control Register
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers

Table 44-1: ADSP-SC58x PKTE Register List (Continued)

Name	Description
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_ARC4IJPTR	ARC4 i and j Pointer Register
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

ADSP-SC58x PKTE Interrupt List

Table 44-2: ADSP-SC58x PKTE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
130	PKTE0_IRQ	PKTE0 Interrupt	Level	

PKTE Definitions

Command Descriptor

An 8-word structure that is either written directly into the packet command MMR set or is placed in a Command Descriptor Ring (CDR) in the processor memory. The packet engine sequentially processes the structure. The command descriptor contains the information that varies for every packet. This information includes pointers to the SA record, the state information, the source packet, and the destination packet.

Command Descriptor Ring (CDR)

A circular contiguous portion of memory which is used to manage one or more command descriptions for the packet engine.

Result Descriptor

When the packet engine completes the processing of a packet, it writes a result descriptor with the state information. The result descriptor can be read directly from the result register set or from the Result Descriptor Ring (RDR) in the processor memory.

Result Descriptor Ring (RDR)

A circular contiguous portion of memory which holds the mirror or copy of the CDR but contains the result descriptors. The RDR and CDR can be overlaid on top of each other.

Security Association (SA) Record

A structure that contains the remainder of the information the packet engine requires to process a packet. Most of the information fields in the SA record such as the key and encryption mode are static for the lifetime of the association. The fields do not require frequent manipulation by the processor core. The SA record non-static fields are the sequence number and sequence number mask. The SA record can have a corresponding state record for saving results from the current operations that are useful for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

Cipher

A method or algorithm to encrypt or decrypt information

Hash

A cryptographic hash is a function that takes an arbitrary block of data and returns a fixed-size bit string. Four main properties define the function:

- It is easy to compute a hash value for any given input

- It is infeasible to generate the original input from a given hash
- It is infeasible to modify the input without changing the resulting hash
- It is infeasible to find two different inputs that result in the same hash

Autonomous Ring Mode (ARM)

Mode of operation in which most of the parameters as well as the data are set up in memory and moved to the engine for configuration and processing through DMA.

Target Command Mode (TCM)

Mode of operation where some parameters are set up in memory and moved into the packet engine through DMA while the other parameters are directly written to the registers. DMA moves the input and output data in and out of the engine.

Direct Host Mode (DHM)

Mode of operation that does not use DMA. All parameters are directly written to and read from the MMRs. The input and output are written to and read from the FIFO buffers.

Cipher Module

The cipher module does the symmetric encrypt or decrypt operations for:

- Data Encryption Standard (DES)
- Triple-DES
- ARC4
- Advanced Encryption Standard (AES) algorithms

The cipher module supports standard modes for DES and AES that include Electronic Code Book (ECB) and Cipher Block Chaining (CBC). The key size for DES is 56 bits, for Triple-DES is 168 bits. The AES module also provides support for AES counter-modes for IPsec and SRTP. All AES modes can use key sizes of 128 bits and 192/256 bits. Key scheduling is automatic and done in parallel with the encrypt or decrypt operation.

Hash Module

The hash module is tightly coupled with the encrypt or decrypt module and provides hardware accelerated one-way hash functions. Operations that combine both hash and encrypt or decrypt functions are provided to reduce processing time for data that needs both applied. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of hashing and the start of encryption to support protocols such as IPsec or SRTP. The HMAC keyed hashing mechanism is supported for MD5, SHA-1, SHA-2-224 and SHA-2-256. The SSL-MAC is supported for MD5 and SHA-1. A second AES-CBC module for the hash function enables parallel processing for AES-CCM, a combined hash and encrypt algorithm.

Pseudo-Random Number Generator

Cipher algorithms that operate in CBC mode or counter-mode, require an IV. This IV must not be secret; however the IV must be unpredictable and unique for each execution of the encryption process. Pseudo-random number generators are deterministic algorithms that output statistically independent and unbiased numbers. True random number generators are non-deterministic and use the randomness that occurs in a physical process. The packet engine incorporates an ANSI X9.31 compliant Pseudo Random Number Generator (PRNG) that it can use to generate unique IVs using strong encryption. The ANSI X9.31 PRNG is defined as part of the ANSI X9 standards that are used to secure financial transactions. The function can also be used for pseudo-random number generation as part of an implementation of the digital signature standard described in NIST FIPS PUB 186-2.

The PRNG function, as defined by ANSI X9.31, is based on the AES cipher. This section describes the function to promote understanding of the different inputs and outputs of the PRNG function itself.

NOTE: The PRNG in the packet engine is only based on the AES cipher with 128-bit keys. Other ciphers and key lengths are not supported for the PRNG based on ANSI X9.31.

Let $e \times K(Y)$ represent the AES encryption of Y under the key K.

The PRNG function uses three inputs:

- K, a 128-bit key
- V, a 128-bit seed value
- DT, a 128-bit date/ time vector which is updated on each iteration

The intermediate value I is the result of an AES encryption of the data and time vector under key K.

$I = e \times K(DT)$

That value I is then XOR-ed with the seed V and AES encrypted under key K. The result R is the output of the PRNG function.

$R = e \times K(I \text{ XOR } V)$

A new seed value V is generated from the AES encryption of the result R XOR'ed with the intermediate value I under the key K.

$V = e \times K(R \text{ XOR } I)$

The PRNG function is deeply integrated inside the datapath of the packet engine. The function is controlled indirectly through the PRNG mode bits in the `PKTE_CTL_STAT.PRNGMD` bit field of the command descriptor and the IV source selection bits in the `PKTE_SA_CMD0.IVSRC` bit field of the SA record.

The PKTE module supports four different modes.

1. Load IV from PRNG for the current operation: `PKTE_CTL_STAT.PRNGMD` = 0b00 and `PKTE_SA_CMD0.IVSRC` = 0b11.
2. PRNG init mode initializes the PRNG with a key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD` = 0b01.

   Before the PRNG function can be used, it must be initialized with a key, seed, and date/time value. At initialization, the key, seed, and date/time values are programmed. Other PRNG operations do not change the key, however the seed, and date/time values are updated (not re-programmed). The date/time is updated every 128 system clock cycles.

   The date/time is a 128-bit value with randomly distributed number of ones and zeros. It must not be all zeros.

   The ***SA Record for PRNG Init Operation*** table shows how the key, seed and date/time values are loaded into the PKTE registers for initialization.

   Table 44-3: SA Record for PRNG Init Operation

Parameter	SA Field	Description
K	SA_KEY0	PRNG key [127:96]
	SA_KEY1	PRNG key [95:64]
	SA_KEY2	PRNG key [63:32]
	SA_KEY3	PRNG key [31:0]
V	SA_IDIGEST0	PRNG seed [127:96]
	SA_IDIGEST1	PRNG seed [95:64]
	SA_IDIGEST2	PRNG seed [63:32]
	SA_IDIGEST3	PRNG seed [31:0]
DT	SA_ODIGEST0	PRNG date/time [127:96]
	SA_ODIGEST1	PRNG date/time [95:64]
	SA_ODIGEST2	PRNG date/time [63:32]
	SA_ODIGEST3	PRNG date/time [31:0]

3. PRNG generate mode generates pseudo-random data on initialized key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD` = 0b10.

   The PRNG function can be used to generated pseudo-random data for other purposes than IVs. For this mode, the PRNG must have been initialized once with the PRNG init mode.

   The PRNG generate mode uses the initialized key and unique changing date/time value as inputs. The pseudo-random data is output to the output buffer of the packet engine.

   The `LEN` field in the command descriptor indicates the amount of pseudo random data that is generated in multiples of 16 bytes. The maximum is 255 × 16 = 4080 bytes.

In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

Directly after the PRNG generate mode, a new pseudo-random number is generated and available for the next operation that uses the option `PKTE_SA_CMD0.IVSRC = PRNG`.

4. PRNG test mode generates pseudo-random data on initialized key, seed, and input (test) data: `PKTE_CTL_STAT.PRNGMD` = 0b11.

   The PRNG test mode can be used to test the correctness of the PRNG function. This mode is similar to the PRNG generate mode, except that the data is read from the input buffer of the packet engine, instead of the date/time value.

   For this mode, the PRNG must have been initialized once with the PRNG Init mode.

   The `LEN` field in the command descriptor indicates the amount of pseudo-random data to be generated in multiples of 16 bytes. The maximum is limited to the `LEN` field in bytes.

   In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In the direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

   Directly after the PRNG test mode, a new pseudo-random number is generated and available for the next operation that uses the option `PKTE_SA_CMD0.IVSRC = PRNG`.

Packet Engine Processing Details

This section describes data processing through the packet engine. It describes padding and supported algorithms for each protocol.

A valid Security Association (SA) must be created before packet processing can start. A formatted SA record must reside in memory and be accessible to the packet engine. The host processor application is responsible for these tasks.

Crypto Padding

Padding is the process of adding data to fill-out a fixed-size plain text data structure. Three factors determine when to use a pad field:

1. If a block cipher encryption algorithm is used, a pad field is used to expand the plain text to a multiple of the block size.
2. Padding can be used to ensure that the cipher text terminates on an n-byte boundary.
3. Padding can conceal the actual length of the payload.

To facilitate peak encrypt or decrypt performance, the packet engine supports the following most commonly used padding functions in hardware:

1. Pad generation and insertion of pad bytes to the end of plain text prior to encryption, for outbound operations.

2. Pad verification to check for correct padding after decrypting a packet for inbound operations.
3. Pad consumption to strip the pad bytes from the plain text data after decrypting a packet, for inbound operations.

Pad Generation and Insertion

The pad type and quantity of bytes the packet engine inserts depends on the plain text length and the value of the following fields:

- `PKTE_SA_CMD0.PADTYPE` and `PKTE_SA_CMD0.EXTPAD` defines the type of padding
- `PKTE_CTL_STAT.PADVAL` defines a value that is inserted in the pad
- `PKTE_SA_CMD0.CIPHER` enforces a certain pad alignment
- `PKTE_SA_CMD1.CIPHERMD` enforces a certain pad alignment
- `PKTE_SA_CMD0.SCPAD` allows stream ciphers to be padded
- `PKTE_CTL_STAT.PADCTLSTAT` controls the pad alignment

The `PKTE_CTL_STAT.PADCTLSTAT` bit field of the result descriptor returns the total number of inserted pad bytes.

Pad Types

The pad type bit field (`PKTE_SA_CMD0.PADTYPE`) and the extended pad bit (`PKTE_SA_CMD0.EXTPAD`) select the pad type for the extended protocol group. The packet engine can generate different pad types in hardware as described in the ***Pad Examples*** table.

The `PKTE_CTL_STAT.PADVAL` bit field, together with the number of pad bytes, defines the value that is inserted in the pad. The format of the pad and the use of this field is best explained in an example (see the ***Pad Examples*** table).

For the IPsec pad type, this field holds the value that is inserted into the next header field (in the ESP trailer) of the innermost operation's header. For the Constant pad type or the Constant SSL pad type, this field holds the inserted fixed constant pad value. For all other pad types, this field is not used and must be zero.

Table 44-4: Pad Types

Pad Type	Value	Description
IPSec	0b000	Append 0 to 255 pad bytes, followed by a pad length byte and a next header byte. The first pad byte appended to the plain text is numbered 1, with subsequent pad bytes making up a monotonically increasing sequence: 1, 2, 3 and up. Append the pad length field that indicates the number of pad bytes (0–255), where a value of zero indicates no pad bytes present. Append the next header byte as specified in the `PKTE_CTL_STAT.PADVAL` field of the command descriptor. A minimum of 2 bytes are appended; zero pad bytes plus the pad length byte plus the next header byte, in which case the `PKTE_CTL_STAT.PADCTLSTAT` field in the result descriptor returns 0x02. A maximum of 257 bytes can be appended, in which case the `PKTE_CTL_STAT.PADCTLSTAT` field in the descriptor returns 0x01.
PKCS#7	0b001	Appends 1–128 pad bytes with a pad byte value equal to the pad length (1–128).
Constant	0b010	Appends 0–255 pad bytes of a user-specified character to the plain text data. This character is specified in the `PKTE_CTL_STAT.PADVAL` field of the command descriptor.
Zero	0b011	Appends 0–255 pad bytes of 0x00 to the plain text data.
Constant SSL	0b110	Appends 0–255 pad bytes of a user-specified character to the plain text data, followed by a ‘pad length’ byte (0–255). This character is specified in the `PKTE_CTL_STAT.PADVAL` field of the command descriptor. A total of 256 bytes can be appended, in which case the pad field returns 0x00.

For example, the ***Pad Examples*** table shows the appended pad for any of the pad types for an outbound (encrypt) operation. The table shows a plain text input of 2 bytes using the 8-byte block cipher crypto-algorithm DES-ECB and a `PKTE_CTL_STAT.PADVAL` field value of 0xAA.

Table 44-5: Pad Examples

Pad Type	Pad field (extended to Plain Text)	`PKTE_CTL_STAT`
IPSec	0x01, 0x02, 0x03, 0x04, 0x04, 0xAA	0x06
PKCS#7	0x06, 0x06, 0x06, 0x06, 0x06, 0x06	0x06
Constant	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA	0x06
Zero	0x00, 0x00, 0x00, 0x00, 0x00, 0x00	0x06
Constant SSL	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0x05	0x06

Pad Length

The ***Pad Alignment*** table lists the alignment (boundaries) to which the packet engine will pad, based on:

- The selected crypto-algorithm
- Crypto mode
- The value of the pad stream cipher bit
- The value of pad control

The minimum number of inserted pad bytes depends on the cipher algorithm, selected using the `PKTE_SA_CMD0.CIPHER` bit field, and the cipher mode, selected using the `PKTE_SA_CMD1.CIPHERMD` bit field.

For block ciphers, the plain text data is always (as a minimum) padded to the next block boundary. More pad bytes beyond the algorithm or protocol alignment requirements can be inserted using the pad control (`PKTE_CTL_STAT.PADCTLSTAT`) in the command descriptor. This feature can be used for *traffic flow security* to conceal the number of plain text bytes in an encrypted packet.

Encrypt operations that use block ciphers have minimum pad requirements based on their block size. The packet engine enforces a minimum pad alignment for block ciphers according to the ***Pad Alignment*** table. For ESP outbound operations, the minimum pad alignment is forced to 4 bytes.

For stream ciphers and null crypto the data is never padded when the stream cipher pad bit (`PKTE_SA_CMD0.SCPAD`) =0. When `PKTE_SA_CMD0.SCPAD` =1 the plain text data is padded to the length as defined by the `PKTE_CTL_STAT.PADCTLSTAT` bits in the command descriptor.

NOTE: The SSL protocol does not allow padding to exceed the ciphers block length. This length is 8 bytes for DES/Triple-DES and 16 bytes for AES. For SSL, the packet engine does not enforce this pad alignment value. The host processor must ensure that the `PKTE_CTL_STAT.PADCTLSTAT` bit field is configured correctly.

Table 44-6: Pad Alignment

Pad Control PADCTLSTAT = PKTE_CTL_STAT[31:24]			0x00	0x01[*1]	0x02	0x04	0x08	0x10	0x20	0x40	0x80[*2]
Crypto Algorithm	**Crypto Mode**	**Pad Stream Ciphers**	**%8**	**%1**	**%4**	**%8**	**%16**	**%32**	**%64**	**%128**	**%256**
DES	ECB, CBC	N/A	8	8	8	8	16	32	64	128	256
AES	ECB, CBC	N/A	16	16	16	16	16	32	64	128	256
	CTR, ICM with ESP	N/A	8	4	4	8	16	32	64	128	256
	CTR, ICM no ESP	no	0	0	0	0	0	0	0	0	0
		yes	8	0	4	8	16	32	64	128	256
NULL	with ESP	N/A	8	4	4	8	16	32	64	128	256
	no ESP	no	8	0	0	0	0	0	0	0	0
		yes	8	0	4	8	16	32	64	128	256
ARC4	with ESP	N/A	8	4	4	8	16	32	64	128	256
	no ESP	no	0	0	0	0	0	0	0	0	0
		yes	8	0	4	8	16	32	64	128	256

*1 If `PKTE_CTL_STAT.PADCTLSTAT` is configured for no padding (0x01), it does not mean that no padding bytes are inserted. When PKCS#7 padding is selected, a pad length field with a value =1 is inserted. When SSL or TLS padding is selected, a pad length field with a value =0 is inserted. When IPsec padding is selected, a pad length field is forced to (`PKTE_CTL_STAT.PADCTLSTAT`=0x20). When zero pad and constant pad are selected, no pad bytes are inserted.

*2 Pad type PKCS#7 supports a maximum length of 128 pad bytes, so the packet engine overrules a 256-byte alignment (`PKTE_CTL_STAT.PADCTLSTAT`=0x80) to a 128-byte boundary (`PKTE_CTL_STAT.PADCTLSTAT`=0x40).

The packet engine does not constrain the pad type that is used for an operation; any pad type can be used for each operation. The user must be aware that some protocol specifications only allow specific pad types. The SRTP specification does not have padding defined as padding performed by RTP. The host must pad the RTP packet.

The host software can implement padding that is not supported in hardware. In this case, the host must select the *zero pad* type and set the `PKTE_CTL_STAT.PADCTLSTAT` bit field in the packet descriptor to zero (no padding). The hardware padding engine does not add any bytes. When using a block cipher, the host must insert pad bytes. Then, the data to be encrypted (plain text and pad bytes) are a multiple of the block ciphers boundary. For stream ciphers, any number of pad bytes can be added.

Pad Verification and Consumption

The packet engine can validate a pad type against the expected values. The value of the following bits controls the pad verify function:

- `PKTE_SA_CMD0.PADTYPE` and `PKTE_SA_CMD0.EXTPAD` define the type of padding
- `PKTE_SA_CMD0.CIPHER` enforces a certain pad alignment
- `PKTE_SA_CMD1.CIPHERMD` enforces a certain pad alignment
- `PKTE_SA_CMD0.SCPAD` allows stream ciphers to be padded

When packet processing is complete, the status byte in the first word of the result descriptor reports the pad verification status. Refer to the Table 44-28 Extended Error Codes - Status Encoding table.

The `PKTE_CTL_STAT.PADCTLSTAT` bits in the first word of the result descriptor return the total number of detected pad bytes and returns zero for a pad verify error.

When the IPsec pad type is selected, the `PKTE_CTL_STAT.PADVAL` bit field of the result descriptor returns the next header field. For IPsec ESP outbound operations, this field returns the decimal value 50. For IPsec inbound operations and basic inbound operations that use the IPsec pad mode, the packet engine returns the next header field it detects on the header of the innermost operation. This value is typical for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other inbound operations, the returned pad value is zero.

Pad verification is performed for inbound (decrypt) operations that use IPsec, TLS/DTLS or PKCS#7 pad type:

- In combination with a block cipher algorithm: DES-ECB, DES-CBC, AES-ECB, AES-CBC,
- In combination with a stream cipher and stream cipher padding `PKTE_SA_CMD0.SCPAD` enabled: AES-CTR, AES-ICM and ARC4
- In combination with null-crypto (no encryption).

Pad Types

The `PKTE_SA_CMD0.PADTYPE` bit field and the extended pad `PKTE_SA_CMD0.EXTPAD` bit selects the pad type for the extended protocol group pad type. The packet engine can verify the different pad types in hardware as described in the ***Pad Types*** table.

The constant and zero pad types are not verified since they do not include a pad length field. The SSL pad type is not verified since it does not have a defined pattern.

Table 44-7: Pad Types

Pad Type	SA_CMD0[18, 7:6]	Description
IPSec	0b000	Verify that the pad field includes 0–255 pad bytes, followed by a correct pad length and a next header byte. Verify that pad bytes appended to the plain text are an incremental count, starting at one. Verify that the pad length field is the number of pad bytes (0–255), where a value of zero indicates no pad bytes present. Verify that a next header byte is present as the last byte of the packet, the value is not verified. This is after removal of the ICV. The total number of detected pad bytes is returned in the `PKTE_CTL_STAT.PADCTLSTAT` field in the result descriptor. **NOTE:** A minimum of 2 bytes must be present, zero pad bytes plus the pad length byte plus the next header byte. In this case the `PKTE_CTL_STAT.PADCTLSTAT` field in the descriptor returns 0x02. A maximum of 257 bytes can be present, in which case the `PKTE_CTL_STAT.PADCTLSTAT` field in the result descriptor returns 0x01. The value of the next header byte is returned in the `PKTE_CTL_STAT.PADVAL` field in the result descriptor.
PKCS#7	0b001	Verify that the pad field includes 1–128 pad bytes, with a pad byte value equal to the pad length (1–128).

When a block cipher is used and the payload is not padded to the nearest block size boundary, as required by the protocol, a *block size error* is generated for all pad types.

Pad Consumption

The packet engine can optionally consume the decrypted pad bytes for an inbound operation that uses the IPsec, SSL, TLS/DTLS, or PKCS#7 pad types. The pad types constant and zero are not consumed since they do not include a pad length field.

Pad consumption (or stripping) is selected on a flow-by-flow basis in the SA-record with the `PKTE_SA_CMD1.CPYPAD` bit. When this bit is set, the length returned in the LEN field of the result descriptor is the total length of the plain text including the pad. When the `PKTE_SA_CMD1.CPYPAD` is disabled, the detected pad length, as returned in the `PKTE_CTL_STAT.PADCTLSTAT` bits, is subtracted from the total length and then returned in the LEN field of the result descriptor.

The pad is always written to the result packet buffer in memory. When the `PKTE_SA_CMD1.CPYPAD` bit is disabled, only the result length is corrected.

Crypto and Hash Algorithms

The packet engine supports a wide range of crypto and hash algorithms to accelerate basic operations and protocol operations. These algorithms are:

- Basic Encrypt and Basic Decrypt Operations
- Basic Hash Operations
- Basic Encrypt-Hash and Basic Hash-Decrypt Operations
- IPSec ESP Operations
- SRTP Operations

The following tables provide allowed algorithm combinations. Those algorithms not listed in the tables are invalid and can give unexpected results.

NOTE: Not all crypto and hash algorithms are available on all product models. For information on algorithm availability, see the product-specific data sheet.

Table 44-8: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
DES, Triple-DES	ECB, CBC
AES	ECB, CBC, CRT, ICM
NULL	—

Table 44-9: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
SHA-1	Basic hash, HMAC, SSL-MAC
SHA-224	Basic hash, HMAC
SHA-256	Basic hash, HMAC
NULL	—

Table 44-10: Algorithms for Basic Encrypt-Hash and Basic Hash-Decrypt Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	ECB, CBC	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—

Table 44-10: Algorithms for Basic Encrypt-Hash and Basic Hash-Decrypt Operations (Continued)

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	ECB, CBC, CRT, ICM	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—
NULL	—	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC

Table 44-11: Algorithms for IPsec ESP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	—

Table 44-12: Algorithms for Basic SSL and Extended SSL Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—

Table 44-12: Algorithms for Basic SSL and Extended SSL Operations (Continued)

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
ARC4	Stateful	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
NULL	—	SHA-1	SSL-MAC
	—	MD5	SSL-MAC
	—	NULL	—

Table 44-13: Algorithms for Basic TLS, Extended TLS and DTLS Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
ARC4[1]	Stateful	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	—	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—

1 Only for Basic TLS and Extended TLS

Table 44-14: Algorithms for SRTP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	ICM	SHA-1	HMAC
NULL	—	SHA-1	HMAC

IV Processing

An initialization vector (IV) is necessary to start a cipher stream or a block cipher in any of the streaming modes of operation. The IV must be unique for each packet. The IV ensures that all cipher texts are unique even if produced by the same encryption key. This functionality prevents every packet from needing a unique encryption key.

Depending on the packet engine operation, the IV can be loaded from different sources. The IV format depends on the algorithm and the source of the IV. The ***Format of the IV*** table provides an overview of all IV formats.

Table 44-15: Format of the IV

Algorithm	IV Source (**`PKTE_SA_CMD0.IVSRC`**)	Format	IV Offset (**`PKTE_SA_CMD1.HSHCOFFST`**)
DES/Triple-DES (CBC)	Previous Result of IV	Internal IV register [63:0]	0x00
	Input Buffer	Input buffer [63:0]	0x02
	Saved IV	State Record Saved IV [63:0]	0x00
	Automatic	PRNG output [63:0]	0x00
AES (CBC)	Previous Result of IV	Internal IV register [127:0]	0x00
	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	PRNG output [127:0]	0x00
AES (ICM) for Basic and SRTP	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
AES (CTR) for Basic and IPsec	Input Buffer	SA_NONCE || Input Buffer[63:0] || 0x00000001	0x02
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	SA_NONCE || PRNG output [95:32] || 0x00000001	0x00

Notes:

1. The `PKTE_SA_CMD1.HSHCOFFST` bit field provides the IV offset. The offset is only applicable for basic hash or encrypt operations. For protocol operations, the offset is automatically enforced.

2. AES-CTR: The Nonce value as described in RFC 3686, is mapped to the `PKTE_SA_NONCE` register. This Nonce value remains constant for the lifetime of the security association.
3. The host processor controls the IV update using the save-IV bit (`PKTE_SA_CMD0.SVIV`). When part of a packet is processed using a stream cipher and the encrypt or decrypt data is not an integer multiple of the block size, the saved IV is invalid. It must not be used to resume processing the packet.
4. The packet engine supports automatic IV generation for outbound operations. A new IV is generated for every packet with the internal PRNG. This automatic IV generation can be used for all DES, Triple-DES, and AES modes that use an IV, except for AES-ICM mode.
5. For outbound operations, automatic IV generation is the most efficient. No additional I/O is required, and the host processor does not need to provide the IV. When the saved IV option supplies the IV, the IV must be changed for each packet sent. This activity happens when the packet engine writes back the IV to the state record after processing.
6. Outbound IPsec operations put the IV explicitly at the front of the packet. For an inbound IPsec operation, loading from the input buffer is most efficient and always used.
7. CBC processing must not use a predictable IV. Do not use the saved IV and previous result IV options for CBC processing. Refer to RFC 3602 for more details.

ARC4 Processing

The ARC4 algorithm supports two modes of operation: stateless and stateful. For SSL, TLS and DTLS operations that use the ARC4 algorithm, the mode must be set to stateful.

Stateless Mode. Each packet is processed with a newly initialized ARC4 key taken from the key field of the SA record. In this mode, the state information from the SA is never to be read.

CAUTION: If an ARC4 operation in stateless mode is interrupted by an *Interface Error* and the ARC4 state building process is aborted, the next packet fails. The ARC4 is not reset between two packets. For the next packet the ARC4 proceeds from the internal state that it was aborted. Even a soft-reset does not reset the ARC4 internal state, a hardware reset is required.

Stateful Mode. When the `PKTE_CTL_STAT.INITARC4` bit =1, the ARC4 algorithm initializes using the key specified in the SA record. When the `PKTE_CTL_STAT.INITARC4` bit =0, the ARC4 context is read from the ARC4 state field of the SA record and the*i* and *j* pointer field of the SA record. The encrypt and decrypt processing continues from this algorithm state.

The packet engine supports ARC4 key sizes of 40 bits to 128 bits. Longer keys can be used, but they cannot be made inside the packet engine.

The host processor applies ARC4 key scheduling function to the s-box and puts the 256-byte result into the ARC4 state record. It writes the *i* and *j* pointers in the SA (initial $i = 1, j = 0$). The host programs the packet engine and specifies stateful operation to continue the ARC4 algorithm.

Hash State Loading

The hash state can be loaded from various sources, depending on the selected protocol and hash algorithm. The ***Different Sources for Loading the Hash State*** table provides a list of all the options.

Table 44-16: Different Sources for Loading the Hash State

Hash Algorithm `PKTE_SA_CMD0.HASHSRC` =	From SA 0b00	RESERVED 0b01	From State 0b10	No Load 0b11
SHA-1	yes	-	yes	Yes
SHA-224	Yes	-	Yes	Yes
SHA-256	Yes	-	Yes	Yes
Null hash	x	x	x	x

Sequence Number Processing

The packet engine supports sequence number generation and verification for IPsec and extended SSL, TLS, and DTLS protocol operations.

A Sequence Number (SN) is an unsigned number that a sender must implement and a receiver can use to support anti-replay service (replay attacks) for a specific SA. This processing includes detection of the same packet that arrives more than once and detection of packets that arrive in an incorrect sequence and is outside an accepted level of order relative to the last received packet.

The packet engine supports the following options.

- SSL v3.0 with 64-bit SN generation and overflow protection (see The SSL Protocol Version 3.0).
- DTLS v1.0 with 48-bit SN generation, verification, and overflow protection (see RFC 4347).
- DTLS full anti-replay support using 64-bit Sequence Number mask.
- IPsec full anti-replay support using 64-bit Sequence Number mask.
- Sequence number loaded from SA for outbound operations and is retrieved from the input packet for inbound operations.

The sequence number and sequence number mask fields are part of the SA record.

Sequence Number Processing in Extended SSL/TLS

SSL and TLS use an implicit sequence number of 64 bits that is not send in the packet but part of the hash.

For inbound operations, no sequence number verification is performed; instead an incorrect sequence number results in an authentication error (`PKTE_CTL_STAT.AUTHERR`).

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine generates a sequence number error when the 64-bit sequence number counter overflows (counter is 2^{64}-1 and increments to 0). The host processor must not send the packet.

For both outbound and inbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine reads the sequence number from the `PKTE_SA_SEQNUM[n]` registers in the SA. When the operation is finished, the packet engine stores the incremented sequence number in the same SA fields. The sequence number mask `PKTE_SA_SEQNUM_MSK[n]` registers in the SA are not used.

For both outbound and inbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled, the packet engine does not increment the sequence number. It expects the host to provide the sequence number through the input buffer as part of the header.

Sequence Number Processing in DTLS

DTLS uses an explicit sequence number of 64 bits that is sent in the packet. The DTLS sequence number is composed of the epoch (16 bits) and packet sequence number (48 bits) that together form the 64-bit number, like the TLS sequence number. The epoch is incremented after each *Change Cipher Spec* message. The packet sequence number is incremented per packet starting from zero after each change cipher spec message.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled, the packet engine reads the sequence number from the `PKTE_SA_SEQNUM[n]` fields in the SA, then inserts the sequence number in the packet. Then the packet engine stores the incremented sequence number in the `PKTE_SA_SEQNUM[n]` fields in the SA. The sequence number mask `PKTE_SA_SEQNUM_MSK[n]` fields in the SA are not used.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine generates a sequence number error when the 48-bit sequence number counter overflows (counter is $2^{48} - 1$ and increments to 0). The host must not send the packet.

For outbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled or `PKTE_SA_CMD1.ENSQNCHK` disabled, the packet engine does not increment and verify a sequence number counter overflow, and therefore never generates a sequence number error. With `PKTE_SA_CMD0.HDRPROC` disabled the Packet Engine does not update the sequence number and sequence number mask fields in the SA and expects the host to provide the sequence number through the input buffer as part of the header.

For inbound operations, with `PKTE_SA_CMD0.HDRPROC` enabled and `PKTE_SA_CMD1.ENSQNCHK` enabled, the packet engine verifies the `PKTE_SA_SEQNUM[n]` fields against the sequence number in the packet using the `PKTE_SA_SEQNUM_MSK[n]` from the SA. Three situations can occur:

1. If the received sequence number falls outside and above the 64-bit sequence number mask, the mask is shifted. The packet engine updates the SA with the received sequence number and the shifted sequence number mask.
2. If the received sequence number falls inside the 64-bit sequence number mask and is not a duplicate sequence number (the same as received before), the corresponding bit in the mask is set. The packet engine updates the SA with the received sequence number and the updated sequence number mask.
3. If the received sequence number falls outside the 64-bit sequence number mask or matches and earlier received number a sequence number error is generated. The packet engine does not update the sequence number and sequence number mask fields in the SA. The host must discard the packet.

For inbound operations, with `PKTE_SA_CMD0.HDRPROC` disabled or `PKTE_SA_CMD1.ENSQNCHK` disabled, the packet engine does not verify the sequence number against the sequence number in the packet and therefore never generates a sequence number error.

With `PKTE_SA_CMD0.HDRPROC` disabled the packet engine does not update the sequence number and sequence number mask fields in the SA and expects the host to provide the sequence number through the input buffer as part of the header.

The following tables provide details of sequence number processing.

Table 44-17: Sequence Number Generation and Verification Control (Outbound)

Header Processing SA_CMD0[19]	Anti-Replay Service SA_CMD1[29]	Description
Outbound		
1	1	Sequence number generation (increment)
1	1	Sequence number overflow check (2^{32}–1) to zero for IPsec, (2^{48}–1) to zero for DTLS and (2^{64}–1) to zero for Extended SSL/TLS.
1	0/1	Sequence number update in SA
0/1	0/1	Extended sequence number update in SA

Table 44-18: Sequence Number Generation and Verification Control (Inbound)

Header Processing SA_CMD0[19]	Anti-Replay Service SA_CMD1[29]	Description
Inbound		
1	1	Sequence number verification (check against sequence number and sequence number mask in SA). Applicable only for IPsec and DTLS.
1	0/1	Sequence number update in SA, except on authentication error
1	1	Sequence number mask update in SA. Applicable only for IPsec and DTLS.
0/1	0/1	Extended sequence number update in SA

Table 44-19: Header Processing Enabled, Anti-replay Service Enabled

	IPsec ESP	Ext. SSL	Ext. TLS	DTLS
Header processing enabled, anti-replay service enabled				
Inbound				
Initial value in the SA SA_SEQNUM1[31:0] SA_SEQNUM0[31:0] SA_SEQNUM_MSK	B3 B0 0x00000000 0x00000000 not used	B3 B0 0x00000000 0x00000000 not used	B3 B0 0x00000000 0x00000000 not used	B3 B0 0xAA550000 0x00000000 not used
Value in the first packet or hashbyte stream, highest byte is B0	B0 B3 0x00000000 B4 B7 0x00000001	B0 B3 0x00000000 B4 B7 0x00000000	B0 B3 0x00000000 B4 B7 0x00000001	B0 B3 0xAA550000 B4 B7 0x00000001
Initial value in the SA after first packet SA_SEQNUM1[31:0] SA_SEQNUM0[31:0] SA_SEQNUM_MSK	B3 B0 0x00000000 0x00000001 not used	B3 B0 0x00000000 0x00000001 not used	B3 B0 0x00000000 0x00000001 not used	B3 B0 0xAA550000 0x00000001 not used
Initial value in the SA SA_SEQNUM1[31:0] SA_SEQNUM0[31:0] SA_SEQNUM_MSK	B3 B0 0xFEDCBA98 0x76543210 not used	B3 B0 0xFEDCBA98 0x76543210 not used	B3 B0 0xFEDCBA98 0x76543210 not used	B3 B0 0xAA55BA98 0x76543210 not used
Value in the first packet or hashbyte stream, highest byte is B0	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xFEDCBA98 B4 B7 0x76543211	B0 B3 0xAA55BA98 B4 B7 0x76543211
Initial value in the SA after the packet SA_SEQNUM_1[31:0] SA_SEQNUM_0[31:0] SA_SEQNUM_MASK	B3 B0 0xFEDCBA98 0x76543211 not used	B3 B0 0xFEDCBA98 0x76543211 not used	B3 B0 0xFEDCBA98 0x76543211 not used	B3 B0 0xAA55BA98 0x76543211 not used
Highest value before overflow SA_SEQNUM_1[31:0] SA_SEQNUM_0[31:0] SA_SEQNUM_MASK	B0 B3 0xFFFFFFFF 0xFFFFFFFF not used	B0 B3 0xFFFFFFFF 0xFFFFFFFF not used	B0 B3 0xFFFFFFFF 0xFFFFFFFF not used	B0 B3 0xAA55FFFF 0xFFFFFFFF not used
Value in the packet or hashbyte stream, highest byte is B0	B0 B3 0x00000000B4 B7 0x00000000	B0 B3 0xFFFFFFFF B4 B7 0xFFFFFFFF	B0 B3 0xFFFFFFFF B4 B7 0xFFFFFFFF	B0 B3 0xAA55FFFF B4 B7 0xFFFFFFFF

Table 44-19: Header Processing Enabled, Anti-replay Service Enabled (Continued)

	IPsec ESP	Ext. SSL	Ext. TLS	DTLS
Header processing enabled, anti-replay service enabled				
Inbound				
Value after overflow in SA	B0 B3	B0 B3	B0 B3	B0 B3
SA_SEQNUM_1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM_0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MASK	not used	not used	not used	not used
Initial value in SA	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK1[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK0[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
Expected value in the first packet or hash-byte stream, highest byte is B0	B0 B3	B0 B3	B0 B3	B0 B3
	0x00000000	0x00000000	0x00000000	0xAA550000
	B4 B7	B4 B7	B4 B7	B4 B7
	0x00000001	0x00000001	0x00000001	0x00000001
Value in SA after first packet	B3 B0	B3 B0	B3 B0	B3 B0
SA_SEQNUM1[31:0]	0x00000000	0x00000000	0x00000000	0xAA550000
SA_SEQNUM0[31:0]	0x00000001	0x00000001	0x00000001	0x00000001
SA_SEQNUM_MSK1[31:0]	0x00000000	0x00000000	0x00000000	0x00000000
SA_SEQNUM_MSK0[31:0]	0x00000001	0x00000001	0x00000001	0x00000001

PKTE Block Diagram

The ***PKTE Block Diagram*** shows the functional blocks within the PKTE.

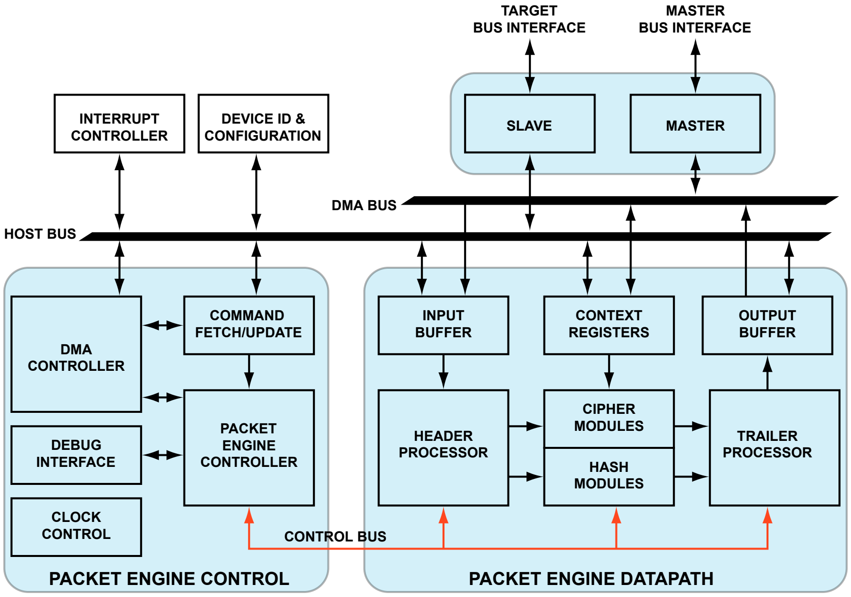

TARGET BUS INTERFACE
MASTER BUS INTERFACE
INTERRUPT CONTROLLER
DEVICE ID & CONFIGURATION
SLAVE
MASTER
DMA BUS
HOST BUS
DMA CONTROLLER
COMMAND FETCH/UPDATE
PACKET ENGINE CONTROLLER
DEBUG INTERFACE
CLOCK CONTROL
INPUT BUFFER
CONTEXT REGISTERS
OUTPUT BUFFER
HEADER PROCESSOR
CIPHER MODULES
HASH MODULES
TRAILER PROCESSOR
CONTROL BUS
PACKET ENGINE CONTROL
PACKET ENGINE DATAPATH

Figure 44-1: PKTE Block Diagram

PKTE Architectural Concepts

The following descriptions provide details on the functional blocks within the security packet engine.

Packet Engine

The packet engine contains symmetric cipher and hash engines. It is optimized to off-load intensive cryptographic operations from the host processor. It can perform parallel and pipelined encryption and hashing operations, reducing the latency, and processing time for packets that need both operations applied. The processor core provides the command information and packet data for the packet engine. The packet engine can run autonomously, using its local DMA controller to perform DMA transfers across the main bus to access the memory of the processor core. The DMA process incorporates flow-control to guarantee proper data flow. Two elements provide the command information that defines the processing for each packet:

- Command descriptor
- SA record

When the packet engine finishes the operation, it updates the SA record, when needed, and provides a result descriptor.

The packet engine has four different modes of operation. The modes give the processor core various levels of control over the command information and packet data transfers to and from the packet engine.

- Autonomous Ring Mode (ARM)
- Target Command Mode (with and without result descriptor ring) (TCM)
- Direct Host Mode (DHM)

Input/Output FIFO Buffers

The data for the packet engine is buffered at both input and output. These buffers decouple the DMA I/O process from the cipher and hash modules inside the packet engine. This functionality enables large DMA burst sizes and allows the crypto-engines to process data during I/O latency periods. Data moves automatically from the input buffer through the encryption and hash engines to the output buffer. If the output buffer is full, the process stops until the data is read and space is available in the output buffer. Each buffer is a 256-byte dual-port RAM.

Parallel Operations

The hash functionality and encrypt or decrypt functionality are tightly coupled. Operations that combine both hash and encrypt or decrypt functions are available to reduce processing time for data that must apply both. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of the hashing and the start of the encryption to support protocols such as IPSec and SRTP.

DMA Controller

The packet engine uses a high-performance DMA controller for autonomous data transfers for:

- Command descriptor reads
- SA record and state record reads
- Packet data read
- Result packet writes
- SA record and state record writes
- Result descriptor writes

Interrupt Controller

The packet engine includes an interrupt controller that, under programmable configuration control, can generate an interrupt on completion of certain operations. Individual interrupts can be masked and cleared. The interrupt registers show both the raw and masked interrupt status of the internal interrupts. The processor core can use interrupts, together with their associated threshold settings, to optimize the overall packet processing in the system. One interrupt can inform the processor core that the input side of the packet engine is almost empty to avoid a stall-on-empty condition. One interrupt can inform the host that the output side is almost full to avoid a stall-on-full condition. The controller uses several interrupts to inform the processor core about errors inside the packet engine. All available interrupts are combined into a single output port as either a level- or edge-active programmable interrupt output.

Clock Controller

The packet engine includes a clock controller that generates clock enable signals. A clock manager external to the packet engine uses the clock enable signals to switch the clocks to modules in the packet engine, reducing power consumption. The power saving can be significant, depending on the crypto-operation and the idle time of the packet engine. The clock controller generates the clock enable signals dynamically depending on the current crypto-operation. A clock control register provides the processor core the possibility to override this dynamic process.

PKTE Operating Modes

The packet engine can be configured in one of three command modes. For all modes the packet engine can generate an interrupt at completion of packet processing:

- Autonomous Ring Mode (ARM). The core prepares descriptors in the CDR and then initiates a descriptor fetch by triggering the packet engine. When a packet operation is complete, the packet engine writes the result descriptor out into a ring in host memory using the system master bus interface.
- Target Command Mode (TCM). The core directly writes the command descriptors to the packet command register set to initiate a packet operation. This process eliminates an extra DMA transfer to fetch a descriptor, but requires the core to synchronously initiate packet processing. This mode can be configured both without RDR or with RDR. In the latter case, the packet engine writes the result descriptor out into the RDR in host memory using the system master bus interface.
- Direct Host Mode (DHM). The core has full control over the packet engine and uses the system slave bus interface. The core provides all the command descriptors, SA record and state record, and packet input data. When the packet engine completes processing, the core must read the packet output data, the SA record, the state record, and the result descriptors.

Autonomous Ring Mode (ARM)

The *Autonomous Ring Mode* allows the packet engine and the host processor to operate asynchronously. A queue of multiple packets in the host processor memory can be processed continuously to provide the highest possible throughput. The packet engine autonomously fetches the command descriptor, the SA record, and optionally the state record and the input data from host processor memory. After the packet engine finishes processing, it autonomously writes the output data, updates the SA record and state record and writes the result descriptor in the host processor memory. It accesses the host processor memory through DMA read transfers across the system bus master.

This mode uses both command descriptor ring (CDR) and result descriptor ring (RDR).

Physically the CDR, RDR, SA record, source packet, and result packet can all be in different memories depending on the system memory architecture. The host processor writes command descriptors to the CDR in host processor memory. Then, it writes to the `PKTE_CDSC_INCR` register with the number of command descriptors that it prepared in the CDR. This write to the `PKTE_CDSC_INCR` register is the trigger for the packet engine to fetch the command descriptors sequentially from the CDR. When a command descriptor is fetched and written to the internal packet command register set, the descriptor is validated. If the ownership bits are set for the packet engine and the command is valid, processing starts. If not, that command is discarded and a result descriptor is written to the RDR with the error code *invalid command descriptor*. In this mode, the host processor can set a threshold on the CDR and enable an associated interrupt. The packet engine generates an interrupt when the number of command descriptors in the CDR is equal or below the threshold value.

The SA record and state record that contains the crypto context information are stored in a memory area. The packet engine autonomously accesses the memory area through DMA transfers across the system bus master. Also, the source packet and result packet are stored in a memory area that the packet engine autonomously accesses through the same bus master interface.

After decoding the command descriptor, the packet engine fetches the SA record and then, optionally, the state record.

Then, the source packet is fetched and stored in the input buffer. Packets less than the size of the input buffers are fetched entirely at once. Larger packets are fetched in parts that completely fill the input buffer. The packet engine initiates a new fetch each time the number of empty spaces in the input buffer reaches its threshold value. When the first packet data is available in the input buffer, the crypto engines start processing the data. After processing, the crypto engines write the result packet to the output buffer.

The packet engine writes the result packet from the output buffer to host processor memory when the number of bytes in the output buffer reaches its threshold value. Packets less than the threshold value are written entirely at once. Larger packets are written in parts that completely empty the output buffer

The source packet data fetching, data processing, and result packet data writing are parallel processes that continue until the last result packet is written to host processor memory. Then, the packet engine optionally writes the SA record and the state record to update the crypto context information. As a final step, the packet engine writes the result descriptor to the RDR. The host processor must either poll the RDR or wait for an interrupt from the packet engine to determine when packet processing is complete.

Target Command Mode (TCM)

This mode provides a synchronous interface between the processor core and the packet engine. The Command Descriptor Ring (CDR) is disabled and the processor core initiates packet processing by writing the command descriptor directly to the internal command descriptor MMRs of the packet engine. The Result Descriptor Ring (RDR) is optional.

- For `PKTE_CFG.MODE` =01, the RDR is disabled and the processor core reads the result descriptor directly from the internal result descriptor register set for the packet engine.
- For `PKTE_CFG.MODE` =10, the RDR is enabled and stored in a memory area that the packet engine can access through its master bus interface as in autonomous ring mode.

In target command mode, the packet engine autonomously fetches the SA record, state record, source packet data as in autonomous ring mode. Also, as in ARM, after processing, the packet engine updates state fields of the SA record and state record in the host processor memory.

Direct Host Mode (DHM)

This mode provides a synchronous interface between the processor core and the packet engine. The packet engine is under full control of the processor core. The host processor writes the command descriptors, SA record, and state record directly to the packet engine registers. Then, the processor core writes the source packet data into the input buffer. When processing is complete, the processor core reads back the result packet data from the output buffer.

Finally, it reads the result descriptor, the updated SA record, and state record directly from the packet engine registers.

PKTE Event Control

The following section provides information about interrupts in the PKTE module.

PKTE Interrupt Signals

The Packet Engine has an internal Interrupt Controller with 9 interrupt sources. There are 7 registers associated with the interrupt controller:

1. Interrupt Unmasked Status - `PKTE_IUMSK_STAT`
2. Interrupt Mask Status - `PKTE_IMSK_STAT`
3. Interrupt Clear Register - `PKTE_INT_CLR`
4. Interrupt Enable Register - `PKTE_INT_EN`
5. Interrupt Mask Disable - `PKTE_IMSK_DIS`
6. Interrupt Mask Enable - `PKTE_IMSK_EN`
7. Interrupt Configuration - `PKTE_INT_CFG`

The ***Packet Engine Interrupt Controller Block Diagram*** shows the blocks of the interrupt controller.

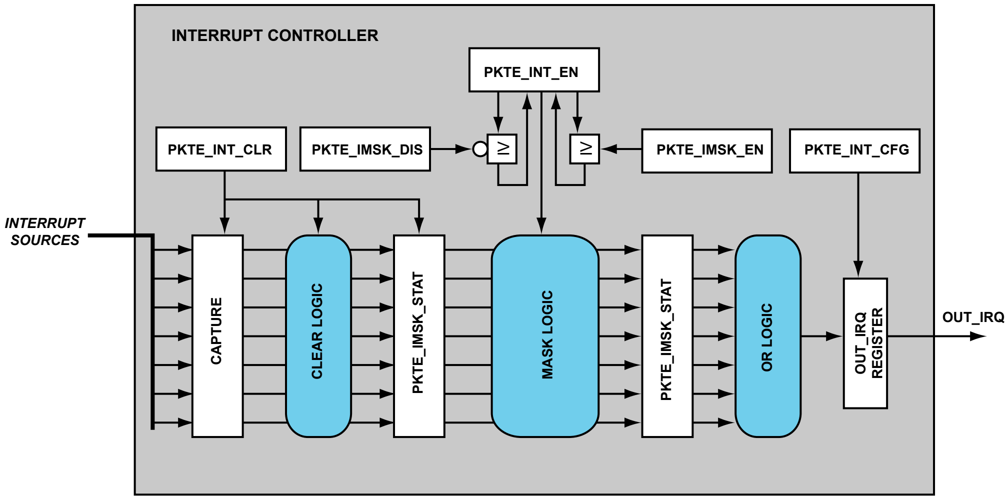

INTERRUPT CONTROLLER
PKTE_INT_EN
PKTE_INT_CLR
PKTE_IMSK_DIS
≥
≥
PKTE_IMSK_EN
PKTE_INT_CFG
INTERRUPT SOURCES
CAPTURE
CLEAR LOGIC
PKTE_IMSK_STAT
MASK LOGIC
PKTE_IMSK_STAT
OR LOGIC
OUT_IRQ REGISTER
OUT_IRQ

Figure 44-2: Packet Engine Interrupt Controller Block Diagram

All of the interrupt sources are pulse or level events in their native form.

These interrupts are captured and stored at their unmasked and masked status in their respective `PKTE_IUMSK_STAT` and `PKTE_IMSK_STAT` registers. This allows the host processor to read the status of any interrupt source either before or after the mask is applied.

The `PKTE_INT_EN` register provides a mask to select what interrupt source are enabled to the output interrupt request. Writing a one to the `PKTE_INT_CLR` register resets both the masked and unmasked interrupt.

The `PKTE_IMSK_EN` and `PKTE_IMSK_DIS` registers can be set to enable and disable individual interrupts respectively in the `PKTE_INT_EN` register. This avoids the need for read-modify-write operations from the host processor.

Ring Interrupts

Two interrupts are provided for efficient ring management: The CDR threshold interrupt (*cdrthrs*h) and the RDR threshold interrupt (*rdrthrsh*).

Command Descriptor Ring

The CDR threshold interrupt (cdrthrsh) is a level-based interrupt and connects to the threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit field. It enables the host processor to efficiently fill the CDR. The host processor writes command descriptors to the CDR with this interrupt masked until the CDR is full. Then the host processor enables the CDR threshold interrupt. When the interrupt is activated, the host processor clears the interrupt and it is guaranteed that it can put CDR threshold number of descriptors in the CDR.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.CDRTHRSH=224,
PKTE_INT_EN.CDRTHRSH=0 /*(IRQ disabled)*/
```

1. The host writes 8 Command Descriptors at once, then writes the `PKTE_CDSC_INCR` register to 8.
2. This is repeated until there are less than 8 empty entries in the CDR.
3. The fill level is now equal to he threshold (224)
4. The host enables the CDR threshold IRQ (`PKTE_INT_EN.CDRTHRSH=1`)
5. The packet engine processes packets and the fill level (`PKTE_CDSC_CNT` register) decreases.
6. Then the CDR threshold IRQ is activated as the fill level equals 224.
7. The host handles the interrupt, clears it and continues with step 1.

Result Descriptor Ring

The RDR threshold interrupt (rdrthrsh) is a level-based interrupt and connects to the threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit field and the timeout value in the RD timeout (`PKTE_RING_THRESH.RDTO`) bit field. It enables the host processor to efficiently empty the RDR. The timeout reminds the host processor that when the threshold kicks in result descriptors stay long in the RDR and must be processed to reduce latency. The timeout counts when the ring is not empty, regardless of the fill level and restarts when the host processor writes the `PKTE_RDSC_CNT` register. Initially the host processor enables the RDR threshold interrupt. When the interrupt is activated the host processor reads result descriptors until the RDR is empty or contains less than the `PKTE_RING_THRESH.RDRTHRSH` number of descriptors.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.RDRTHRSH=32, timeout=1ms
PKTE_INT_EN.RDRTHRSH=1     /*(IRQ enabled)*/
```

1. The Packet Engine writes the Result Descriptor, timeout counter starts.
2. The Packet Engine writes 32 more Result Descriptors, fill level (`PKTE_RDSC_CNT` register) increases to 33.
3. The fill level exceeds threshold within 1ms, the RDR threshold IRQ is activated.
4. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. The write to the `PKTE_RDSC_DECR` register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.
5. The Packet Engine writes 8 more Result Descriptors, fill level increases to 33.
6. The fill level exceeds threshold within 1 ms, the RDR threshold IRQ is activated.
7. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. The write to the `PKTE_RDSC_DECR` register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.
8. After 1 ms, the timeout counter interrupt is activated.
9. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the `PKTE_RDSC_DECR` register with 8. This is repeated until there are less than 8 full entries in the RDR. The fill level is now under the threshold and the RDR threshold IRQ interrupt is inactive. Each write to the `PKTE_RDSC_DECR` register restarts the timeout counter.

PKTE Programming Model

The host processor must always follow a pre-defined sequence of five phases required by the packet engine on a per packet basis when using direct host mode. The following sections describe the five phases.

Phase 1. Write the Command Descriptor

1. Write the first command descriptor word with status and control information to the `PKTE_CTL_STAT` register.
2. Optionally, write the user ID to the `PKTE_USERID` register.
3. Write the last descriptor word to the `PKTE_LEN` register.
4. Write the value 0x1 to the `PKTE_CDSC_CNT` register. This opertion triggers the packet engine to validate the command descriptor. If the command descriptor is invalid, an error is generated. (See the `PKTE_CTL_STAT` section in the Register Descriptions). If the command descriptor is valid, the packet engine waits for an `PKTE_SA_RDY` register write.

Phase 2. Write the State Registers, (ARC4 Buffer) and SA Registers

All required fields of the SA record and state record must be written. The fields required depend on the operation. The last field to be written is the `PKTE_SA_RDY` register. This register triggers the packet engine to start processing.

1. Write the required state record fields.
    - ARC4 state
    - IV
    - Digest count
    - State digest
2. Write the required SA record data.
3. To complete the SA record and state record, write the `PKTE_SA_RDY` register.

Phase 3. Write the Source Packet Data and Read Result Packet Data

The packet engine has input and output buffers. If a source packet is smaller than the size of the input buffer, then the packet can be written in one part. Otherwise, it must be written in multiple parts. The same applies to the output data. If the result packet size is smaller than the size of the output buffer, then the packet can be read in one part. Otherwise, it must be read in multiple parts.

> **NOTE:** An outbound packet that is smaller than the size of the input buffer can increase in size due to padding and does not always fit in the output buffer. Conversely, an inbound packet that is larger than the size of the input buffer can decrease in size, and due to de-padding, can fit in the output buffer. If the input buffer becomes empty or the output buffer becomes full, the engine stalls.

Two following steps describe different situations:

- Source packet smaller than the size of the input buffer, start at step 1.
- Source packet larger than the size of the input buffer, start at step 3.

The host processor must follow these steps:

1. Write the source packet data. Write the full source packet to the input buffer. Go to step 4.
2. Write the input buffer count register (`PKTE_INBUF_CNT`) with the number of valid bytes that are written to the input buffer. This value must correspond to the value in the`PKTE_LEN.TOTLEN` field of the command descriptor rounded up to the next multiple of 4 bytes. Go to step 5 to check the packet engine status.
3. Write part of the source packet data. The `PKTE_STAT.IBUFEMPTYCNT` field indicates the amount of free space in the input buffer. Programs write the number of bytes determined by the setting in the `PKTE_BUF_THRESH` register. Write the (partial) source packet to the input buffer. The host processor must resume where it ended the previous write operation. Do not write more than the buffer size at once. Go to step 4.

4. Write the PKTE_INBUF_CNT register with the number of valid bytes written to the input buffer. Go to step 5 to check the packet engine status.
5. Check packet engine status. Wait for an interrupt or poll the PKTE_STAT register for any of the following conditions:
   - Condition 1 - An error interrupt or any of the bits [7:5] in the PKTE_STAT register becomes active to indicate a packet processing error. Depending on the type of error, the host processor must take appropriate action. Usually, the result packet is not valid after a processing error has occurred. Go to phase 5 to read the result descriptor.
   - Condition 2 - An operation done interrupt or the PKTE_STAT.OPDN bit becomes active (the packet engine completed processing). Go to step 8 to read the remaining output data.
   - Condition 3 - An output buffer threshold interrupt or the PKTE_STAT.OBUFREQ bit becomes active. Go to step 6 to read a block of output data.
   - Condition 4 - An input buffer threshold interrupt or the PKTE_STAT.IBUFREQ bit becomes active. Go to step 3 to write a block of input data.
6. Read part of the output data. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of bytes in the output buffer. This value is rounded up to full words. Programs read the number of bytes indicated in the PKTE_BUF_THRESH register. Read the (partial) output packet from the output buffer. The host processor must resume where it ended the previous read operation. Do not read more than the input buffer size at once. Go to step 7.
7. Write the PKTE_OUTBUF_CNT register with the number of valid bytes read from the output buffer. Go to step 5 to check the packet engine status.
8. Read the remaining output data. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of bytes in the output buffer. This value is rounded up to words. Read the (partial) packet output data from the output buffer. The host processor must resume where it ended the previous read operation. Go to step 9.
9. Write the PKTE_OUTBUF_CNT register with the number of valid bytes read from the output buffer. Go to phase 4.

Phase 4. Read the Result Descriptor

1. Read the first result descriptor word from the PKTE_CTL_STAT.
2. Optionally read the user ID from the PKTE_USERID register.
3. Read the last result descriptor word from the PKTE_LEN register.
4. Write the value 0x1 to the PKTE_RDSC_DECR register. This operation allows the packet engine to accept new command descriptors. Go to phase 5.

Phase 5. Read the SA Record and State Record

Depending on the operation, the SA record or state record is updated. Check the bit fields [23:16] in the `PKTE_CTL_STAT` register for the following conditions:

- Condition 1 - At least one error bit in the bit fields [23:16] of the `PKTE_CTL_STAT` register is set. Do not update the local host processor maintained version of the SA record and state record but take any required action.
- Condition 2 - None of the error bits in the bit fields [23:16] of the `PKTE_CTL_STAT` register is set, the packet is processed normally (without errors). Update the host processor maintained version of the SA record and state record with the result read from the packet engine registers:
  - ARC4 state
  - Sequence number
  - Sequence number mask
  - Result IV
  - Result digest count
  - Result digest

PKTE Mode Configuration

Before using the packet engine, it must be configured. The mode of the packet engine must be defined and the PRNG (if used) must be initialized.

Configure the packet engine in one of three command modes:

- Autonomous Ring Mode: `PKTE_CFG.MODE` =b'11
- Target Command Mode: `PKTE_CFG.MODE` =b'10
- Direct Host Mode: `PKTE_CFG.MODE` =b'00

PKTE Programming Concepts

The following sections provide conceptual information for programming the PKTE.

Packet Engine Descriptor

IMPORTANT: Depending on the mode, ARM, TCM, or DHM, the descriptor is either:

- in the memory of the host processor in the command descriptor ring, or
- written directly to the descriptor registers in the packet engine

References to descriptor registers are for either the register that is mirrored in the descriptor structure in memory or for the actual register itself.

Command descriptors are host-supplied commands that control the real-time operation of the packet engine. The packet engine returns result descriptors at the end of an operation that provide the status information to the host. The ***Command Descriptor Structure*** and the ***Result Descriptor Structure*** tables show these descriptors.

Table 44-20: Command Descriptor Structure

<table>
<tr><th>Word Offset</th><th>31:24</th><th colspan="2">23:20</th><th>19:16</th><th>15:8</th><th>7:0</th><th>Address Offset</th></tr>
<tr><td>0</td><td>Pad Control</td><td colspan="3">—</td><td>Next Header/ Pad Value</td><td>Control</td><td>0x000</td></tr>
<tr><td>1</td><td colspan="6">Source Address</td><td>0x004</td></tr>
<tr><td>2</td><td colspan="6">Destination Address</td><td>0x008</td></tr>
<tr><td>3</td><td colspan="6">SA Address</td><td>0x00C</td></tr>
<tr><td>4</td><td colspan="6">SA State Address</td><td>0x010</td></tr>
<tr><td>5</td><td colspan="6">Reserved/ARC4 State Address</td><td>0x014</td></tr>
<tr><td>6</td><td colspan="6">User ID</td><td>0x018</td></tr>
<tr><td>7</td><td>Bypass (words)</td><td>Control</td><td>Reserved</td><td colspan="3">Input Packet Length (bytes)</td><td>0x01C</td></tr>
</table>

Table 44-21: Result Descriptor Structure

<table>
<tr><th>Word Offset</th><th>31:24</th><th colspan="2">23:20</th><th>19:16</th><th>15:8</th><th>7:0</th><th>Address Offset</th></tr>
<tr><td>0</td><td>Pad Status</td><td colspan="3">Status</td><td>Next Header/ Pad Value</td><td>Control</td><td>0x000</td></tr>
<tr><td>1</td><td colspan="6">Source Address</td><td>0x004</td></tr>
<tr><td>2</td><td colspan="6">Destination Address</td><td>0x008</td></tr>
<tr><td>3</td><td colspan="6">SA Address</td><td>0x00C</td></tr>
<tr><td>4</td><td colspan="6">SA State Address</td><td>0x010</td></tr>
<tr><td>5</td><td colspan="6">Reserved/ARC4 State Address</td><td>0x014</td></tr>
<tr><td>6</td><td colspan="6">User ID</td><td>0x018</td></tr>
<tr><td>7</td><td>Bypass (words)</td><td>Control</td><td>Reserved</td><td colspan="3">Input Packet Length (bytes)</td><td>0x01C</td></tr>
</table>

When the packet engine is configured for autonomous ring mode, command descriptors and result descriptors reside in a ring in host memory. Command descriptors are automatically fetched from the Command Descriptor Ring (CDR) through DMA into the command descriptor registers. When an operation is complete, the result descriptors are automatically read from the packet engine and through DMA to the Result Descriptor Ring (RDR).

When the packet engine is configured for direct host mode, the host processor manually writes the command descriptor directly to the internal command descriptor MMR set. When an operation is complete, the host processor manually reads the result descriptor directly from the result descriptor MMR set.

The target command mode is a combination of the direct host mode and the autonomous ring mode. The host processor writes the command descriptor directly to the internal command descriptor register set. When an operation is complete there are two options.

1. The host processor can read the result descriptor directly from the result descriptor registers.
2. The result descriptors reside in a ring in host memory and the result descriptors are automatically DMA'd from the packet engine to the RDR.

When the host processor writes a command descriptor to the command descriptor registers, the packet engine is triggered when the host processor updates the `PKTE_CDSC_CNT` register. This functionality guarantees that all fields in the command descriptor are valid before the command is executed.

Descriptor Processing

This section describes the functional steps of the packet engine while processing the command descriptors.

Descriptor Ring Configuration

At initialization, the host processor specifies the size of the Command Descriptor Ring (CDR). The Result Descriptor Ring (RDR) has the same size.

NOTE: In some configurations, these two rings overlay each other with the results written on top of the command descriptors. This configuration is called overlaid ring mode.

When the packet engine is configured and enabled, it fetches the descriptors from the CDR using system bus master reads.

Descriptor Ring Processing

To validate the descriptor exchange between the host processor and the packet engine, the ownership bits `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` are used. One pair of ownership bits is in the first word of the descriptor (`PKTE_CTL_STAT`), and one pair is in the last word (`PKTE_LEN`). The ‘consumer’ of a descriptor must verify that both ownership pairs match to ensure that a race condition did not occur between one party writing and the other party reading the descriptor. A race condition can occur when a memory locking scheme is not used.

Each pair [`PKTE_CTL_STAT.PERDY`, `PKTE_CTL_STAT.HOSTRDY`] of ownership bits provide 3 states:

- b'00 = idle or null descriptor
- b'01 = host processor has written a descriptor in the CDR and passed ownership to the packet engine
- b'10 = packet engine processing complete: packet engine has written the descriptor in the RDR and passed ownership back to the host.
- b'11 = Reserved

At initialization, the host sets the entire CDR memory area to zero, when the CDR is used.

1. The host processor writes one or more command descriptors to the CDR. The host processor must set the `PKTE_CTL_STAT.HOSTRDY` bit to 1 and the `PKTE_CTL_STAT.PERDY` bit to 0 to indicate that ownership has passed to the packet engine. These bits are mirrored in the `PKTE_LEN` descriptor word.
2. The host processor must write the `PKTE_CDSC_INCR` register with the number of new valid command descriptors in the CDR.
3. The packet engine reads and validates one command descriptor.
4. The packet engine reads the SA record and state record, processes the packet and updates the SA record and state record.
5. If the rings are not overlaid and the `PKTE_CFG.ENCDRUPDT` bit is 1, the packet engine writes the result descriptor to the CDR with the `PKTE_CTL_STAT.HOSTRDY` bit set to 0 and `PKTE_CTL_STAT.PERDY` bit set to 1. These bits are mirrored in the `PKTE_LEN` descriptor word.
6. The packet engine writes the result descriptor to the RDR. The packet engine sets the `PKTE_CTL_STAT.PERDY` bit to 1 and the `PKTE_CTL_STAT.HOSTRDY` bit to 0 to indicate that the ownership has passed to the host. These bits are mirrored in the `PKTE_LEN` descriptor word.
7. The packet engine decrements the value in the `PKTE_CDSC_CNT` register. If the value is not zero, the packet engine reads with the next command descriptor (step 3).
8. The packet engine increments the value in the `PKTE_RDSC_CNT` register.
9. The host processor reads one or more result descriptors from the RDR and processes the results.
10. The host processor must write the `PKTE_RDSC_DECR` register with the number of processed result descriptors in the RDR.

Ownership of the Descriptor

The ownership of the command descriptor and result descriptor is set by ownership bits in the first and last word of the respective descriptor, in the `PKTE_CTL_STAT` register and the `PKTE_LEN` register. For the command descriptor, it is the processor core that sets the ownership to the packet engine. For the result descriptor, it is the packet engine that sets the ownership to the host processor.

The packet engine reads the ownership bits before processing, irrespective of the mode of the packet engine. The ownership bits are used to validate and identify the descriptor. When two separate rings are used, the packet engine can be programmed to clear the ownership bits of the command descriptors in the CDR so the host processor knows which descriptors are processed.

NOTE: This update of the ownership bits can be disabled in the `PKTE_CFG` register when the host processor actively counts the number of descriptors in the CDR to prevent ring wrapping.

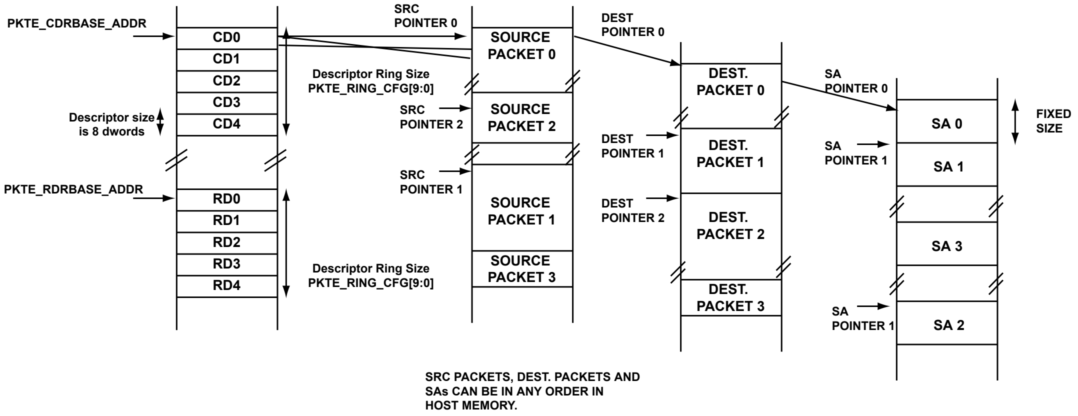

PKTE_CDRBASE_ADDR
CD0
CD1
CD2
CD3
CD4
Descriptor size is 8 dwords
PKTE_RDRBASE_ADDR
RD0
RD1
RD2
RD3
RD4
Descriptor Ring Size PKTE_RING_CFG[9:0]
Descriptor Ring Size PKTE_RING_CFG[9:0]
SRC POINTER 0
SRC POINTER 2
SRC POINTER 1
SOURCE PACKET 0
SOURCE PACKET 2
SOURCE PACKET 1
SOURCE PACKET 3
DEST POINTER 0
DEST POINTER 1
DEST POINTER 2
DEST. PACKET 0
DEST. PACKET 1
DEST. PACKET 2
DEST. PACKET 3
SA POINTER 0
SA POINTER 1
SA POINTER 1
SA 0
SA 1
SA 3
SA 2
FIXED SIZE
SRC PACKETS, DEST. PACKETS AND SAs CAN BE IN ANY ORDER IN HOST MEMORY.

Figure 44-3: Descriptor Rings in Autonomous Ring Mode

Description and Use of the SA Record and State Record Structure

The SA record is a packed structure that contains the remainder of the information needed by the packet engine to process a packet. Most of the information fields in the SA record, such as the key and encryption mode, are static for the lifetime of the association. The fields do not require frequent manipulation by the host processor. The SA record non-static fields are the sequence number and sequence number mask.

The SA record can have a corresponding state record that is used to save results from the current operations that can be used for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

If an SA record is used for operations that use ARC4 processing in stateful mode, it has a corresponding ARC4 state record that holds the ARC4 State.

In this manual, the state record and the ARC4 state record are referred to as state record.

There is no practical limit to how many SA records and corresponding state records the packet engine can support.

In the autonomous ring mode and target command mode, once the packet engine has validated a command descriptor, it automatically fetches the SA record and optional state record. After processing, the packet engine updates the stateful fields in the SA record and state record in the host processor memory.

In direct host mode, after the descriptor is validated, the host must write the SA record directly into the internal registers of the packet engine. After processing, the host reads the stateful fields from the SA registers of the packet engine and saves them back to the SA record in the host processor memory.

SA Record Structure

The *SA Record Structure* table shows the structure for an SA Record. When using direct host mode, the corresponding elements are accessed directly with the registers. When using autonomous ring mode or target command mode, the SA Record is defined, configured and accessed in host memory.

Table 44-22: SA Record Structure

Word Offset	Description (name)	Use
0	PKTE_SA_CMD0[31:0]	SA Control word 0 (all operations)
1	PKTE_SA_CMD1[31:0]	SA Control word 1 (all operations)
2	PKTE_SA_KEY0[31:0]	Key word (DES, Triple-DES, AES-128/192/256, ARC4,)
3	PKTE_SA_KEY1[63:32]	Key word (DES, Triple-DES, AES-128/192/256, ARC4,)
4	PKTE_SA_KEY2[95:64]	Key word (Triple-DES, AES-128/192/256, ARC4,)
5	PKTE_SA_KEY3[127:96]	Key word (Triple-DES, AES-128/192/256, ARC4,)
6	PKTE_SA_KEY4[159:128]	Key word (Triple-DES, AES-192/256)
7	PKTE_SA_KEY5[191:160]	Key word (Triple-DES, AES-192/256)
8	PKTE_SA_KEY6[223:192]	Key word (AES-256)
9	PKTE_SA_KEY7[255:224]	Key word (AES-256)
10	PKTE_SA_IDIGEST0[31:0]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
11	PKTE_SA_IDIGEST1[63:32]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
12	PKTE_SA_IDIGEST2[95:64]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
13	PKTE_SA_IDIGEST3[127:96]	Inner Hash digest (Basic Hash and HMAC with MD5,SHA-1, SHA-224, SHA-256)
14	PKTE_SA_IDIGEST4[159:128]	Inner Hash digest (Basic Hash and HMAC with SHA-1, SHA-224, SHA-256)
15	PKTE_SA_IDIGEST5[191:160]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
16	PKTE_SA_IDIGEST6[223:192]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
17	PKTE_SA_IDIGEST7[255:224]	Inner Hash digest (Basic Hash and HMAC with SHA-256)
18	PKTE_SA_ODIGEST0[31:0]	Outer Hash digest (HMAC with MD5,, SHA-1, SHA-224, SHA-256)
19	PKTE_SA_ODIGEST1[63:32]	Outer Hash digest (HMAC with MD5,, SHA-1, SHA-224, SHA-256)
20	PKTE_SA_ODIGEST2[95:64]	Outer Hash digest (HMAC with MD5,, SHA-1, SHA-224, SHA-256)

Table 44-22: SA Record Structure (Continued)

Word Offset	Description (name)	Use
21	PKTE_SA_ODIGEST3[127:96]	Outer Hash digest (HMAC with MD5,, SHA-1, SHA-224, SHA-256)
22	PKTE_SA_ODIGEST4[159:128]	Outer Hash digest (HMAC with SHA-1, SHA-224, SHA-256)
23	PKTE_SA_ODIGEST5[191:160]	Outer Hash digest (HMAC with SHA-224, SHA-256)
24	PKTE_SA_ODIGEST6[223:192]	Outer Hash digest (HMAC with SHA-224, SHA-256)
25	PKTE_SA_ODIGEST7[255:224]	Outer Hash digest (HMAC with SHA-256)
26	PKTE_SA_SPI[31:0]	SPI (IPsec), Type[23:16] / Version [15:0] (SSL, TLS, DTLS)
27	PKTE_SA_SEQNUM0[31:0]	Sequence Number (IPsec, SSL, TLS, DTLS with Header Processing)
28	PKTE_SA_SEQNUM1[63:32]	
29	PKTE_SA_SEQNUM_MSK0[31:0]	Sequence Number Mask (IPsec, DTLS inbound with Header Processing)
30	PKTE_SA_SEQNUM_MSK1[63:32]	
31	PKTE_SA_NONCE[31:0] / PKTE_SA_READY	Nonce value (AES-CTR, AES-ICM)/ARC4, i & j pointers (ARC4,)/SA ready indicator (Direct Host Mode)

Some of these fields may be updated by the packet engine. These include:

- PKTE_SA_SEQNUM0
- PKTE_SA_SEQNUM1
- PKTE_SA_SEQNUM_MSK0
- PKTE_SA_SEQNUM_MSK1

All the other fields remain unchanged.

SA State Structure

The security association state structure contains information that may be updated after each packet, such as the IV and the intermediate hash result. The ***SA State Structure*** table shows the SA state structure and usage. In direct host mode, the elements are accessed directly using the PKTE registers. In target command mode and autonomous ring mode, this structure is defined and updated in host memory.

Table 44-23: SA State Structure

Word Offset	Description (name)	Use
0	PKTE_STATE_IV0[31:0]	Initialization Vector (DES, Triple DES, AES)

Table 44-23: SA State Structure (Continued)

Word Offset	Description (name)	Use
1	PKTE_STATE_IV1[63:32]	Initialization Vector (DES, Triple DES, AES)
2	PKTE_STATE_IV2[95:64]	Initialization Vector (AES)
3	PKTE_STATE_IV3[127:96]	Initialization Vector (AES)
4	PKTE_STATE_BYTE_CNT0[31:0]	Current hash byte count (MD5, SHA-1, SHA-224, SHA-256)
5	PKTE_STATE_BYTE_CNT1[63:32]	Current hash byte count (MD5, SHA-1, SHA-224, SHA-256)
6	PKTE_STATE_IDIGEST0[31:0]	Inner Hash digest (mirror of PKTE_SA_IDIGEST0)
7	PKTE_STATE_IDIGEST1[63:32]	Inner Hash digest (mirror of PKTE_SA_IDIGEST1)
8	PKTE_STATE_IDIGEST2[95:64]	Inner Hash digest (mirror of PKTE_SA_IDIGEST2)
9	PKTE_STATE_IDIGEST3[127:96]	Inner Hash digest (mirror of PKTE_SA_IDIGEST3)
10	PKTE_STATE_IDIGEST4[159:128]	Inner Hash digest (mirror of PKTE_SA_IDIGEST4)
11	PKTE_STATE_IDIGEST5[191:160]	Inner Hash digest (mirror of PKTE_SA_IDIGEST5)
12	PKTE_STATE_IDIGEST6[223:192]	Inner Hash digest (mirror of PKTE_SA_IDIGEST6)
13	PKTE_STATE_IDIGEST7[255:224]	Inner Hash digest (mirror of PKTE_SA_IDIGEST7)

ARC4 State Structure

The ***ARC4 State Structure*** table describes the state structure used with ARC4. When using the PKTE in direct host mode, these fields are accessed with the registers, starting at the value in the `PKTE_ARC4STATE_BUF` register. When using the PKTE in autonomous ring mode or target command mode, these fields are defined and accessed in a structure in host memory.

Table 44-24: ARC4 State Structure

Word Offset	Description (name)*1	Use
0	PKTE_ARC4_STATE0[3:0]*2	ARC4 (Basic, SSL and TLS)
1	PKTE_ARC4_STATE1[7:4]	
...	...	
62	PKTE_ARC4_STATE2[251:248]	
63	PKTE_ARC4_STATE3[255:252]	

*1 The indices in these fields indicate bytes.

*2 There are no corresponding named registers for these fields. PKTE_ARC4_STATE0 corresponds to `PKTE_ARC4STATE_BUF` and PKTE_ARC4_STATE1 corresponds to the 32-bit register following `PKTE_ARC4STATE_BUF` and so on.

Configuring Operations in the PKTE

The operation (cipher, hash function, and others) that the PKTE performs is configured primarily in the `PKTE_SA_CMD0` register. The following sections include a series of tables to help configure the least significant 16 bits of the `PKTE_SA_CMD0` register. These fields include:

- The operation code field (`PKTE_SA_CMD0.OPCD`)
- The direction field (`PKTE_SA_CMD0.DIR`)
- The operation group field (`PKTE_SA_CMD0.OPGRP`)
- The padding type (`PKTE_SA_CMD0.PADTYPE`)
- The cipher selection (`PKTE_SA_CMD0.CIPHER`)
- The hash selection (`PKTE_SA_CMD0.HASH`)

Basic Operations and Decoding

Table 44-25: Basic Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b00	0	0b000	Encrypt	0b00	1	0b000	Decrypt
0b00	0	0b001	Encrypt - Hash	0b00	1	0b001	Hash - Decrypt
0b00	0	0b010	Reserved	0b00	1	0b010	Reserved
0b00	0	0b011	Hash	0b00	1	0b011	Hash
0b00	0	0b100... 0b110	Reserved	0b00	1	0b100... 0b110	Reserved
0b00	0	0b111	PRNG	0b00	1	0b111	Reserved

Table 44-26: Protocol Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b01	0	0b000	ESP Outbound	0b01	1	0b000	ESP Inbound
0b01	0	0b001... 0b011	Reserved	0b01	1	0b001	Reserved
0b01	0	0b100	Basic SSL Outbound	0b01	1	0b010	Basic SSL Inbound
0b01	0	0b101	Basic TLS Outbound	0b01	1	0b011	Basic TLS Inbound

Table 44-26: Protocol Operation Decoding (Continued)

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b01	0	0b110	Reserved	0b01	1	0b100...0b110	Reserved
0b01	0	0b111	SRTP Outbound	0b01	1	0b111	SRTP Inbound

NOTE: For SSL/TLS and SRTP, no header processing is performed in hardware.

Table 44-27: Extended Protocol Operation Decoding

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b110	0	0b000	Reserved	0b11	1	0b000	Reserved
0b110	0	0b001	DTLS Outbound	0b11	1	0b001	DTLS Inbound
0b110	0	0b010...0b011	Reserved	0b11	1	0b010...0b011	Reserved
0b110	0	0b100	Ext. SSL Outbound	0b11	1	0b100	Ext. SSL Inbound
0b110	0	0b101	Ext. TLS v1.0 Outbound	0b11	1	0b101	Ext. TLS v1.0 Inbound
0b110	0	0b110	Ext. TLS v1.1 Outbound	0b11	1	0b110	Ext. TLS v1.1 Inbound
0b110	0	0b111	Reserved	0b11	1	0b111	Reserved

Error Code Description

The `PKTE_CTL_STAT` register is used to configure the packet engine for processing in Direct Host Mode (DHM) or Target Command Mode (TCM). The `PKTE_CTL_STAT` structure element in memory is used when the packet engine is configured for Autonomous Ring Mode (ARM). In both cases, when an operation is started, errors are reported in the status field (bits [23:16]) of this register or structure element. The ***Extended Error Codes - Status Encoding*** table provides a guide on how to decipher the meaning of the bits that are set when an error occurs.

Extended Error Codes

The following table provides information about the extended errors associated with the PKTE module.

Table 44-28: Extended Error Codes - Status Encoding

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0000_0000	0x00	NA	Successful completion. No errors occurred during processing of the packet.	Packet fully processed
0b----_---1	0x-1	NA	Authentication Error. For an inbound IPsec ESP operation, the Integrity Check Value (ICV) does not match the computed value. For an inbound SRTP operation, the authentication tag does not match the computed value. For a basic SSL/TLS, Extended SSL/TLS or DTLS operation the Message Authentication Code (MAC) does not match the computed value.	Packet fully processed
0b----_--1-	0x-2	NA	Pad Verify Error. For inbound operations that use pad type Constant TLS, IPsec or PKCS#7, the decrypted pad does not match the expected values for the selected pad type.	Packet fully processed
0b----_-1--	0x-4	NA	Sequence Number Error. For an inbound IPsec or DTLS operation, there was a fault in the Anti-Replay Sequence Number. For an outbound IPsec packet, the sequence number overflows; count is 2^{32}– 1 and increments to 0. For an outbound DTLS operation, the sequence number overflows; count is 2^{48}– 1 and increments to 0. For an outbound SSL or TLS operation, the sequence number overflows; count is 2^{64}– 1 and increments to 0.	Packet fully processed
0b0000_1---	0x08	1	System Bus error. The master bus interface generates an error due to ERROR response from system slave. The slave bus interface generates an error due to request for non-word (32-bit) access.	Packet is aborted. The host must reject the packet and apply a hardware reset to the system.
0b0001_1---	0x18	2	Invalid Command Descriptor Error. The ownership bits in the command descriptor are not set to the packet engine, after the `PKTE_CDSC_CNT` register is incremented.	Command descriptor is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0010_1---	0x28	3	Invalid Crypto Operation Error. A reserved operation is selected.	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.

Table 44-28: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0011_1---	0x38	4	Invalid Crypto Algorithm Error. A reserved cipher is selected, refer to `PKTE_SA_CMD0.CIPHER`. A reserved hash is selected, refer to `PKTE_SA_CMD0.HASH`.	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0100_1---	0x48	5	SPI Error. On an inbound packet, the 32-bit SPI value in the packet does not match the value in the SA while header processing is enabled. Note: A failure caused by an SPI mismatch, in general should not occur because the host checks the SPI and does not send an incorrect SPI to the packet engine.	Packet is fully processed. The host must reject the packet.
0b0101_1---	0x58	3	Zero Length Error. The packet length defined in the command descriptor `PKTE_LEN.TOTLEN` is zero, which is illegal.	Packet command is ignored, no packet processed. The host must reject the packet.
0b0110_1xxx	0x68	6	Invalid Packet Length Error. For Basic Encrypt-Hash and Hash-Decrypt operations: `PKTE_LEN.TOTLEN` < Hash/Encrypt Offset For IPsec ESP inbound operations: `PKTE_LEN.TOTLEN` < ICV length or `PKTE_LEN.TOTLEN` is non-4 byte aligned For SRTP inbound operations: `PKTE_LEN.TOTLEN` ≤ IV (opt.) + Bypass Offset + ROC For SSL inbound operations: `PKTE_LEN.TOTLEN` ≤ 1 or packet length > 65535 bytes (SSL packet-bypass length) For TLS and DTLS inbound operations: `PKTE_LEN.TOTLEN` ≤ 13 or payload length > 65535 bytes (data to be hashed) Note: For IPsec ESP the ICV is stripped before the length is checked.	Packet processing is aborted. Result packet length is zero. The host must reject the packet and apply a software reset of the packet engine.

Table 44-28: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0111_1xxx	0x78	7	Block Size Error. The length of the inbound packet defined in the Command Descriptor `PKTE_LEN.TOTLEN` is not a multiple of the DES or AES block cipher length. For outbound packets the size is always automatically aligned (padded) to the correct block size. The hashed packet length is not a multiple of the hash block size for intermediate hash operation. For a final hash operation no error is generated. Note: For IPsec ESP operations the ICV is stripped before the block size is checked.	Packet is fully processed. The host must reject the packet.
0b1000_1xxx	0x88	8	Processing Error. The number of bytes in the input buffer is more than defined in the `PKTE_LEN.TOTLEN` field. The number of bytes written to the output buffer is less than processed in the datapath.	Packet processing aborted. Result packet length is zero. The host must reject the packet and apply a software reset.
0b1010_1xxx 0b1111_1xxx	Reserved			

Number Format

When dealing with cryptographic functions, data and keys are large vectors. For instance, AES supports keys of sizes 128, 192, and 256 bits. When a key needs to be loaded or read, multiple 32-bit key registers are used, namely `PKTE_SA_KEY[n]` registers. The first key register `PKTE_SA_KEY[n]`0 holds bits 31:0, and `PKTE_SA_KEY[n]`1 holds the next thirty two bits 63:32, and so on.

Generally, for large vectors defined as Byte0, Byte1, Byte2, Byte3 and so on, the values are stored in the PKTE registers as PKTE_REG0 = Byte3Byte2Byte1Byte0, followed by PKTE_REG1 = Byte7Byte6Byte5Byte4 and so on.

PKTE Programming Examples

Use these examples to extend your understanding of PKTE features, operating modes, event control, and programming modes.

Calculating SHA in Direct Host Mode

This section describes how to configure the packet engine to calculate a hash digest using one of supported SHA algorithms in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the `PKTE_CFG.MODE` bit =0

2. Set the ownership back to the packet engine to process a command descriptor by setting the `PKTE_CTL_STAT.PERDY` bit =0 and the `PKTE_CTL_STAT.HOSTRDY` bit =1.
3. Set the `PKTE_CTL_STAT.HASHFINAL` bit to indicate this command descriptor handles all the input data for the hash calculation. This configuration is needed for the packet engine because the last block requires special handling (see FIPS 180-4 for details).
4. Set size of the input data in bytes in the `PKTE_LEN.TOTLEN` bit field.
5. Also set the `PKTE_LEN.PEDONE` bit =0 and the `PKTE_LEN.HSTRDY` bit =1. These bits must be the same as the `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` bits to guarantee ownership.
6. Set the `PKTE_CDSC_CNT` register =1 to trigger the packet engine to start validating the command descriptor. In this case, the `PKTE_CTL_STAT`, `PKTE_LEN` and `PKTE_CDSC_CNT` registers are the only command descriptor registers modified.
7. Configure the `PKTE_SA_CMD0` and `PKTE_SA_CMD1` registers to define the operation. For an SHA, set the `PKTE_SA_CMD0.OPCD` bit field =0b011 for hash operation and the `PKTE_SA_CMD0.OPGRP` bit field =0b00 for basic operation.
8. Select the specific SHA function using the `PKTE_SA_CMD0.HASH` bit field as follows.
   - For SHA-1, `PKTE_SA_CMD0.HASH` =0b0001
   - SHA-224, `PKTE_SA_CMD0.HASH` =0b0010
   - for SHA-256, `PKTE_SA_CMD0.HASH` =0b0011
9. Depending on the SHA selected, the appropriate digest length must be chosen for the `PKTE_SA_CMD0.DIGESTLEN` bit field as follows.
   - For SHA-1, `PKTE_SA_CMD0.DIGESTLEN` =0b0101 (5 words)
   - For SHA-224, `PKTE_SA_CMD0.DIGESTLEN` =0b0111 (7 words)
   - For SHA-256, `PKTE_SA_CMD0.DIGESTLEN` =0b1000 (8 words)
10. The SHA specifies initial constants. These constants can be pre-loaded or read from memory. In this example, by setting the `PKTE_SA_CMD0.HASHSRC` bit field =0b11, the packet engine provides the correct initial constants depending on the SHA chosen.
11. Next, set the `PKTE_SA_CMD1.CPYDGST` bit =1 and `PKTE_SA_CMD1.CPYPAD` bit =1 to move the result to the output buffer of the packet engine at the `PKTE_DATAIO_BUF` location.
12. At this point, write to the `PKTE_SA_RDY` register with any value to trigger the operation.
13. Start writing the input to the data buffer of the packet engine starting at the `PKTE_DATAIO_BUF` location.
14. Write the `PKTE_INBUF_CNT` register with the length of the input rounded up to the next multiple of 4. For example, if the input length is 30 bytes, set this register to 32.
15. Poll the `PKTE_STAT` register to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the digest is available in the packet engine data I/O buffer.

NOTE: The input data or message is input into the packet engine data buffer in big endian format while the result or digest is little endian format.

Performing AES Decryption in Direct Host Mode

This section describes how to configure the packet engine to decrypt using AES-128 in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the `PKTE_CFG.MODE` bit =0
2. Start configuring the command descriptor registers. Set the ownership back to the packet engine to process a command descriptor by setting the `PKTE_CTL_STAT.PERDY` bit =0 and the `PKTE_CTL_STAT.HOSTRDY` bit =1.
3. Next, configure the `PKTE_LEN.TOTLEN` bit field with the size of the packet or message to decrypt. If the entire input message (cipher text) fits into the 256-byte data I/O buffer of the packet engine, the process can be done in one shot.
4. Set the `PKTE_LEN.PEDONE` bit =0 and the `PKTE_LEN.HSTRDY` bit =1. These bits must have the same setting as the `PKTE_CTL_STAT.PERDY` and `PKTE_CTL_STAT.HOSTRDY` bits to guarantee ownership.
5. Set the `PKTE_CDSC_CNT` register =1 to trigger the packet engine to start validating the command descriptor. In this case, the `PKTE_CTL_STAT`, `PKTE_LEN`, and `PKTE_CDSC_CNT` registers are the only command descriptor registers modified.
6. Next, configure the `PKTE_SA_CMD0` and `PKTE_SA_CMD1` registers to define the operation.
   - For a AES decrypt inbound cipher operation, set the `PKTE_SA_CMD0.OPCD` bit field =0b000 and the `PKTE_SA_CMD0.DIR` bit field =0b1.
   - Set the`PKTE_SA_CMD0.OPGRP` bit field =0b00 for basic operation.
   - To choose the AES cipher, set the `PKTE_SA_CMD0.CIPHER` bit field =0b0011. Set the `PKTE_SA_CMD0.HASH` bit field to 0b1111 to choose the NULL function.
7. Next, set the `PKTE_SA_CMD1.AESKEYLEN` bit field to select the appropriate key length. In this case, setting it to 0b10 select 128 bits. Also, set `PKTE_SA_CMD1.CIPHERMD` bit field to select the mode. In this case, setting it to 0b01 select CBC mode.
8. Continue configuring the Security Association (SA) record by loading the key in the `PKTE_SA_KEY[n]` registers.
9. Next load the initialization vector in the SA state registers (`PKTE_STATE_IV[n]`).
10. Finally, write anything in to the `PKTE_SA_RDY` register to trigger the operation.
11. The input data can now be written into the data I/O buffer starting at `PKTE_DATAIO_BUF`.

12. After the data is written, write the length (or next multiple of 4) into `PKTE_INBUF_CNT` register.
13. Poll `PKTE_STAT` to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the result can be found in the same data I/O buffer.

ADSP-SC58x PKTE Register Descriptions

Security Packet Engine (PKTE) contains the following registers.

Table 44-29: ADSP-SC58x PKTE Register List

Name	Description
PKTE_ARC4STATE_ADDR	Packet Engine ARC4 State Record Address
PKTE_ARC4STATE_BUF	Starting Entry of 256-byte ARC4 State Buffer
PKTE_BUF_PTR	Packet Engine Buffer Pointer Register
PKTE_BUF_THRESH	Packet Engine Buffer Threshold Register
PKTE_CDRBASE_ADDR	Packet Engine Command Descriptor Ring Base Address
PKTE_CDSC_CNT	Packet Engine Command Descriptor Count Register
PKTE_CDSC_INCR	Packet Engine Command Descriptor Count Increment Register
PKTE_CFG	Packet Engine Configuration Register
PKTE_CLK_CTL	PE Clock Control Register
PKTE_CONT	PKTE Continue Register
PKTE_CTL_STAT	Packet Engine Control Register
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register

Table 44-29: ADSP-SC58x PKTE Register List (Continued)

Name	Description
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_ARC4IJPTR	ARC4 i and j Pointer Register
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

Packet Engine ARC4 State Record Address

The `PKTE_ARC4STATE_ADDR` register holds the start address of the SA ARC4 state record.

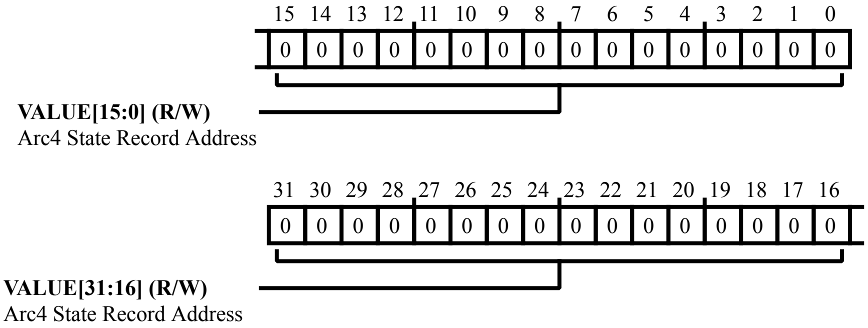

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Arc4 State Record Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Arc4 State Record Address

Figure 44-4: PKTE_ARC4STATE_ADDR Register Diagram

Table 44-30: PKTE_ARC4STATE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Arc4 State Record Address.

Starting Entry of 256-byte ARC4 State Buffer

The `PKTE_ARC4STATE_BUF` register is used to store the pre-processed key that initializes the ARC4 module. In direct host mode, before processing starts, the Host must write the ARC4 state, starting from the base address and increment the address pointer for each write. When processing completes the Host must read the ARC4 state and copy it to the local Host maintained state record. After a reset, a read from any address in the address range of the ARC4 buffer returns an undefined value.

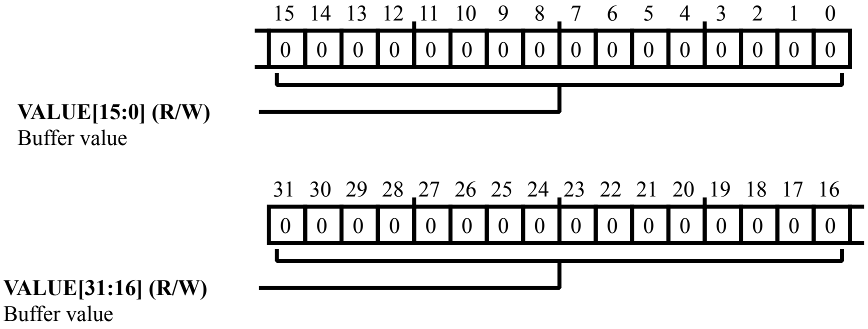

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Buffer value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Buffer value

Figure 44-5: PKTE_ARC4STATE_BUF Register Diagram

Table 44-31: PKTE_ARC4STATE_BUF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Buffer value. The `PKTE_ARC4STATE_BUF.VALUE` bit field stores the pre-processed key that initializes the ARC4 module.

Packet Engine Buffer Pointer Register

The PKTE_BUF_PTR register contains the offset of the next buffer address (entry) to be read or written by the packet engine. This register is used in direct host mode only.

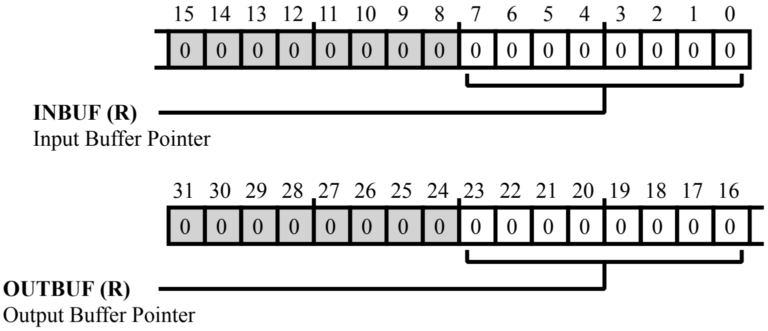

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INBUF (R)
Input Buffer Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OUTBUF (R)
Output Buffer Pointer

Figure 44-6: PKTE_BUF_PTR Register Diagram

Table 44-32: PKTE_BUF_PTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	OUTBUF	Output Buffer Pointer. The PKTE_BUF_PTR.OUTBUF bit field indicates the offset of the next address (entry) in the output buffer that will be written next by the packet engine. This bit field is reset to zero after starting up and decremented by 4 at every output buffer write operation. Pointers wrap around; the maximum value this field can have equals the output buffer size minus 4.
7:0 (R/NW)	INBUF	Input Buffer Pointer. The PKTE_BUF_PTR.INBUF bit field indicates the offset of the next address (entry) in the input buffer that will be read next by the packet engine. The bit field is reset to zero after starting up and incremented by 4 at every input buffer read operation. Pointers wrap around; the maximum value this field can have equals the input buffer size minus 4.

Packet Engine Buffer Threshold Register

When in autonomous ring mode or target command mode, the `PKTE_BUF_THRESH` register defines the high- and low-level value at which the packet engine starts to transfer packet data in or out of the internal packet buffers. These parameters can be used to control the DMA burst size for packet data input and output from the packet engine. In direct host mode, this register contains both threshold values to reduce the amount of packet engine interrupts.

The input buffer threshold (ibufthrsh) interrupt indicates that the input buffer counter is less than or equal to the input buffer threshold value set in this register - this interrupt can be used to wake up a process that stalled on a full input buffer.

The output buffer threshold (obufthrsh) interrupt indicates that the output buffer counter exceeds the output buffer threshold setting. The output buffer interrupt remains active until the output buffer counter is decremented to zero again.

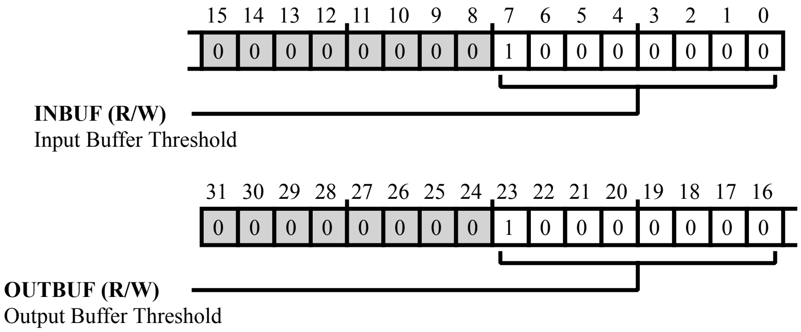


Figure 44-7: PKTE_BUF_THRESH Register Diagram

Table 44-33: PKTE_BUF_THRESH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	OUTBUF	Output Buffer Threshold. The `PKTE_BUF_THRESH.OUTBUF` bit field specifies how many bytes must be available in the packet engine output buffer before an output transfer starts. Valid values range from 0 to 252, in multiples of 4. In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application. In direct host mode, the output buffer threshold (obufthrsh) interrupt activates when the output buffer counter for the output buffer exceeds the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.

Table 44-33: PKTE_BUF_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	INBUF	Input Buffer Threshold. The `PKTE_BUF_THRESH.INBUF` bit field specifies how many bytes must be free in the packet engine input buffer before an input transfer starts. Valid values range from 0 to 252, in multiples of 4. In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application. In direct host mode, the input buffer threshold (ibufthrsh) interrupt activates when the input buffer counter for the input buffer is below or equal the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.

Packet Engine Command Descriptor Ring Base Address

The `PKTE_CDRBASE_ADDR` register holds the command descriptor ring base address in host memory. It is only applicable in autonomous ring mode. The `PKTE_CDRBASE_ADDR` register is ignored for all other modes when command descriptors are directly written into the descriptor registers.

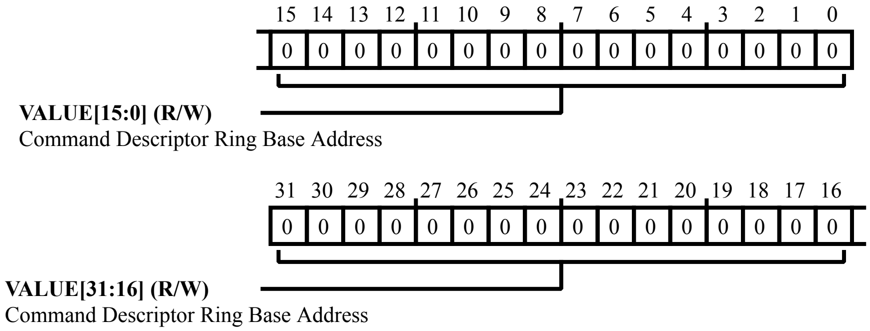

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Command Descriptor Ring Base Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Command Descriptor Ring Base Address

Figure 44-8: PKTE_CDRBASE_ADDR Register Diagram

Table 44-34: PKTE_CDRBASE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor Ring Base Address. The `PKTE_CDRBASE_ADDR.VALUE` bit field specifies the base location of the command descriptor ring in the host memory space.

Packet Engine Command Descriptor Count Register

The `PKTE_CDSC_CNT` register holds the counter for the number of descriptors in the Command Descriptor Ring (CDR). It is decremented by the packet engine each time a valid descriptor is read from the CDR and processed.

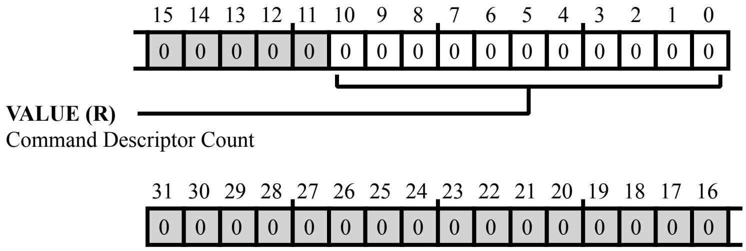

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Command Descriptor Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-9: PKTE_CDSC_CNT Register Diagram

Table 44-35: PKTE_CDSC_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Command Descriptor Count. The `PKTE_CDSC_CNT.VALUE` bit field provides the number of command descriptors in the command descriptor ring. The packet engine decrements the counter when a valid command descriptor is read from the CDR and processed.

Packet Engine Command Descriptor Count Increment Register

The `PKTE_CDSC_INCR` register is accessible by the host connected through the system slave bus. The host can increment the command descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

In autonomous ring mode, the host must prepare 1 to 255 valid command descriptors in the CDR and then write this register with a value between 1 and 255. The write triggers the packet engine to fetch the command descriptors from the CDR. In direct host mode or target command mode, the host must write one valid command descriptor to the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is available.

A CDR threshold interrupt is activated when the command descriptor counter is less than or equal to the threshold value set in the `PKTE_RING_THRESH` register. This interrupt can be used to wake up a process that stalled on a full CDR.

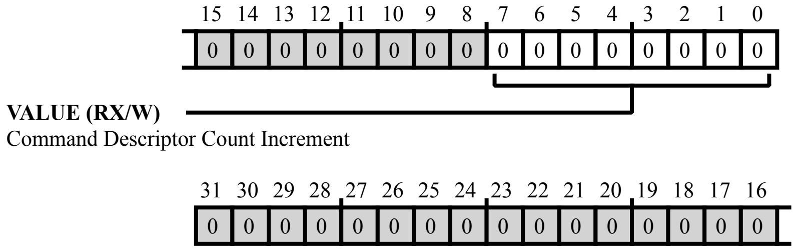


Figure 44-10: PKTE_CDSC_INCR Register Diagram

Table 44-36: PKTE_CDSC_INCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Command Descriptor Count Increment. The value written to the `PKTE_CDSC_INCR.VALUE` bit field is added to the command descriptor counter. The counter is protected against overflow (see the `PKTE_RING_STAT` register description). Note that bits[10:8] should be written with zeros.

Packet Engine Configuration Register

The `PKTE_CFG` register is used to select static settings that control the packet-processing path. This register is typically the last one to be written during the initialization sequence. These settings are typically set at initialization and not changed again.

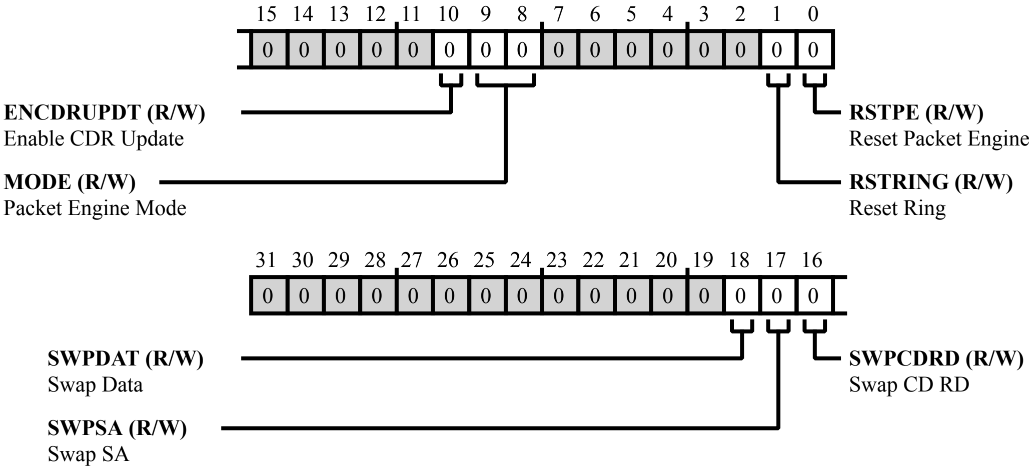

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENCDRUPDT (R/W)
Enable CDR Update
MODE (R/W)
Packet Engine Mode
RSTPE (R/W)
Reset Packet Engine
RSTRING (R/W)
Reset Ring
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SWPDAT (R/W)
Swap Data
SWPSA (R/W)
Swap SA
SWPCDRD (R/W)
Swap CD RD

Figure 44-11: PKTE_CFG Register Diagram

Table 44-37: PKTE_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	SWPDAT	Swap Data. The `PKTE_CFG.SWPDAT` bit enables endian swap for packet data as configured in the `PKTE_ENDIAN_CFG.MSTRBSWP` bits for the packet data DMA read and write.	
		0	No Endian Swap
		1	Apply Endian Swap
17 (R/W)	SWPSA	Swap SA. The `PKTE_CFG.SWPSA` bit enables endian swap for a SA record as configured in the `PKTE_ENDIAN_CFG.MSTRBSWP` bits for the SA record and state record DMA read and write. If the `PKTE_ENDIAN_CFG.MSTRBSWP` bits specify no endian swap, this bit is ignored.	
		0	No Endian Swap
		1	Apply Endian Swap

Table 44-37: PKTE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	SWPCDRD	Swap CD RD. The `PKTE_CFG.SWPCDRD` bit enables endian swap for descriptors as configured in the `PKTE_ENDIAN_CFG.MSTRBSWP` bits for the command descriptor DMA read and result descriptor DMA write. If the `PKTE_ENDIAN_CFG.MSTRBSWP` bits specify no endian swap, this bit is ignored.	
		0	No Endian Swap
		1	Apply Endian Swap
10 (R/W)	ENCDRUPDT	Enable CDR Update. The `PKTE_CFG.ENCDRUPDT` bit enables the packet engine to update, (clear the ownership bits) in the command descriptor in the CDR.	
		0	Do Not Clear Ownership Bits. The packet engine does not clear the ownership bits in the command descriptor when it completes an operation. The host application must clear the ownership bits in "old descriptors" before the packet engine is allowed to wrap around the CDR to re-encounter these "old descriptors". This setting has the advantage of eliminating a separate DMA write to the CDR.
		1	Clear Ownership Bits. The packet engine clears (set to zero) the ownership bits in the current command descriptor in the CDR. This prevents the packet engine from re-processing an "old descriptor" when it wraps around the CDR.
9:8 (R/W)	MODE	Packet Engine Mode. The `PKTE_CFG.MODE` bit field selects how the packet engine receives commands.	
		0	Direct Host Mode.
		1	Target Command Mode with Result Descriptor Ring Disabled.
		2	Target Command Mode with Result Descriptor Ring Enabled.
		3	Autonomous Ring Mode

Table 44-37: PKTE_CFG Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">1
(R/W)</td><td rowspan="3">RSTRING</td><td colspan="2">Reset Ring.
The PKTE_CFG.RSTRING bit resets the internal counters for the CDR and RDR, PKTE_CDSC_CNT and PKTE_RDSC_CNT registers) to zero. Resets the PKTE_RING_PTR register to the base address. After the reset the rings are empty.
This bit must be written with a '1' to reset the descriptor ring manager and then re-written with a '0' to release the reset. Note that this bit can remain in the reset state if the CDR ring is disabled (PKTE_CFG.MODE is not 0b11).
Note that this reset must be coordinated with the 'owner' of the descriptor ring to ensure that the pointers are in sync after the reset.</td></tr>
<tr><td>0</td><td>Release the Descriptor Ring Manager Reset</td></tr>
<tr><td>1</td><td>Reset the Descriptor Ring Manager</td></tr>
<tr><td rowspan="3">0
(R/W)</td><td rowspan="3">RSTPE</td><td colspan="2">Reset Packet Engine.
The PKTE_CFG.RSTPE bit resets the packet engine and the state machine logic that drives header processing, DMA, and context management. The PKTE_CFG.RSTPE bit resets the PKTE_CTL_STAT and PKTE_LEN internal registers.
This bit must be written with a 1 to reset the packet engine and then re-written with a 0 to release the reset. Note that this bit should not be used by a typical application. It is provided to use during development testing or to recover from critical errors.
Note that the PKTE_CTL_STAT.PADVAL and PKTE_CTL_STAT.PADCTLSTAT bit fields are only reset when in autonomous ring mode, but the PKTE_CTL_STAT.PRNGMD bit is not reset. Halt mode is not affected by this reset as well. When exiting out of halt mode, a HW reset is required or a write to the PKTE_CONT register.</td></tr>
<tr><td>0</td><td>Release the Packet Engine Reset</td></tr>
<tr><td>1</td><td>Reset the Packet Engine</td></tr>
</table>

PE Clock Control Register

The PKTE_CLK_CTL register controls the clock enable signals. This register can be used to enable the clock for read and write access to SA registers or to enable the required clock signals for certain crypto functions. The setting of this register overrides the packet engine dynamic clock enable.

In autonomous ring mode and target command modes, this register can be all zeros; the packet engine dynamically requests the external clock manager to activate the module clocks. This register can be used in combination with the debugging interface for internal register access.

In direct host mode, the clock enable bits for the packet engine (`PKTE_CLK_CTL.ENPECLK`) and for ARC4 (`PKTE_CLK_CTL.ENARC4CLK`) must be enabled to write and read the SA record and state record registers. All module clocks that are required for the current operation must be enabled during processing.

Note that all the clocks are enabled by default to reset all the registers within the packet engine. After a system reset the host can program this register to disable clocks for power reduction.

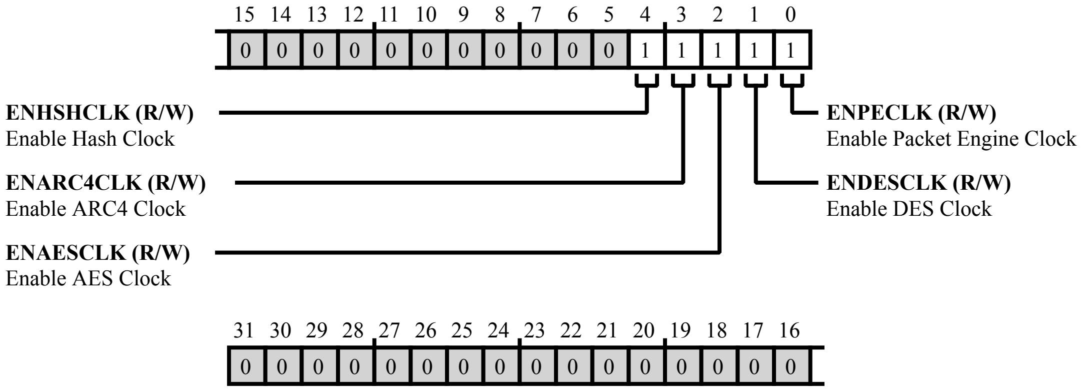

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
ENHSHCLK (R/W)
Enable Hash Clock
ENARC4CLK (R/W)
Enable ARC4 Clock
ENAESCLK (R/W)
Enable AES Clock
ENPECLK (R/W)
Enable Packet Engine Clock
ENDESCLK (R/W)
Enable DES Clock
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-12: PKTE_CLK_CTL Register Diagram

Table 44-38: PKTE_CLK_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	ENHSHCLK	Enable Hash Clock. The `PKTE_CLK_CTL.ENHSHCLK` bit enables the clock to the hash functions.	
		0	Do not enable the hash clock
		1	Enable the hash clock
3 (R/W)	ENARC4CLK	Enable ARC4 Clock. The `PKTE_CLK_CTL.ENARC4CLK` bit enables the clock to the ARC4 function.	
		0	Do not enable the ARC4 clock
		1	Enable the ARC4 clock

Table 44-38: PKTE_CLK_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	ENAESCLK	Enable AES Clock. The `PKTE_CLK_CTL.ENAESCLK` bit enables the clock to the AES encrypt/decrypt function.	
		0	Do not enable the AES clock
		1	Enable the AES clock
1 (R/W)	ENDESCLK	Enable DES Clock. The `PKTE_CLK_CTL.ENDESCLK` bit enables the clock to the DES function.	
		0	Do not enable the DES clock
		1	Enable the DES clock
0 (R/W)	ENPECLK	Enable Packet Engine Clock. The `PKTE_CLK_CTL.ENPECLK` bit enables the clock in the PKTE data path.	
		0	Do not enable the PKTE data path clock
		1	Enable the PKTE data path clock

PKTE Continue Register

A write to the `PKTE_CONT` register (with any value) releases the packet engine from a halt state when in halt mode.

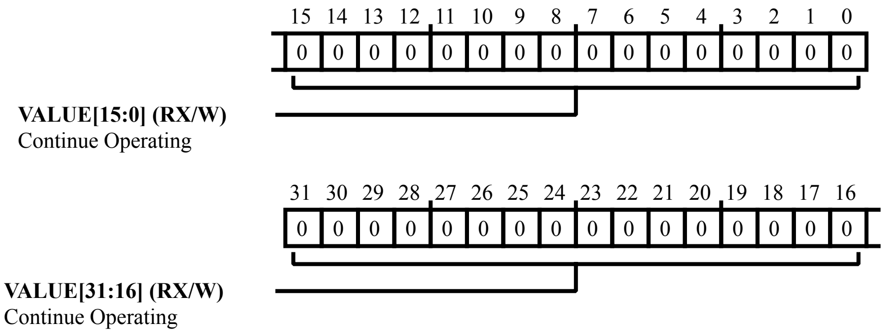

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (RX/W)
Continue Operating
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (RX/W)
Continue Operating

Figure 44-13: PKTE_CONT Register Diagram

Table 44-39: PKTE_CONT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Continue Operating. The `PKTE_CONT.VALUE` bit field releases the packet engine from a halt state when written with any value in halt mode.

Packet Engine Control Register

The PKTE_CTL_STAT register has a dual function. Together with the data in the SA, this register provides the basic command information for the packet engine to process a packet. When the packet engine successfully or unsuccessfully completes an operation, the packet engine control/status register provides the result status for the host.

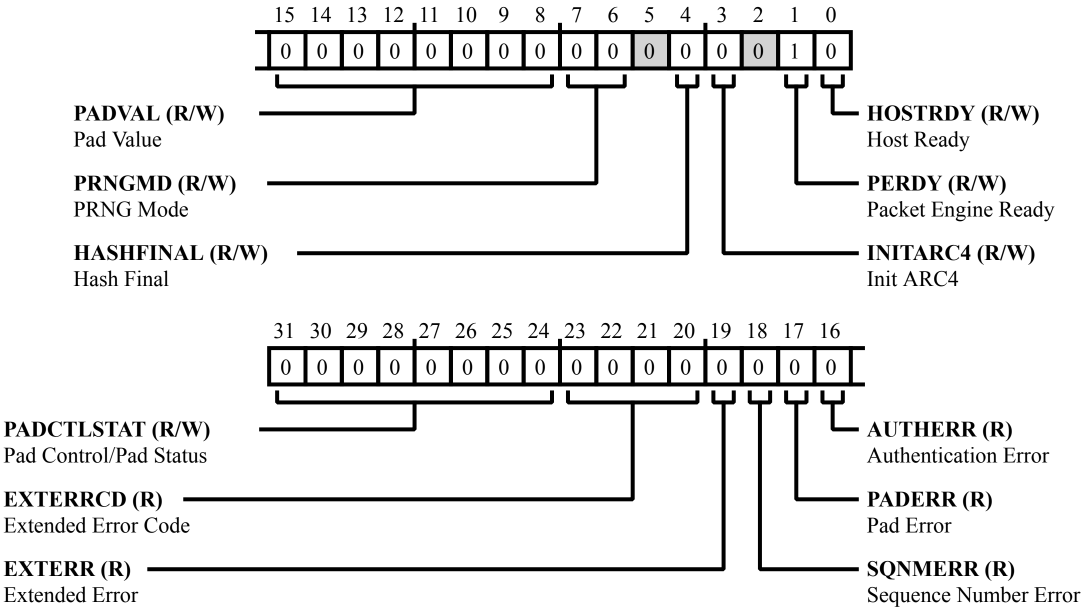

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
PADVAL (R/W)
Pad Value
PRNGMD (R/W)
PRNG Mode
HASHFINAL (R/W)
Hash Final
HOSTRDY (R/W)
Host Ready
PERDY (R/W)
Packet Engine Ready
INITARC4 (R/W)
Init ARC4
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PADCTLSTAT (R/W)
Pad Control/Pad Status
EXTERRCD (R)
Extended Error Code
EXTERR (R)
Extended Error
AUTHERR (R)
Authentication Error
PADERR (R)
Pad Error
SQNMERR (R)
Sequence Number Error

Figure 44-14: PKTE_CTL_STAT Register Diagram

Table 44-40: PKTE_CTL_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	PADCTLSTAT	Pad Control/Pad Status. The PKTE_CTL_STAT.PADCTLSTAT bit field is used to control the pad boundary for pad insertion (outbound) and after processing returns the number of inserted (outbound) or detected (inbound) pad bytes. For the command descriptor, the enumerations below provide the codes for the pad boundary for the outbound operations. This can be used for traffic flow security to conceal the number of payload bytes in an encrypted packet. For the result descriptor inbound operations that use pad types SSL, TLS, IPsec or PKCS#7, it returns the number of detected pad bytes. For all other inbound operations, it returns zero since the other pad modes do not allow implicit determination of pad count. If a pad verify failure occurs, it returns zero. For an outbound operation, it returns the number of inserted pad bytes for all pad types. The pad value includes added bytes such as the pad length and the next header field in an IPsec ESP pad type.	
		0	Align packet end to modulo 8-byte boundary
		1	Align packet end to modulo 1-byte boundary

Table 44-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Align packet end to modulo 4-byte boundary
		4	Align packet end to modulo 8-byte boundary
		8	Align packet end to modulo 16-byte boundary
		16	Align packet end to modulo 32-byte boundary
		32	Align packet end to modulo 64-byte boundary
		64	Align packet end to modulo 128-byte boundary
		128	Align packet end to modulo 256-byte boundary
23:20 (R/NW)	EXTERRCD	Extended Error Code. The `PKTE_CTL_STAT.EXTERRCD` bit field represents an encoded error condition.	
19 (R/NW)	EXTERR	Extended Error. The `PKTE_CTL_STAT.EXTERR` bit field provides an extended error code.	
		0	No Extended Error
		1	Extended Error
18 (R/NW)	SQNMERR	Sequence Number Error. The `PKTE_CTL_STAT.SQNMERR` bit indicates that for an inbound operation, there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition.	
		0	No Sequence Number Error
		1	Sequence Number Error
17 (R/NW)	PADERR	Pad Error. The `PKTE_CTL_STAT.PADERR` bit indicates that for an inbound operation the decrypted pad does not match the expected values.	
		0	No Pad Error
		1	Pad Error
16 (R/NW)	AUTHERR	Authentication Error. The `PKTE_CTL_STAT.AUTHERR` bit indicates that for an inbound operation the authentication value in the packet does not match the computed value.	
		0	No Authentication Error
		1	Authentication Error

Table 44-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	PADVAL	Pad Value. The `PKTE_CTL_STAT.PADVAL` bit field is used to pass the pad value between the host and the packet engine. Command Descriptor: (write-only) For outbound operations that use pad type IPsec, the host must populate this field with the value that is to be inserted into the next header field. For the IPsec ESP operation, this next header is part of the ESP trailer of the innermost operation's header and the value must be 50 decimal. For outbound encrypt operations that use the pad type constant or constant SSL, the host must specify the fixed constant value in this field. For all other outbound and inbound operations, this field is not used. Result Descriptor: (read only) For inbound operations that use pad type IPsec, the packet engine returns the next header field that it detects. For IPsec ESP inbound operations, this is the next header field in the innermost operation's header, which will typically be the value for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other outbound operations, the packet engine will not update this field. For all other inbound operations, the returned pad value is zero.	
7:6 (R/W)	PRNGMD	PRNG Mode. The `PKTE_CTL_STAT.PRNGMD` bits select the pseudo-random number generator mode.	
		0	Operation does not use the PRNG function.
		1	PRNG Init. PRNG is initialized with a SEED, KEY and an LFSR value as defined in the SA.
		2	PRNG Generate. Pseudo-random data is generated with the LFSR as input value. Before this mode can be used, the PRNG must be initialized with a valid SEED, KEY and LFSR using PRNG Init (`PKTE_CTL_STAT.PRNGMD`=b'01).
		3	PRNG Test. It can be used to test the PRNG function with custom input data. Before this mode can be used, the PRNG must be initialized once with a valid SEED using PRNG Init (`PKTE_CTL_STAT.PRNGMD`=b'01).

Table 44-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	HASHFINAL	Hash Final. When the `PKTE_CTL_STAT.HASHFINAL` bit is zero, the data to be hashed must be a multiple of the hash block size, 64 bytes for SHA-1, MD5, SHA-224, SHA-256. This bit is only applicable for Basic Hash, Basic Encrypt-Hash and Basic Hash-Decrypt operations that use the SHA-1, MD5, SHA-224, SHA-256 hash algorithm. The `PKTE_CTL_STAT.HASHFINAL` bit is overruled for HMAC operations that always completes the hash and always returns the last written value on a read by the host.	
		0	Perform Intermediate Hash Operation. The packet engine performs an intermediate hash operation by generating an intermediate hash digest on the data presented on the input. No hash pad is applied.
		1	Perform Final Hash Operation. The packet engine appends the required final hash pad and generates the final hash digest on the data presented on the input. This completes the hash operation.
3 (R/W)	INITARC4	Init ARC4. The `PKTE_CTL_STAT.INITARC4` bit initializes the ARC4 crypto algorithm with a new key. This bit always returns the last written value on a read by the Host. This bit is only applicable for operations that use the ARC4 algorithm and must be zero for all other operations.	
		0	Load ARC4 State and ARC4 i/j pointer from the SA The ARC4 State and ARC4 i/j pointer are loaded from the SA to continue the encrypt/decrypt processing from the previous algorithm state.
		1	Read ARC4 key from the SA-record and initialize ARC4 S-boxes using this key The ARC4 key is read from the SA-record, the ARC4 S-boxes are initialized using this key, prior to the encryption/decryption of data. This bit overrules Stateful mode as defined in bits [9:8] of PKTE_SA_CMD1.
1 (R/W)	PERDY	Packet Engine Ready. The `PKTE_CTL_STAT.PERDY` bit indicates that the packet engine has completed processing the command descriptor and returns the result descriptor with ownership set to the host. This bit can be reset to 0 by the host and the packet engine, but only the packet engine can set this bit. When the packet engine is idle (not processing), this bit always returns '1' on a read by the host.	

Table 44-40: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	HOSTRDY	Host Ready. The `PKTE_CTL_STAT.HOSTRDY` bit indicates that the host has populated the command descriptor. This bit can be reset to 0 by the host and the packet engine, but only the host can set this bit. When the packet engine is idle (not processing), this bit always returns '0' on a read by the host.

Starting Entry of 256-byte Data Input/Output Buffer

When in direct host mode, the source packet data is written here to be transferred to the packet engine. The host can monitor the available space in the input buffer through the `PKTE_STAT` register. This is also the location in the packet engine from where output data is read when in direct host mode. The host can monitor the available bytes in the output buffer through the `PKTE_STAT` register.

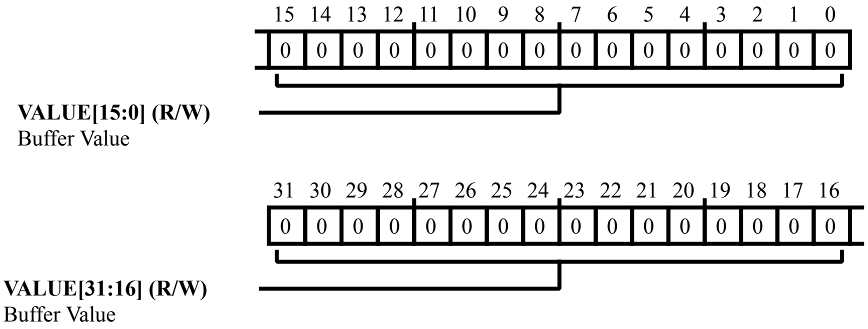

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Buffer Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Buffer Value

Figure 44-15: PKTE_DATAIO_BUF Register Diagram

Table 44-41: PKTE_DATAIO_BUF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Buffer Value.

Packet Engine Destination Address

The PKTE_DEST_ADDR register holds the starting (byte) address to write the result packet from the requested operation.

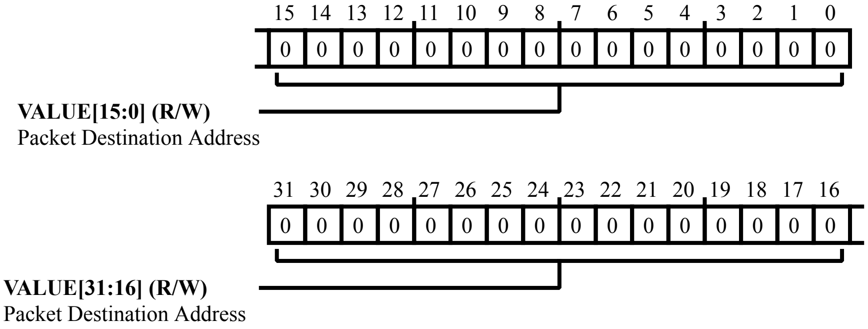

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Packet Destination Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Packet Destination Address

Figure 44-16: PKTE_DEST_ADDR Register Diagram

Table 44-42: PKTE_DEST_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Destination Address.

Packet Engine DMA Configuration Register

The `PKTE_DMA_CFG` register configures the maximum burst transfer size, enables incremental transfers, and insertions IDLE cycles between two bus transfers.

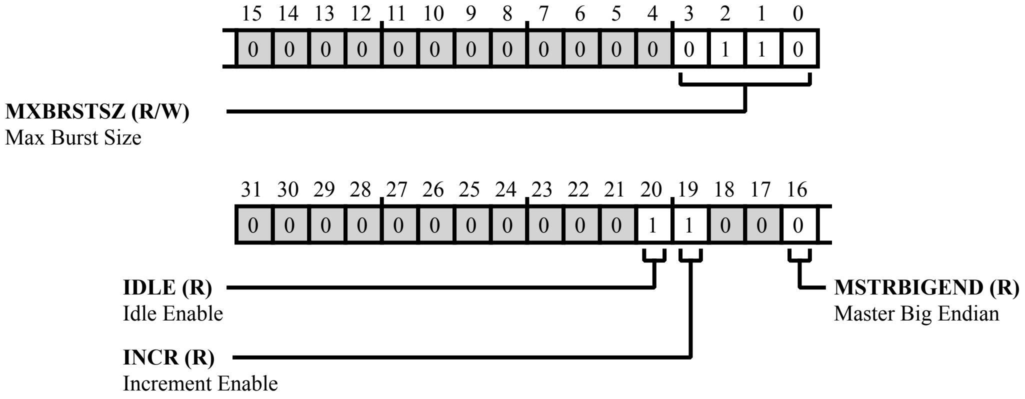

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
MXBRSTSZ (R/W)
Max Burst Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
IDLE (R)
Idle Enable
INCR (R)
Increment Enable
MSTRBIGEND (R)
Master Big Endian

Figure 44-17: PKTE_DMA_CFG Register Diagram

Table 44-43: PKTE_DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	IDLE	Idle Enable. The `PKTE_DMA_CFG.IDLE` bit allows the peripheral bus master to insert one additional IDLE transfer between two successive peripheral bus master burst operations. This provides the arbiter one additional cycle to hand over the grant to another peripheral bus master.	
		0	The peripheral bus master inserts no IDLE cycle between two successive burst operations
		1	The peripheral bus master inserts one additional IDLE transfer between two successive burst operations

Table 44-43: PKTE_DMA_CFG Register Fields (Continued)

<table>
<tr><th>Bit No.
(Access)</th><th>Bit Name</th><th colspan="2">Description/Enumeration</th></tr>
<tr><td rowspan="3">19
(R/NW)</td><td rowspan="3">INCR</td><td colspan="2">Increment Enable.
The <code>PKTE_DMA_CFG.INCR</code> bit lets the peripheral bus master generate INC4, INC8 and INC16 type of burst transfers.
By default, the peripheral bus master generates the largest possible incremental burst of unspecified length (INCR) with a maximum length (in bytes) as configured by the <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field. In case there are less than 4 bytes of data available or the 1kB boundary will be crossed using a burst operation, then a single transfer of size byte is generated.
When the <code>PKTE_DMA_CFG.INCR</code> bit is set, the peripheral bus master generates one or more incremental burst of specified length (INC4, INC8, INC16). In case there is less data available then the smallest possible burst (INC4) or the 1kB boundary will be crossed using a burst operation, then an unspecified length burst or a single transfer of size byte is generated.</td></tr>
<tr><td>0</td><td>The bus master will generate only INCR burst types</td></tr>
<tr><td>1</td><td>The bus master will generate INC4, INC8 and INC16 burst types</td></tr>
<tr><td rowspan="3">16
(R/NW)</td><td rowspan="3">MSTRBIGEND</td><td colspan="2">Master Big Endian.
The <code>PKTE_DMA_CFG.MSTRBIGEND</code> bit determines whether the engine is used in a little or big endian system.</td></tr>
<tr><td>0</td><td>Little endian</td></tr>
<tr><td>1</td><td>Big endian</td></tr>
<tr><td>3:0
(R/W)</td><td>MXBRSTSZ</td><td colspan="2">Max Burst Size.
The <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field configures the maximum size of an unspecified length burst (INC) at the bus in bytes. When there is less data available than the <code>PKTE_DMA_CFG.MXBRSTSZ</code> bit field setting or the 1kB boundary will be crossed using a burst operation, then the length of the burst can be less than <code>PKTE_DMA_CFG.MXBRSTSZ</code>. Any requested transfers larger than this size are broken up in to multiple burst transfers of this size or less.</td></tr>
</table>

Packet Engine Endian Configuration Register

The packet engine incorporates a powerful interface specific endian handler. This endian handler allows byte lane swapping in each direction for data passing through the host interface.

The `PKTE_ENDIAN_CFG` register configures the byte order function for the peripheral bus master and peripheral bus slave interface. The bits for the peripheral bus master are combined in four sets of two bits; each group configures a byte swap function for a particular DMA transfer. The same applies for the peripheral bus slave interface.

The `PKTE_ENDIAN_CFG` register also defines the endian swapping that occurs for host-initiated target transfers and for packet engine master DMA read and write transfers. Individual endian swap enable bits in the configuration (`PKTE_CFG`) register can enable the endian swap for various transaction types: command descriptors and result descriptors, SA records and state records, packet data.

In direct host mode, only target operations are supported. Only the target endian configuration of this register is applicable.

Note: This register is typically programmed once during the initialization phase, although software is allowed to dynamically change the setting in this register just before initiating a data transfer. The developer will have to analyze the benefit of the cycles needed to write the endian register dynamically versus handling endian swapping for some data structures in the host system (most modern processors support a byte swap in zero cycles). Certainly the endian swap should be set correctly for the packet data, since this represents the majority of the data transferred.

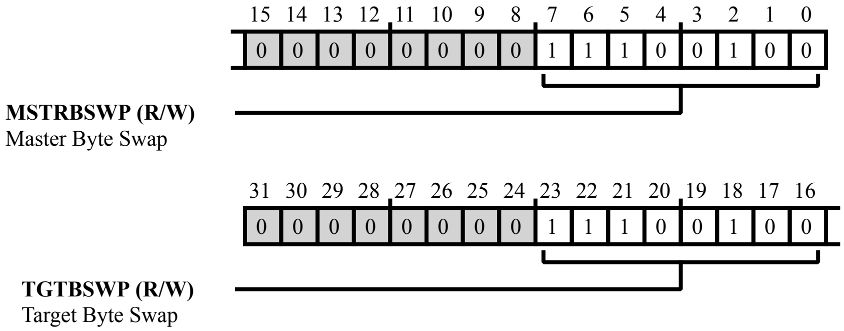


Figure 44-18: PKTE_ENDIAN_CFG Register Diagram

Table 44-44: PKTE_ENDIAN_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TGTBSWP	Target Byte Swap. The `PKTE_ENDIAN_CFG.TGTBSWP` bit field configures the byte swap for peripheral bus target transfers. Note that only target word transfers are supported. Each double-bit field in this register specifies the source of the indicated byte lane. The field values are interpreted as follows: 00 = byte 0, 01 = byte 1, 10 = byte 2, 11 = byte 3. Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).

Table 44-44: PKTE_ENDIAN_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MSTRBSWP	Master Byte Swap. The `PKTE_ENDIAN_CFG.MSTRBSWP` bit field configures the byte swap for peripheral bus master multi-byte transfers, including command descriptors, result descriptors, SA records, state records and packet data. Separate controls in the `PKTE_CFG` register can enable this swap individually for each of the 4 types of data. Each double-bit field in this register specifies the source of the indicated peripheral bus byte lane. The field values are interpreted as follows: 00=byte 0, 01=byte 1, 10=byte 2, 11=byte 3. Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).

Packet Engine Halt Control Register

Th `PKTE_HLT_CTL` register controls the packet engine halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. During the halt mode, the host can read all internal registers for examination without side-effects. When halted, the host should not write to any registers. To continue packet engine operation, the host must write to the `PKTE_CONT` (PKTE continue) register.

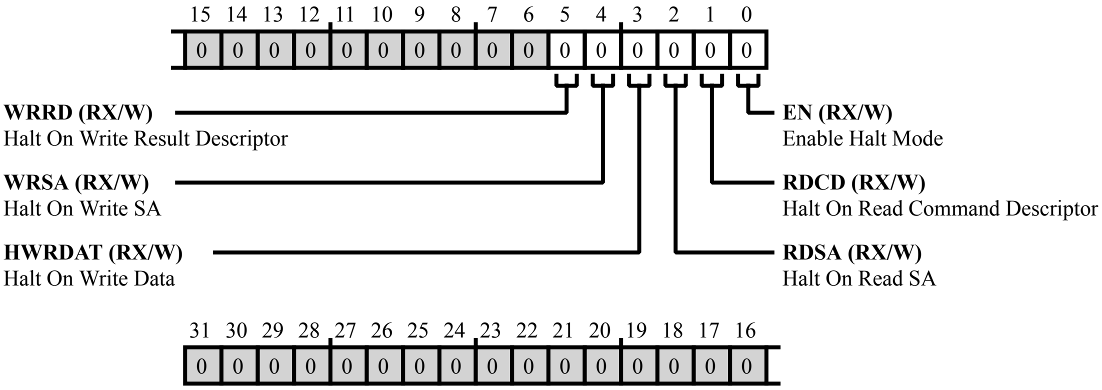

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WRRD (RX/W)
Halt On Write Result Descriptor
WRSA (RX/W)
Halt On Write SA
HWRDAT (RX/W)
Halt On Write Data
EN (RX/W)
Enable Halt Mode
RDCD (RX/W)
Halt On Read Command Descriptor
RDSA (RX/W)
Halt On Read SA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-19: PKTE_HLT_CTL Register Diagram

Table 44-45: PKTE_HLT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	WRRD	Halt On Write Result Descriptor. The `PKTE_HLT_CTL.WRRD` bit halts the packet engine in the HALT_WRITE_STATUS state after it completes a result descriptor write operation to the result descriptor ring. The host can use this bit to examine the result descriptor that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
4 (RX/W)	WRSA	Halt On Write SA. The `PKTE_HLT_CTL.WRSA` bit halts the packet engine in the HALT_WRITE_SA state after it completes an SA write operation to the host memory. The host can use this bit to examine the security context that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation

Table 44-45: PKTE_HLT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RX/W)	HWRDAT	Halt On Write Data. The `PKTE_HLT_CTL.HWRDAT` bit halts the packet engine in the HALT_DATA state after it completes writing the result packet data to the host memory. The host can use this bit to examine the result packet that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
2 (RX/W)	RDSA	Halt On Read SA. The `PKTE_HLT_CTL.RDSA` bit halts the packet engine in the HALT_READ_SA state after it completes an SA read operation from the host memory. The host can use this bit to examine the security context that is currently in the SA registers.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
1 (RX/W)	RDCD	Halt On Read Command Descriptor. The `PKTE_HLT_CTL.RDCD` bit halts the packet engine in the HALT_READ_DESCR state after it completes a command descriptor read operation from the command descriptor ring. It will halt whether the descriptor is valid or invalid. The host can use this bit to examine the command descriptor that is currently in the internal command descriptor registers.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
0 (RX/W)	EN	Enable Halt Mode. The `PKTE_HLT_CTL.EN` bit enables halt mode where the packet engine can halt processing at any processing state as indicated by bits [5:1]. When halted, the packet engine continues operation on a write to the `PKTE_CONT` register.	
		0	Do not enable halt mode
		1	Enable halt mode

Packet Engine Halt Status Register

The PKTE_HLT_STAT register reflects the status of the packet engine in halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. When the packet engine is halted, the host can read all internal registers for examination without side effects. The host should not write to any registers.

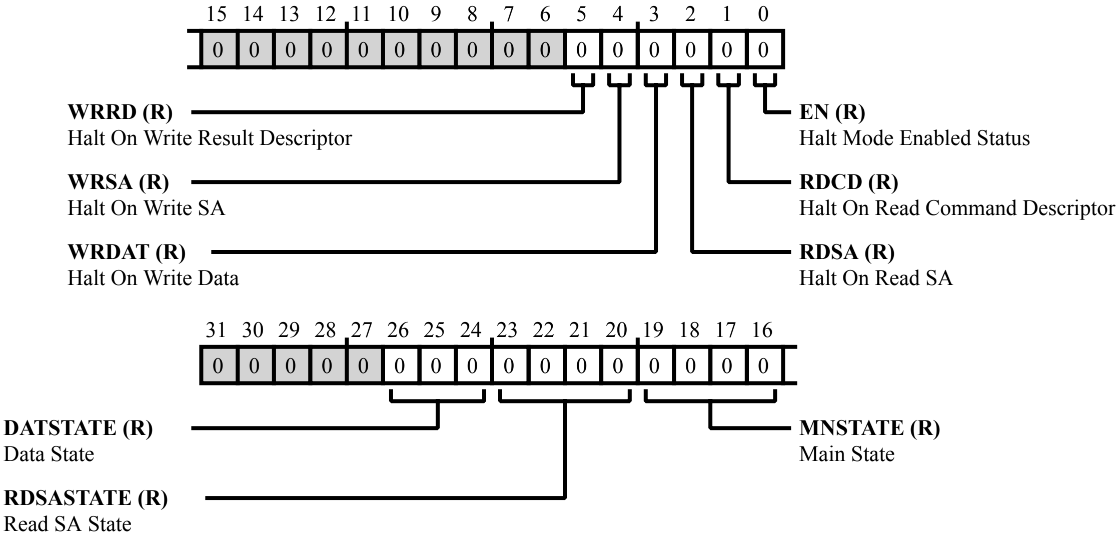

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WRRD (R)
Halt On Write Result Descriptor
WRSA (R)
Halt On Write SA
WRDAT (R)
Halt On Write Data
EN (R)
Halt Mode Enabled Status
RDCD (R)
Halt On Read Command Descriptor
RDSA (R)
Halt On Read SA
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATSTATE (R)
Data State
RDSASTATE (R)
Read SA State
MNSTATE (R)
Main State

Figure 44-20: PKTE_HLT_STAT Register Diagram

Table 44-46: PKTE_HLT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26:24 (R/NW)	DATSTATE	Data State. The PKTE_HLT_STAT.DATSTATE bit field indicates the state of the packet engine read data FSM.	
		0	DATA_IDLE, no operation
		1	DATA_READ
		2	DATA_WRITE
		3	DATA_WAIT
		5	DATA_PAD_READ
		6	DATA_BYP_READ
		7	RESERVED
23:20 (R/NW)	RDSASTATE	Read SA State. The PKTE_HLT_STAT.RDSASTATE bit field indicates the state of the packet engine read SA FSM.	

Table 44-46: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	SA_IDLE, no operation
		1	SA_READ_CMD
		2	SA_READ_STATE_IV
		3-5	RESERVED
		6	SA_READ_ARC4_STATE
		7	SA_READ_WAIT
		8	RESERVED
		9	SA_WRITE_PROT_HDR
		10	RESERVED
		11	SA_WRITE_IV
		12	SA_WRITE_DIGEST
		13	SA_WRITE_ARC4_IJ_PNTR
		14	SA_WRITE_ARC4_STATE
		15	SA_WRITE_WAIT
19:16 (R/NW)	MNSTATE	Main State. The `PKTE_HLT_STAT.MNSTATE` bit field indicates the state of the packet engine main FSM.	
		0	MAIN_IDLE, no operation
		1	MAIN_READ_CD, reading command descriptor
		2	MAIN_READ_SA, reading SA
		3	MAIN_DATA, processing data
		4	MAIN_WRITE_SA, writing SA
		5	MAIN_WRITE_STATUS, writing status
		6	MAIN_WRITE_CD, updating command descriptor
		7	MAIN_WRITE_RD, updating result descriptor
		8	MAIN_INIT_WAIT, wait single clock
		9	MAIN_HALT_READ_CD, halt after read command descriptor
		10	MAIN_HALT_READ_SA, halt after read SA
		11	MAIN_HALT_DATA, halt after processing data
		12	MAIN_HALT_WRITE_SA, halt after write SA

Table 44-46: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		13	MAIN_WAIT_FOR_CLOCK, wait for clocks to be active
		15	MAIN_HALT_WRITE_RD, halt after write result descriptor
5 (R/NW)	WRRD	Halt On Write Result Descriptor. The `PKTE_HLT_STAT.WRRD` bit reflects the value in the `PKTE_HLT_CTL.WRRD` bit.	
4 (R/NW)	WRSA	Halt On Write SA. The `PKTE_HLT_STAT.WRSA` bit reflects the value in the `PKTE_HLT_CTL.WRSA` bit.	
3 (R/NW)	WRDAT	Halt On Write Data. The `PKTE_HLT_STAT.WRDAT` bit reflects the value in the `PKTE_HLT_CTL.HWRDAT` bit.	
2 (R/NW)	RDSA	Halt On Read SA. The `PKTE_HLT_STAT.RDSA` bit reflects the value in the `PKTE_HLT_CTL.RDSA` bit.	
1 (R/NW)	RDCD	Halt On Read Command Descriptor. The `PKTE_HLT_STAT.RDCD` bit reflects the value in the `PKTE_HLT_CTL.RDCD` bit.	
0 (R/NW)	EN	Halt Mode Enabled Status.	
		0	Halt mode not enabled
		1	Halt mode enabled

Interrupt Mask Disable Register

The host can use the PKTE_IMSK_DIS register to clear individual bits in the PKTE_INT_EN register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 clears the interrupt enable bit, a 0 does not affect the interrupt enable bit in the PKTE_INT_EN register. Clearing the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

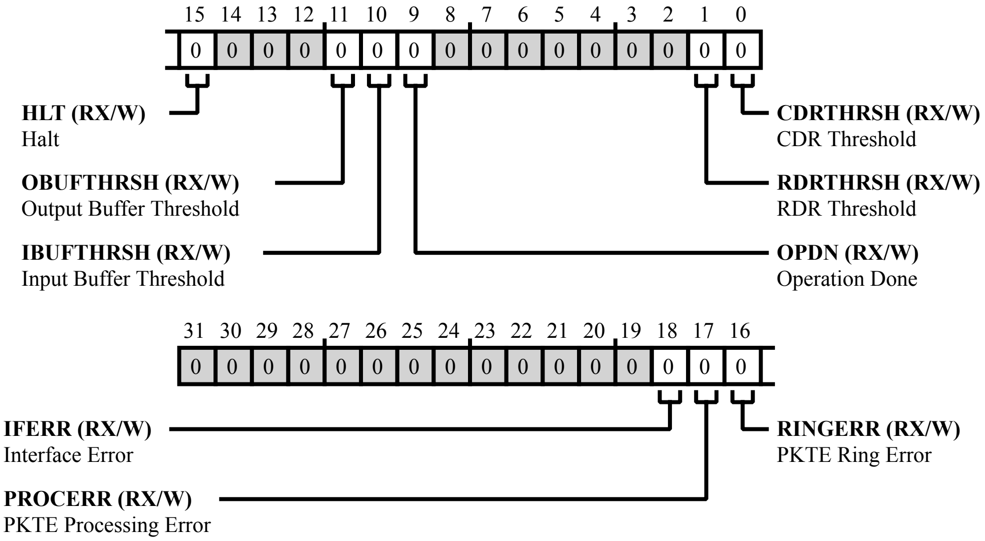

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (RX/W)
Halt
OBUFTHRSH (RX/W)
Output Buffer Threshold
IBUFTHRSH (RX/W)
Input Buffer Threshold
CDRTHRSH (RX/W)
CDR Threshold
RDRTHRSH (RX/W)
RDR Threshold
OPDN (RX/W)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (RX/W)
Interface Error
PROCERR (RX/W)
PKTE Processing Error
RINGERR (RX/W)
PKTE Ring Error

Figure 44-21: PKTE_IMSK_DIS Register Diagram

Table 44-47: PKTE_IMSK_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Write the `PKTE_IMSK_DIS.IFERR` bit to clear when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Write the `PKTE_IMSK_DIS.PROCERR` bit to clear an extended error that occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. Write the `PKTE_IMSK_DIS.RINGERR` bit to clear a CDR overflow or an RDR underflow.
15 (RX/W)	HLT	Halt. Write the `PKTE_IMSK_DIS.HLT` bit to clear when the packet engine is in the halt state.

Table 44-47: PKTE_IMSK_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. Write the `PKTE_IMSK_DIS.OBUFTHRSH` bit to clear the output buffer counter exceeds the output buffer threshold value defined in `PKTE_BUF_THRESH.OUTBUF` bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Write the `PKTE_IMSK_DIS.IBUFTHRSH` bit to clear when the input buffer counter is less than or equal to the input buffer threshold value defined in `PKTE_BUF_THRESH.INBUF` bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Write the `PKTE_IMSK_DIS.RDRTHRSH` bit to clear when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit, or the RD counter for the RDR in the `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Write the `PKTE_IMSK_DIS.CDRTHRSH` bit to clear when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Interrupt Mask Enable Register

The host can use the PKTE_IMSK_EN register to set individual bits in the PKTE_INT_EN register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 sets the interrupt enable bit, a 0 does not affect the interrupt enable bit in the PKTE_INT_EN register. Setting the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

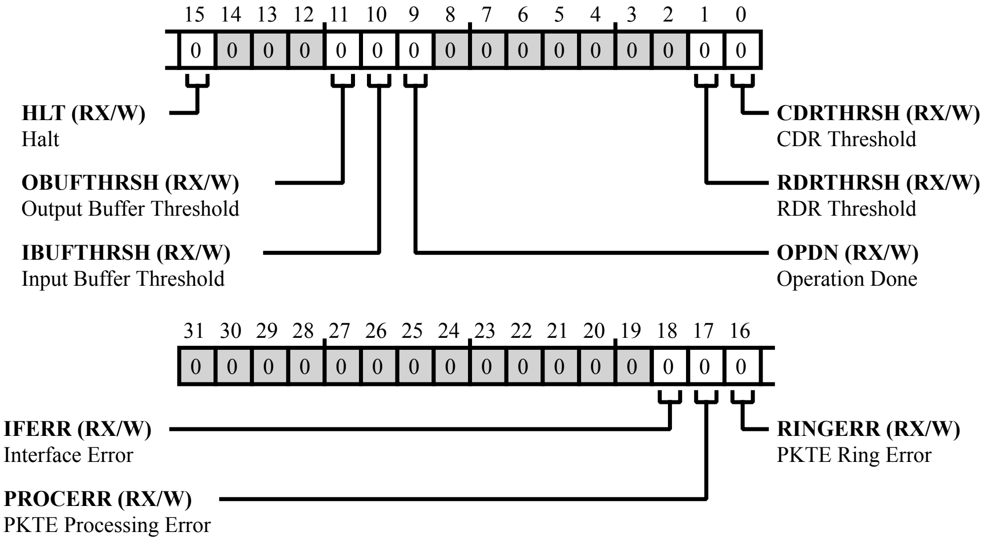

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (RX/W)
Halt
OBUFTHRSH (RX/W)
Output Buffer Threshold
IBUFTHRSH (RX/W)
Input Buffer Threshold
CDRTHRSH (RX/W)
CDR Threshold
RDRTHRSH (RX/W)
RDR Threshold
OPDN (RX/W)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (RX/W)
Interface Error
PROCERR (RX/W)
PKTE Processing Error
RINGERR (RX/W)
PKTE Ring Error

Figure 44-22: PKTE_IMSK_EN Register Diagram

Table 44-48: PKTE_IMSK_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Set the `PKTE_IMSK_EN.IFERR` bit to indicate a host request for a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Set the `PKTE_IMSK_EN.PROCERR` bit to indicate an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error.
15 (RX/W)	HLT	Halt. Set the `PKTE_IMSK_EN.HLT` bit to indicate when the packet engine is in the halt state.

Table 44-48: PKTE_IMSK_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. Set the `PKTE_IMSK_EN.OBUFTHRSH` bit to indicate that the output buffer counter exceeds the output buffer threshold value defined in the `PKTE_BUF_THRESH.OUTBUF` bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Set the `PKTE_IMSK_EN.IBUFTHRSH` bit to indicate the input buffer counter is less than or equal to the input buffer threshold value defined in `PKTE_BUF_THRESH.INBUF` bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Set the `PKTE_IMSK_EN.RDRTHRSH` bit to indicate when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit, or the RD counter for the RDR in `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Set the `PKTE_IMSK_EN.CDRTHRSH` bit to indicate when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Interrupt Masked Status Register

The PKTE_IMSK_STAT register provides interrupt status visibility to the host, after the interrupt mask is applied. This lets the host view the selected sources of interrupts that are directed to the interrupt output signal, that is connected to the system interrupt controller. As with the unmasked status register, all interrupt bits are latched and must be cleared using the PKTE_INT_CLR register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

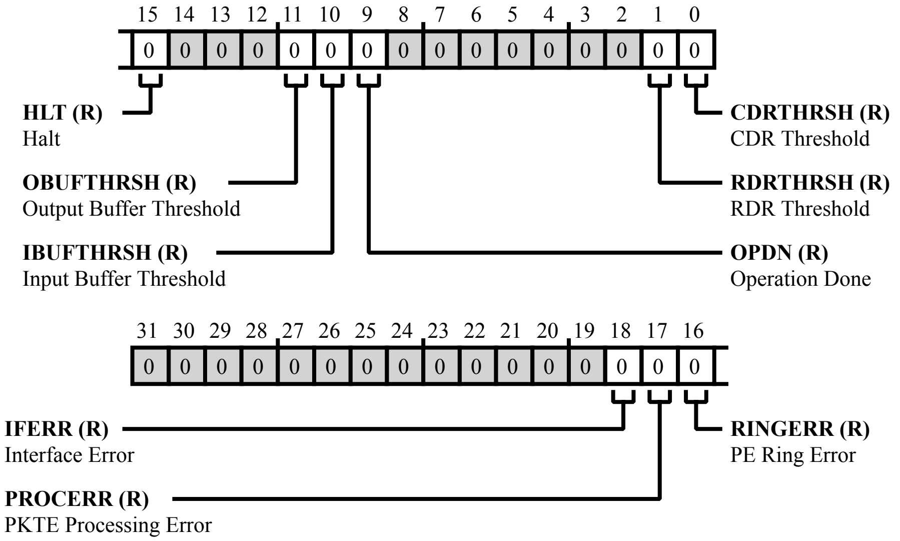

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (R)
Halt
OBUFTHRSH (R)
Output Buffer Threshold
IBUFTHRSH (R)
Input Buffer Threshold
CDRTHRSH (R)
CDR Threshold
RDRTHRSH (R)
RDR Threshold
OPDN (R)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (R)
Interface Error
PROCERR (R)
PKTE Processing Error
RINGERR (R)
PE Ring Error

Figure 44-23: PKTE_IMSK_STAT Register Diagram

Table 44-49: PKTE_IMSK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The PKTE_IMSK_STAT.IFERR bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The PKTE_IMSK_STAT.PROCERR bit is when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PE Ring Error. The PKTE_IMSK_STAT.RINGERR bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The PKTE_IMSK_STAT.HLT bit is set when the packet engine is in the HALT state.

Table 44-49: PKTE_IMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	OBUFTHRSH	Output Buffer Threshold. The `PKTE_IMSK_STAT.OBUFTHRSH` bit is set when the output buffer counter exceeds the output buffer threshold value defined in `PKTE_BUF_THRESH.OUTBUF` bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The `PKTE_IMSK_STAT.IBUFTHRSH` bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in `PKTE_BUF_THRESH.INBUF` bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The `PKTE_IMSK_STAT.RDRTHRSH` bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit, or the RD counter for the RDR in `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The `PKTE_IMSK_STAT.CDRTHRSH` bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Packet Engine Input Buffer Count Register

The `PKTE_INBUF_CNT` register provides the number of bytes available in the input buffer. The `PKTE_INBUF_CNT` register is used in direct host mode only.

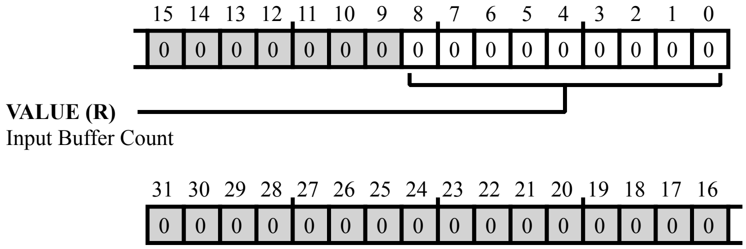

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Input Buffer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-24: PKTE_INBUF_CNT Register Diagram

Table 44-50: PKTE_INBUF_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Input Buffer Count. The `PKTE_INBUF_CNT.VALUE` bit field provides the number of bytes in the input buffer. The packet engine decrements the counter by 4 when a 32-bit word is read from the input buffer.

Packet Engine Input Buffer Count Increment Register

A host connected through the system slave bus can increment the input buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The `PKTE_INBUF_INCR` register is used in direct host mode only.

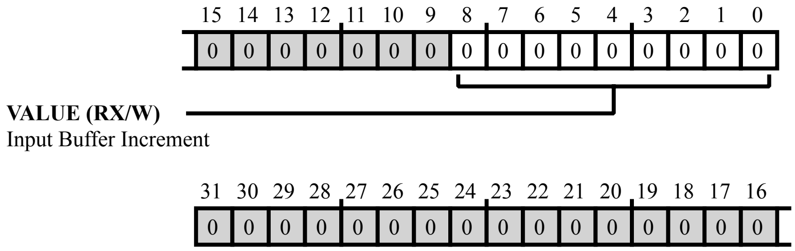

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (RX/W)
Input Buffer Increment
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-25: PKTE_INBUF_INCR Register Diagram

Table 44-51: PKTE_INBUF_INCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Input Buffer Increment. The value written is added to the input buffer counter. Valid values range from 4 to 256, in multiples of 4.

Interrupt Configuration Register

The PKTE_INT_CFG register configures the interrupt type that is sent to the interrupt line connected to the system interrupt controller. (Note that this only effects the final output of the interrupt subsystem).

Configuring the interrupt output type for pulse causes the interrupt signal to pulse low for two clock cycles when activated. When set for level, the interrupt signal is set low until cleared by the host (it follows the bit in the masked status register). For the host, this is typically set to level.

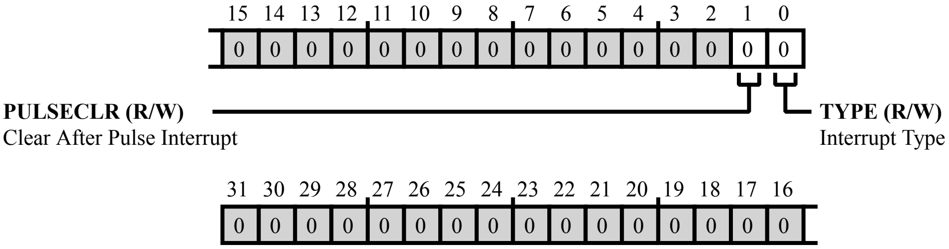


Figure 44-26: PKTE_INT_CFG Register Diagram

Table 44-52: PKTE_INT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PULSECLR	Clear After Pulse Interrupt. The PKTE_INT_CFG.PULSECLR bit clears the latched interrupt source after the pulse interrupt.	
		0	Manually clear pulse interrupt source. Do not automatically clear the interrupt sources after pulsing the interrupt output. Clear the source by writing to the PKTE_INT_CLR register.
		1	Automatically clear pulse interrupt source. After pulsing the interrupt output, automatically clear the sources.
0 (R/W)	TYPE	Interrupt Type. The PKTE_INT_CFG.TYPE bit selects the type, pulse or level, for the interrupt output to the system.	
		0	Level. The interrupt output is a level signal that is set low when an enabled interrupt is active until the interrupt is cleared.
		1	Pulse. The interrupt output is a two clock cycle low-active pulse, activated when an enabled interrupt is active.

Interrupt Clear Register

The `PKTE_INT_CLR` register allows the host processor to clear pending interrupts. A 1 written to a given bit in this register clears the corresponding interrupt. A 0 leaves the interrupt latch unchanged for that position.

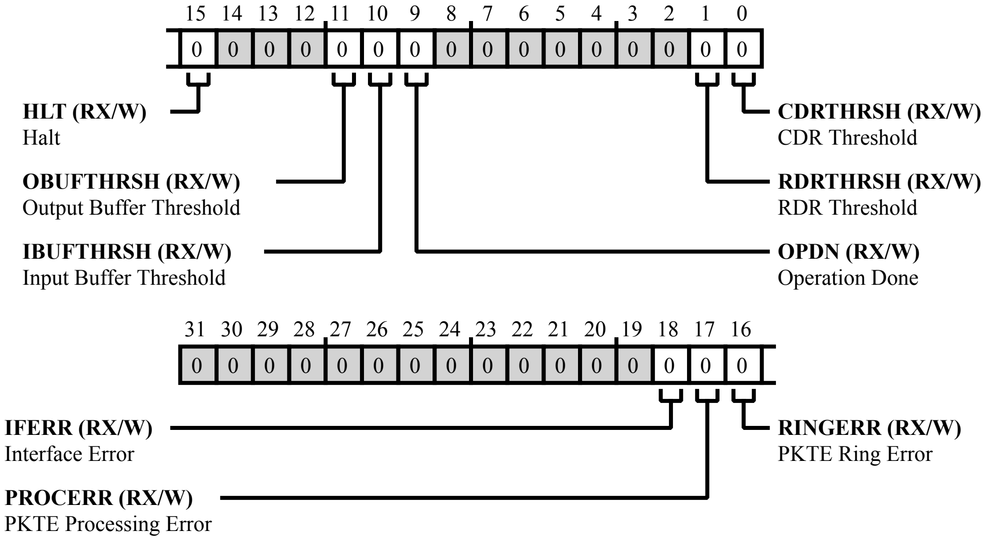

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (RX/W)
Halt
OBUFTHRSH (RX/W)
Output Buffer Threshold
IBUFTHRSH (RX/W)
Input Buffer Threshold
CDRTHRSH (RX/W)
CDR Threshold
RDRTHRSH (RX/W)
RDR Threshold
OPDN (RX/W)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (RX/W)
Interface Error
PROCERR (RX/W)
PKTE Processing Error
RINGERR (RX/W)
PKTE Ring Error

Figure 44-27: PKTE_INT_CLR Register Diagram

Table 44-53: PKTE_INT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. The `PKTE_INT_CLR.IFERR` bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. The `PKTE_INT_CLR.PROCERR` bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. The `PKTE_INT_CLR.RINGERR` bit is set on a CDR overflow or an RDR underflow.
15 (RX/W)	HLT	Halt. The `PKTE_INT_CLR.HLT` bit is set when the packet engine is in the HALT state.
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. The `PKTE_INT_CLR.OBUFTHRSH` bit is set when the output buffer counter exceeds the output buffer threshold value defined in `PKTE_BUF_THRESH.OUTBUF` bit.

Table 44-53: PKTE_INT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. The `PKTE_INT_CLR.IBUFTHRSH` bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in `PKTE_BUF_THRESH.INBUF` bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. The `PKTE_INT_CLR.RDRTHRSH` bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit, or the RD counter for the RDR in `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. The `PKTE_INT_CLR.CDRTHRSH` bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Interrupt Enable Register

The `PKTE_INT_EN` register configures the interrupt mask for the host interrupt. This register is a bitmap for each of the possible interrupt sources. A 1 enables the interrupt source and a 0 disables the source. If an interrupt source is disabled, a cleared bit also clears the matching interrupt in the `PKTE_IMSK_STAT` register.

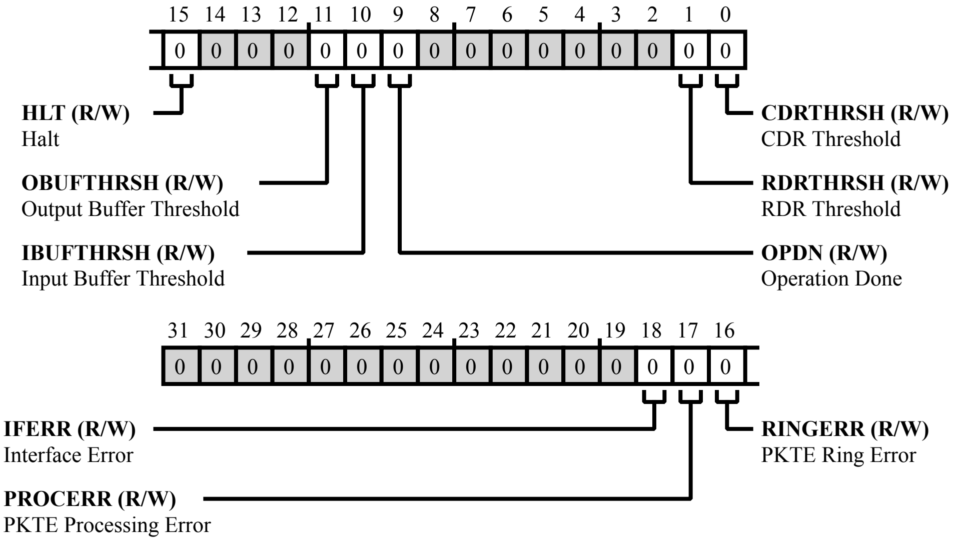

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (R/W)
Halt
OBUFTHRSH (R/W)
Output Buffer Threshold
IBUFTHRSH (R/W)
Input Buffer Threshold
CDRTHRSH (R/W)
CDR Threshold
RDRTHRSH (R/W)
RDR Threshold
OPDN (R/W)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (R/W)
Interface Error
PROCERR (R/W)
PKTE Processing Error
RINGERR (R/W)
PKTE Ring Error

Figure 44-28: PKTE_INT_EN Register Diagram

Table 44-54: PKTE_INT_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	IFERR	Interface Error. Set the `PKTE_INT_EN.IFERR` bit for host requests for a non 32-bit access to the packet engine interrupt or when the packet engine receives an error writing data back out to the host memory system.
17 (R/W)	PROCERR	PKTE Processing Error. Set the `PKTE_INT_EN.PROCERR` bit to enable the extended error occurred before, during or after processing the current packet in the packet engine interrupt.
16 (R/W)	RINGERR	PKTE Ring Error. Set the `PKTE_INT_EN.RINGERR` bit to enable the CDR overflow or RDR underflow interrupt.
15 (R/W)	HLT	Halt. Set the `PKTE_INT_EN.HLT` bit for when the packet engine is in the HALT state.
11 (R/W)	OBUFTHRSH	Output Buffer Threshold. Set the `PKTE_INT_EN.OBUFTHRSH` bit for to trigger an interrupt when the output buffer counter exceeds the output buffer threshold value defined in the `PKTE_BUF_THRESH.OUTBUF` bit.

Table 44-54: PKTE_INT_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	IBUFTHRSH	Input Buffer Threshold. Set the `PKTE_INT_EN.IBUFTHRSH` bit for to trigger an interrupt when the input buffer counter is less than or equal to the input buffer threshold value defined in the `PKTE_BUF_THRESH.INBUF` bit.
9 (R/W)	OPDN	Operation Done.
1 (R/W)	RDRTHRSH	RDR Threshold. Set the `PKTE_INT_EN.RDRTHRSH` bit for to trigger an interrupt when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH` bit, or the RD counter for the RDR in `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/W)	CDRTHRSH	CDR Threshold. Set the `PKTE_INT_EN.CDRTHRSH` bit for to trigger an interrupt when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Interrupt Unmasked Status Register

The `PKTE_IUMSK_STAT` register provides interrupt status visibility to the host, prior to the interrupt mask being applied. Using this register, the host can view all potential sources of incoming interrupts. All of these sources, whether masked in or out, are latched in this register and must be cleared using the `PKTE_INT_CLR` register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

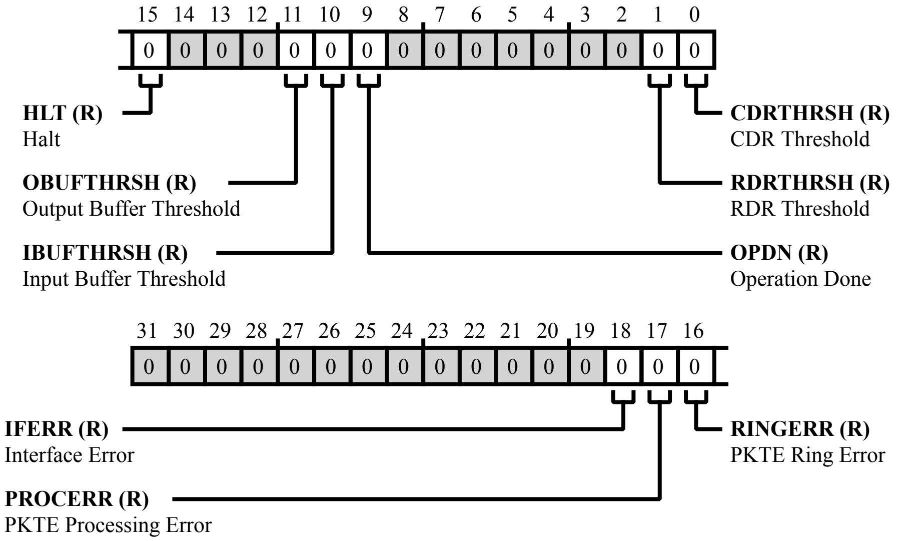

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HLT (R)
Halt
OBUFTHRSH (R)
Output Buffer Threshold
IBUFTHRSH (R)
Input Buffer Threshold
CDRTHRSH (R)
CDR Threshold
RDRTHRSH (R)
RDR Threshold
OPDN (R)
Operation Done
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IFERR (R)
Interface Error
PROCERR (R)
PKTE Processing Error
RINGERR (R)
PKTE Ring Error

Figure 44-29: PKTE_IUMSK_STAT Register Diagram

Table 44-55: PKTE_IUMSK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The `PKTE_IUMSK_STAT.IFERR` bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The `PKTE_IUMSK_STAT.PROCERR` bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PKTE Ring Error. The `PKTE_IUMSK_STAT.RINGERR` bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The `PKTE_IUMSK_STAT.HLT` bit is set when the packet engine is in the HALT state.

Table 44-55: PKTE_IUMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	OBUFTHRSH	Output Buffer Threshold. The `PKTE_IUMSK_STAT.OBUFTHRSH` interrupt is triggered when the output buffer counter exceeds the output buffer threshold value defined in `PKTE_BUF_THRESH.OUTBUF` bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The `PKTE_IUMSK_STAT.IBUFTHRSH` interrupt is triggered when the input buffer counter is less than or equal to the input buffer threshold value defined in `PKTE_BUF_THRESH.INBUF` bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The `PKTE_IUMSK_STAT.RDRTHRSH` bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the `PKTE_RING_THRESH.RDRTHRSH`, or the RD counter for the RDR in `PKTE_RDSC_CNT` register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The `PKTE_IUMSK_STAT.CDRTHRSH` bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the `PKTE_RING_THRESH.CDRTHRSH` bit.

Packet Engine Length Register

The PKTE_LEN register gives the length of the packet, the bypass data and a second set of ownership bits.

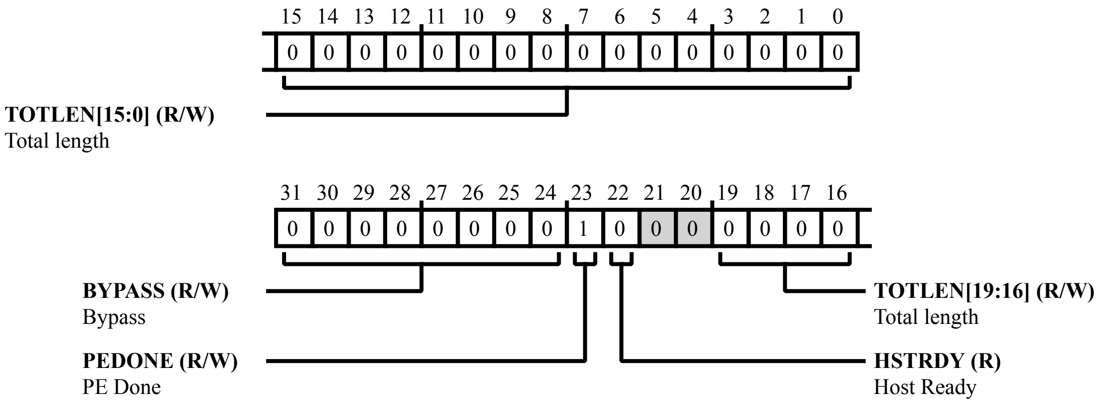

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TOTLEN[15:0] (R/W)
Total length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
BYPASS (R/W)
Bypass
PEDONE (R/W)
PE Done
TOTLEN[19:16] (R/W)
Total length
HSTRDY (R)
Host Ready

Figure 44-30: PKTE_LEN Register Diagram

Table 44-56: PKTE_LEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYPASS	Bypass. The PKTE_LEN.BYPASS bit field indicates the length of data in words that must bypass the packet engine and are directly copied from the source buffer to the destination buffer. The packet engine does not process this data. Valid bypass offsets range from 0 (0x00) to 255 (0xFF) words. For SRTP operations, this field specifies the offset in words between the hash and encrypt/decrypt data.
23 (R/W)	PEDONE	PE Done. The PKTE_LEN.PEDONE bit is a mirrored bit from the PKTE_CTL_STAT.PERDY bit. The bit is repeated here to guarantee ownership consistency between the first and last word. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.
22 (R/NW)	HSTRDY	Host Ready. The PKTE_LEN.HSTRDY bit is a mirrored bit of the PKTE_CTL_STAT.HOSTRDY bit. The bit is repeated here to guarantee ownership consistency between the first and last word. It should also be set along with the PKTE_CTL_STAT.HOSTRDY bit when the command descriptor is finished being populated. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.

Table 44-56: PKTE_LEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/W)	TOTLEN	Total length. Command Descriptor: The `PKTE_LEN.TOTLEN` bit field indicates the total length (in bytes) of all data to be passed to the packet engines input buffer for an operation. Exceptions are the PRNG init and PRNG generate operations. The PRNG init operation does not require any input data; this field must be zero. For the PRNG generate operation, this field indicates the number of pseudo-random bytes to be generated. Valid lengths range from 16 (0x00010) to 255*16 = 4080 (0x00FF0) bytes in multiples of 16 bytes. Valid lengths for the basic operation range from 1 (0x00001) to 1,048,575 (0xFFFFF) bytes. This is the length of the data to be encrypted or hashed and includes the bypass data and padding bytes. Valid lengths for IPsec ESP range from 1 (0x00001) to 65535 (0x0FFFF) bytes. This is the length of the IP payload. Valid lengths for SSL v3.0, TLS v1.x and DTLS range from 1 (0x00001) to 16383 (0x03FFF). This is the length of the payload. Valid lengths for SRTP range from 1 (0x00001) to 65535 (0x0FFFF). This is the length of the payload. Note: A length of zero bytes is illegal and will result in an error status code in the result descriptor. Result Descriptor: Upon completion of an operation, the `PKTE_LEN.TOTLEN` field indicates the result length of the result packet. Valid lengths range from 1 (0x001) to 1,048,575 (0xFFFFF) bytes. This includes the bypass data and padding bytes. Note: When an extended error (`PKTE_CTL_STAT`[18]=1) is reported in the result descriptor and no packet data is processed, this field returns zero.

Packet Engine Output Buffer Count Register

The `PKTE_OUTBUF_CNT` register provides the number of data bytes there are in the output buffer. The `PKTE_OUTBUF_CNT` register is used in direct host mode only.

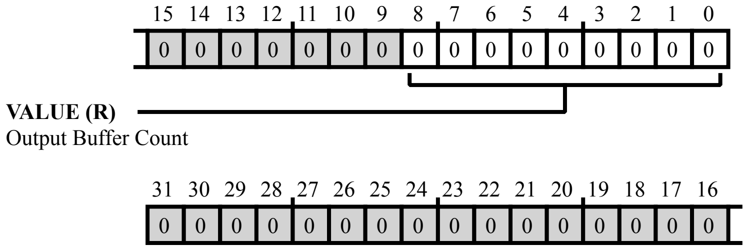

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Output Buffer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-31: PKTE_OUTBUF_CNT Register Diagram

Table 44-57: PKTE_OUTBUF_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Output Buffer Count. The `PKTE_OUTBUF_CNT.VALUE` bit field provides the number of bytes in the output buffer. The packet engine increments the counter by 4 when a 32-bit word is written to the output buffer.

Packet Engine Output Buffer Count Decrement Register

A host connected via the system slave bus can decrement the output buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The `PKTE_OUTBUF_DECR` register is used in direct host mode only.

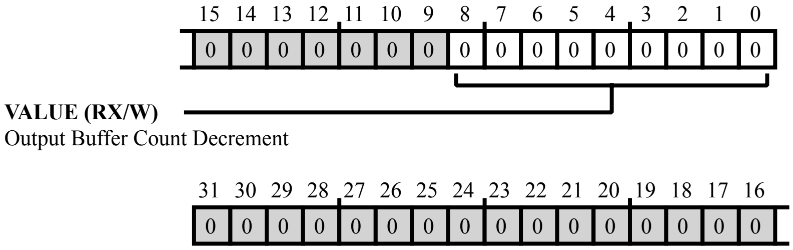

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (RX/W)
Output Buffer Count Decrement
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-32: PKTE_OUTBUF_DECR Register Diagram

Table 44-58: PKTE_OUTBUF_DECR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Output Buffer Count Decrement. The `PKTE_OUTBUF_DECR.VALUE` bit field is the value written is subtracted to the output buffer counter. Valid values range from 4 to 256, in multiples of 4.

Packet Engine Result Descriptor Ring Base Address

The PKTE_RDRBASE_ADDR register holds the result descriptor ring base address in host memory. It is only applicable in autonomous ring mode and target command mode with RDR enabled. Note that in target command mode, the CDR is not used, but the RDR must be configured when enabled so that the packet engine knows where to write the result descriptors.

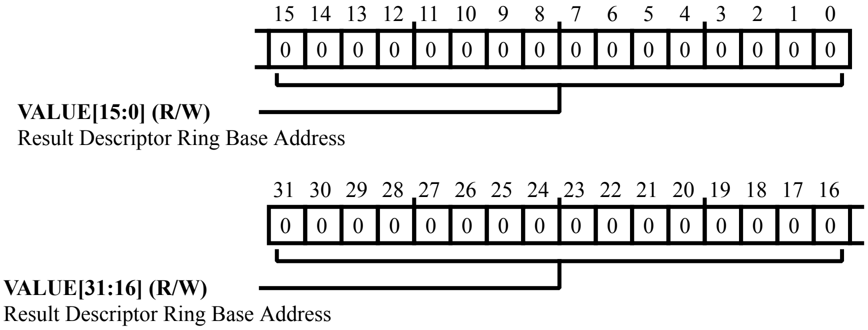

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Result Descriptor Ring Base Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Result Descriptor Ring Base Address

Figure 44-33: PKTE_RDRBASE_ADDR Register Diagram

Table 44-59: PKTE_RDRBASE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Result Descriptor Ring Base Address. The `PKTE_RDRBASE_ADDR.VALUE` bit field specifies the base location of the result descriptor ring in the host memory space.

Packet Engine Result Descriptor Count Registers

The PKTE_RDSC_CNT register holds the counter for the number of descriptors in the Result Descriptor Ring (RDR). It is incremented by the packet engine each time a valid result descriptor is written to the RDR.

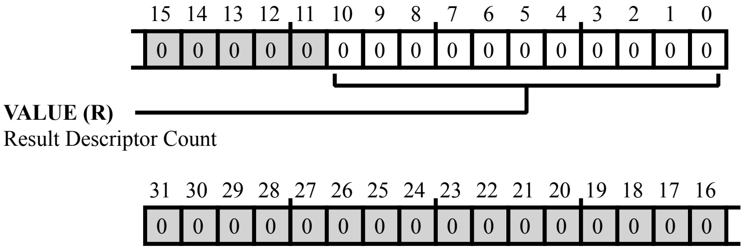

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Result Descriptor Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-34: PKTE_RDSC_CNT Register Diagram

Table 44-60: PKTE_RDSC_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Result Descriptor Count. The PKTE_RDSC_CNT.VALUE bit field provides the number of result descriptors in the result descriptor ring. The packet engine increments the counter when a valid result descriptor is written to the RDR.

Packet Engine Result Descriptor Count Decrement Registers

The PKTE_RDSC_DECR register is accessible by the host connected through the system slave bus can decrement the result descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

With an RDR enabled, this is the number of result descriptors that have been read by the host. With an RDR disabled, this indicates that the host has read one valid result descriptor.

In autonomous ring mode or target command mode with the RDR enabled, the host must process 1 to 255 result descriptors from the RDR and then write this register with the number of result descriptors that have been processed by the host.

In direct host mode or target command mode with the RDR disabled, the host must read one result descriptor from the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is read. An RDR threshold interrupt is activated when the result descriptor counter exceeds the threshold value set in the PKTE_RING_THRESH register. This interrupt can be used to wake up a process that stalled on an empty RDR.

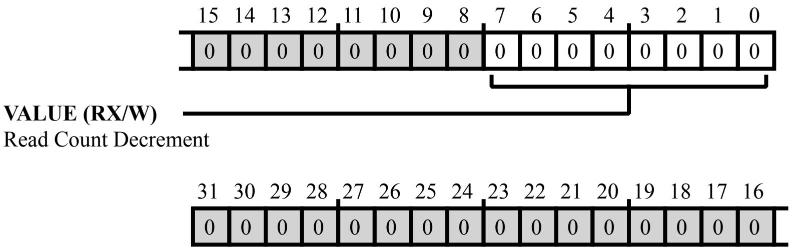


Figure 44-35: PKTE_RDSC_DECR Register Diagram

Table 44-61: PKTE_RDSC_DECR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Read Count Decrement. The value written to the `PKTE_RDSC_DECR.VALUE` bit field is subtracted from the result descriptor counter. The counter is protected against underflow (See the PKTE_RING_STAT register). Note that bits [10:8] should be written with zeros.

Packet Engine Ring Configuration

The PKTE_RING_CFG register configures the size (in number of descriptor ring entries minus 1) for both the command descriptor ring and result descriptor ring in host memory. This register is only applicable for autonomous ring mode and target command mode with RDR enabled.

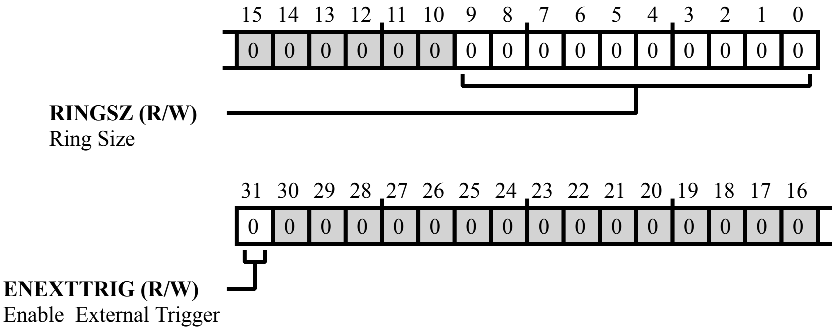

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RINGSZ (R/W)
Ring Size
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENEXTTRIG (R/W)
Enable External Trigger

Figure 44-36: PKTE_RING_CFG Register Diagram

Table 44-62: PKTE_RING_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ENEXTTRIG	Enable External Trigger. The PKTE_RING_CFG.ENEXTTRIG signal enables the increment of the PKTE_CDSC_CNT register through the external input pin ext_cd_cnt_incr and enables the decrement of the PKTE_RDSC_CNT fields through the external input pin ext_rd_cnt_decr.
9:0 (R/W)	RINGSZ	Ring Size. The PKTE_RING_CFG.RINGSZ bit field specifies the size of the command ring in number of descriptors, minus 1. Valid sizes range from 1 (for 2 descriptors) to 1023 (for 1024 descriptors). The accompanying result ring will have the same size.

Packet Engine Ring Pointer Status

The PKTE_RING_PTR register holds the pointers to the current entry of the Command Descriptor Ring (CDR) and Result Descriptor Ring (RDR).

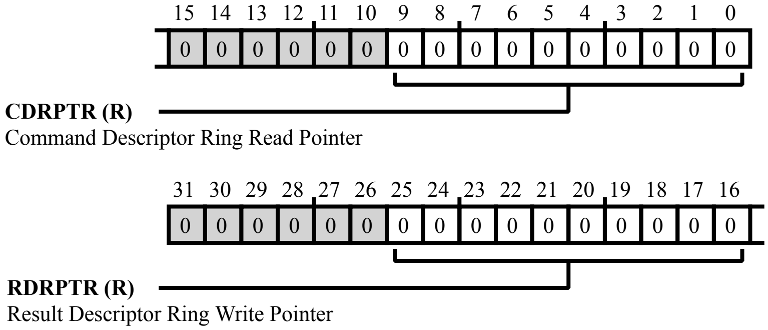

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CDRPTR (R)
Command Descriptor Ring Read Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDRPTR (R)
Result Descriptor Ring Write Pointer

Figure 44-37: PKTE_RING_PTR Register Diagram

Table 44-63: PKTE_RING_PTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/NW)	RDRPTR	Result Descriptor Ring Write Pointer. The PKTE_RING_PTR.RDRPTR bit field indicates the entry number in the RDR that will be written next by the packet engine. The PKTE_RING_PTR.RDRPTR bit field is reset to zero after starting up and updated after every result descriptor write DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (PKTE_RING_CFG.RINGSZ) bit field.
9:0 (R/NW)	CDRPTR	Command Descriptor Ring Read Pointer. The PKTE_RING_PTR.CDRPTR bit field indicates the entry number in the CDR that will be read next by the packet engine. The PKTE_RING_PTR.CDRPTR bit field is reset to zero after starting up and updated after every command descriptor read DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (PKTE_RING_CFG.RINGSZ) field.

Packet Engine Ring Status

The `PKTE_RING_STAT` register gives indication of either a Command Descriptor Ring (CDR) overflow or a Result Descriptor Ring (RDR) underflow. A ring error (ringerr) interrupt in the interrupt controller is activated on a command descriptor ring overflow or a result descriptor ring underflow. This type of error can occur when the host and the packet engine get out-of-sync. The host can read this register to retrieve information on which ring is corrupted. The corrupted ring must be reset and reinitialized. See the `PKTE_CFG.RSTRING` bit.

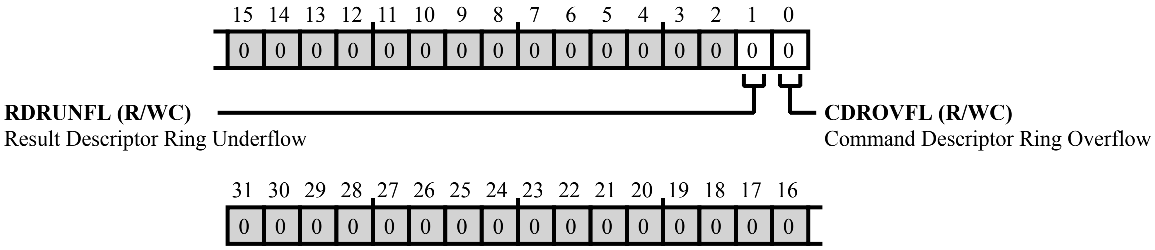

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDRUNFL (R/WC)
Result Descriptor Ring Underflow
CDROVFL (R/WC)
Command Descriptor Ring Overflow
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-38: PKTE_RING_STAT Register Diagram

Table 44-64: PKTE_RING_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/WC)	RDRUNFL	Result Descriptor Ring Underflow. The `PKTE_RING_STAT.RDRUNFL` bit is set when the command descriptor count (`PKTE_RDSC_CNT`) register is decremented below zero. This bit is reset with a write of any value.
0 (R/WC)	CDROVFL	Command Descriptor Ring Overflow. The `PKTE_RING_STAT.CDROVFL` bit is set when the command descriptor count (`PKTE_CDSC_CNT`) register is incremented above the ring size (`PKTE_RING_CFG.RINGSZ`) bits. This bit is reset with a write of any value.

Packet Engine Ring Threshold Registers

To reduce the amount of packet engine result interrupts, the PKTE_RING_THRESH register contains threshold and time-out values.

The CDR threshold (cdrthrsh) interrupt indicates that the command descriptor counter is less than or equal to the CDR threshold (cdrthrsh) value set in this register. This interrupt can be used to wake up a process that stalled on a full CDR.

The RDR threshold (rdrthrsh) interrupt indicates that the result descriptor counter exceeds the result descriptor threshold set here, or that the result descriptor counter is non-zero for a time longer than the result descriptor time-out setting. The RDR result interrupt remains active until the result descriptor counter is decremented below the RDR threshold (rdrthrsh) value. In case the interrupt is the result of a time-out and the result descriptor counter is below the threshold value, the result descriptor counter must be decremented once before the interrupt can be cleared in the interrupt controller.

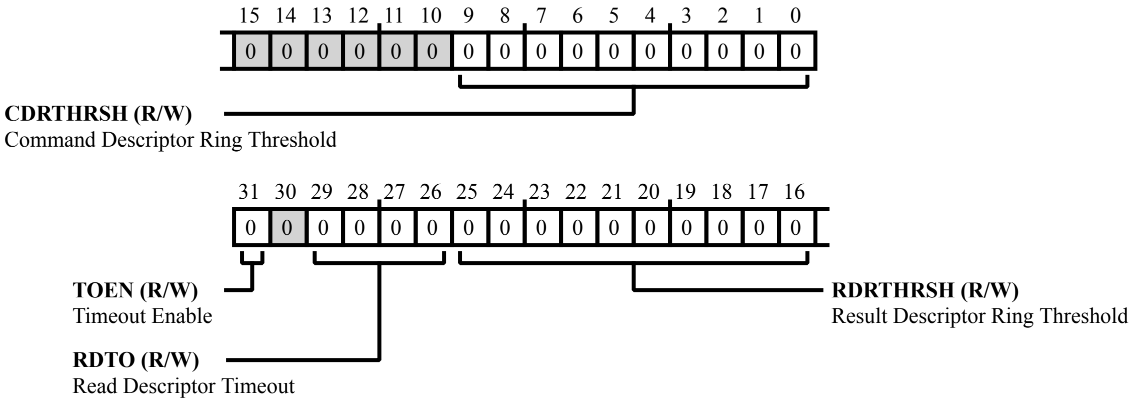

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CDRTHRSH (R/W)
Command Descriptor Ring Threshold
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TOEN (R/W)
Timeout Enable
RDTO (R/W)
Read Descriptor Timeout
RDRTHRSH (R/W)
Result Descriptor Ring Threshold

Figure 44-39: PKTE_RING_THRESH Register Diagram

Table 44-65: PKTE_RING_THRESH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TOEN	Timeout Enable. A 1 in the `PKTE_RING_THRESH.TOEN` bit indicates the result descriptor timeout counter is enabled. This bit can be used to de-activate the timeout counter to save power.

Table 44-65: PKTE_RING_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:26 (R/W)	RDTO	Read Descriptor Timeout. The timeout enable (`PKTE_RING_THRESH.TOEN`) bit in this register must be set to activate this `PKTE_RING_THRESH.RDTO` result descriptor timeout counter. The rdrthrsh interrupt activates when the RD counter for the RDR is non-zero for more than $2^{(N+10)}$ internal system clock cycles, where 'N' is the value set in this field. Valid settings range from 0 to 15. The minimum time-out value for N=0 is 1024 clock cycles and the maximum time-out value for N=15 is 33554432 clock cycles. At 100 MHz, this is 5.12 us for N=0 and ~335.55 ms for N=15. Note: The time-out delay may not be exact - expect a variation on the order of 1024 system clock cycles (just more than one microsecond at 100 MHz system clock frequency).
25:16 (R/W)	RDRTHRSH	Result Descriptor Ring Threshold. The rdrthrsh interrupt activates when the RD counter for the RDR exceeds the value set in the `PKTE_RING_THRESH.RDRTHRSH` field. Valid settings range from 0 to 1023.
9:0 (R/W)	CDRTHRSH	Command Descriptor Ring Threshold. The cdrthrsh interrupt activates when CD counter for the CDR is below or equal the value set in the `PKTE_RING_THRESH.CDRTHRSH` field. Valid settings range from 0 to 1023.

Packet Engine SA Address

The `PKTE_SA_ADDR` register holds the start address of the SA record.

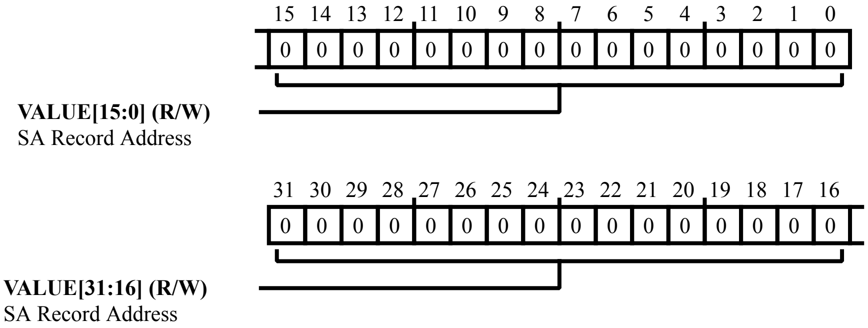

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
SA Record Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
SA Record Address

Figure 44-40: PKTE_SA_ADDR Register Diagram

Table 44-66: PKTE_SA_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Record Address. The `PKTE_SA_ADDR.VALUE` bit field holds the start address of the SA record.

ARC4 i and j Pointer Register

When starting a new ARC4 operation the `PKTE_SA_ARC4IJPTR` register contains the initialization value, which is zeros. After processing the ARC4 algorithm it contains the latest status of the ARC4_IJ_PNTR.

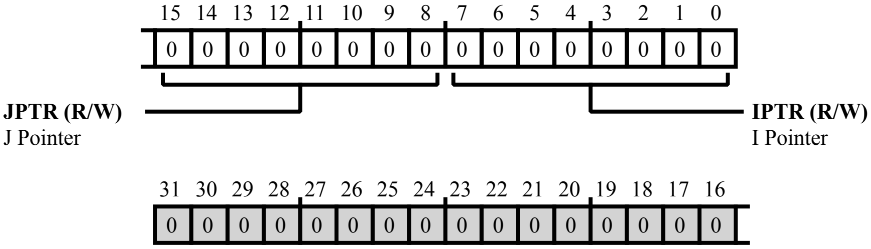

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
JPTR (R/W)
J Pointer
IPTR (R/W)
I Pointer
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 44-41: PKTE_SA_ARC4IJPTR Register Diagram

Table 44-67: PKTE_SA_ARC4IJPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	JPTR	J Pointer. The `PKTE_SA_ARC4IJPTR.JPTR` bit field contains the j pointer into s-box array for swapping bytes with i pointer.
7:0 (R/W)	IPTR	I Pointer. The `PKTE_SA_ARC4IJPTR.IPTR` bit field contains the i pointer into s-box array for swapping bytes with j pointer.

SA Command 0

The two SA command registers, `PKTE_SA_CMD0` and `PKTE_SA_CMD1`, are used to control the cryptographic operation of the packet engine. The `PKTE_SA_CMD0` register contains the major control bits to define an operation while the `PKTE_SA_CMD1` register contains the minor control bits. In direct host mode, this is a write-only register.

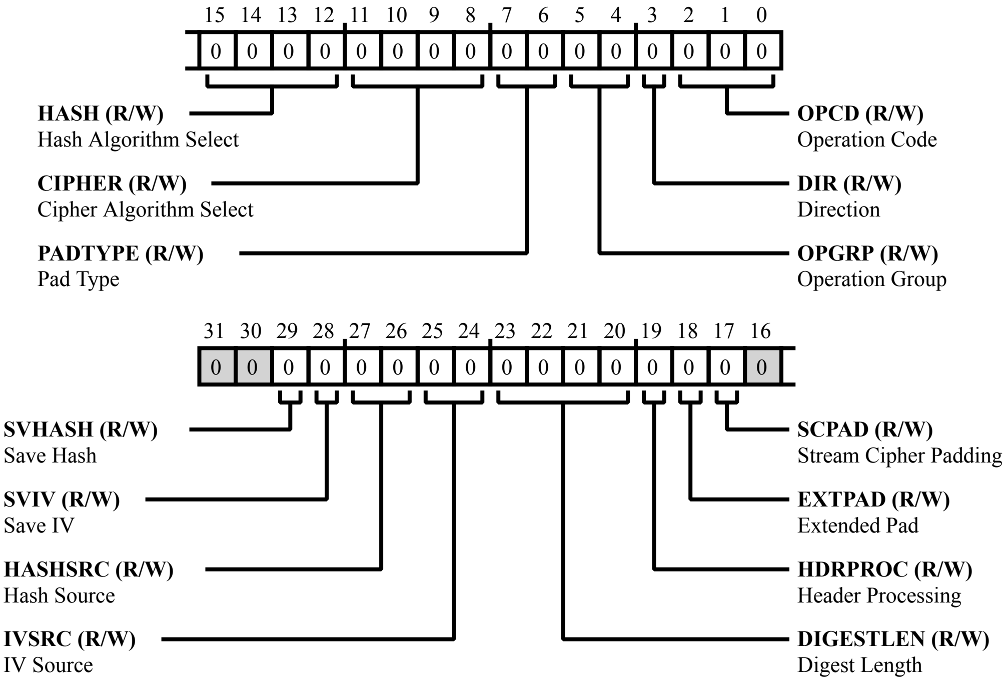

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HASH (R/W)
Hash Algorithm Select
CIPHER (R/W)
Cipher Algorithm Select
PADTYPE (R/W)
Pad Type
OPCD (R/W)
Operation Code
DIR (R/W)
Direction
OPGRP (R/W)
Operation Group
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SVHASH (R/W)
Save Hash
SVIV (R/W)
Save IV
HASHSRC (R/W)
Hash Source
IVSRC (R/W)
IV Source
SCPAD (R/W)
Stream Cipher Padding
EXTPAD (R/W)
Extended Pad
HDRPROC (R/W)
Header Processing
DIGESTLEN (R/W)
Digest Length

Figure 44-42: PKTE_SA_CMD0 Register Diagram

Table 44-68: PKTE_SA_CMD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	SVHASH	Save Hash. The `PKTE_SA_CMD0.SVHASH` bit indicates that the Hash State is saved to the STATE_BYTE_CNT_X and STATE_IDIGEST_X fields in the SA record in memory after completion of a crypto operation.	
		0	Hash state is not saved
		1	Hash state is saved

Table 44-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	SVIV	Save IV. The `PKTE_SA_CMD0.SVIV` bit field indicates that for DES or the AES the Initialization Vector (IV) is saved to the STATE_IV_X fields in the state record , or the ARC4 state is saved to the ARC4 state record, after completion of the crypto operation.	
		0	ARC4 State is not saved.
		1	ARC4 State is not saved.
27:26 (R/W)	HASHSRC	Hash Source. The `PKTE_SA_CMD0.HASHSRC` bit field selects the source of the hash digest used by the algorithm.	
		0	From SA. Digest only hash byte count is forced to 0x40.
		1	Reserved
		2	From State. Read saved inner hash digest and saved hash byte count.
		3	No Load. Use the hash algorithm defined constants for the initial hash. Hash byte count is 0x00.
25:24 (R/W)	IVSRC	IV Source. The `PKTE_SA_CMD0.IVSRC` bit field selects the source of the initialization vector used by the crypto algorithm.	
		0	No load. Use previous result IV, not applicable for inbound data. This option should never be used for operations with DES-CBC or AES-CBC, (see RFC3602) or any AES counter modes
		1	From input buffer. The IV is provided as part of the input data stream.
		2	From State. Read STATE_IV_X, from the SA structure. Refer to inner hash digest register structure. Useful for resume operations.
		3	From internal PRNG. Not applicable for inbound operations.
23:20 (R/W)	DIGESTLEN	Digest Length. The `PKTE_SA_CMD0.DIGESTLEN` bit field defines the length of the hash digest in words as put in the output buffer.	
		0	3 Words (96-bit output)
		1	1 Word
		2	2 Words

Table 44-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	3 Words (IPsec)
		4	4 Words (MD5 and AES-based hash)
		5	5 Words (SHA-1)
		6	6 Words
		7	7 Words (SHA-224)
		8	8 Words (SHA-256)
		9	Reserved
		10	10 bytes (SRTP and TLS)
		11-15	Reserved
19 (R/W)	HDRPROC	Header Processing. The `PKTE_SA_CMD0.HDRPROC` bit enables header processing for protocol operations. There is no header-processing support for basic SSL, basic TLS and SRTP protocol operations as defined in the protocol group (see the Crypto and Hash Algorithms section). This bit must be zero for these operations; however, the protocol header must be supplied to the packet engine since it is part of the hash calculation. Refer to the protocol specifications for more information about header-processing support for a protocol.	
		0	No header processing
		1	Header processing; insert the protocol header for outbound operations, verify the protocol header for inbound operations.
18 (R/W)	EXTPAD	Extended Pad. The `PKTE_SA_CMD0.EXTPAD` bit extends the number of padding types. Used in combination with `PKTE_SA_CMD0.PADTYPE`.	
17 (R/W)	SCPAD	Stream Cipher Padding. The `PKTE_SA_CMD0.SCPAD` bit enables padding for stream ciphers algorithms.	
15:12 (R/W)	HASH	Hash Algorithm Select. The `PKTE_SA_CMD0.HASH` bit field selects the hash algorithm.	
		0	MD5
		1	SHA-1
		2	SHA-224
		3	SHA-256
		4-14	Reserved
		15	Null

Table 44-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	CIPHER	Cipher Algorithm Select. The `PKTE_SA_CMD0.CIPHER` bit field selects the cipher algorithm to be used for encryption and decryption. Note: Each type of protocol operation supports different sets of crypto algorithms. Refer to the Crypto and Hash Algorithms general processing section for details of the supported algorithms.	
		0	DES
		1	Triple-DES
		2	ARC4
		3	AES
		4-14	Reserved
		15	Null
7:6 (R/W)	PADTYPE	Pad Type. The `PKTE_SA_CMD0.PADTYPE` bit field indicates the type of crypto that must be generated for outbound packets or checked for inbound packets.	
		0	Select IPSec operation (if Bit 18=0); Reserved (if Bit 18=1)
		1	PKCS#7 (if Bit 18=0); Select TLS/DTLS Pad, required for TLS/DTLS operation (if Bit 18=1)
		2	Constant pad (if Bit 18=0); Select Constant SSL Pad, required for SSL operation (if Bit 18=1)
		3	Zero pad (if Bit 18=0), Reserved (if Bit 18=1)
5:4 (R/W)	OPGRP	Operation Group. The `PKTE_SA_CMD0.OPGRP` bit field defines the operation groups. Refer to the Basic Operations and Decoding section for more information.	
		0	Basic operation group
		1	Protocol operation group
		2	Extended protocol operations group
		3	Reserved
3 (R/W)	DIR	Direction. The `PKTE_SA_CMD0.DIR` bit field selects the direction of operation.	
		0	Outbound operations
		1	Inbound operations

Table 44-68: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	OPCD	Operation Code. The `PKTE_SA_CMD0.OPCD` bit field selects the operation within the operation group.

SA Command 1

The PKTE_SA_CMD1 register contains the minor control bits that define an operation. In direct host mode, this is a write-only register.

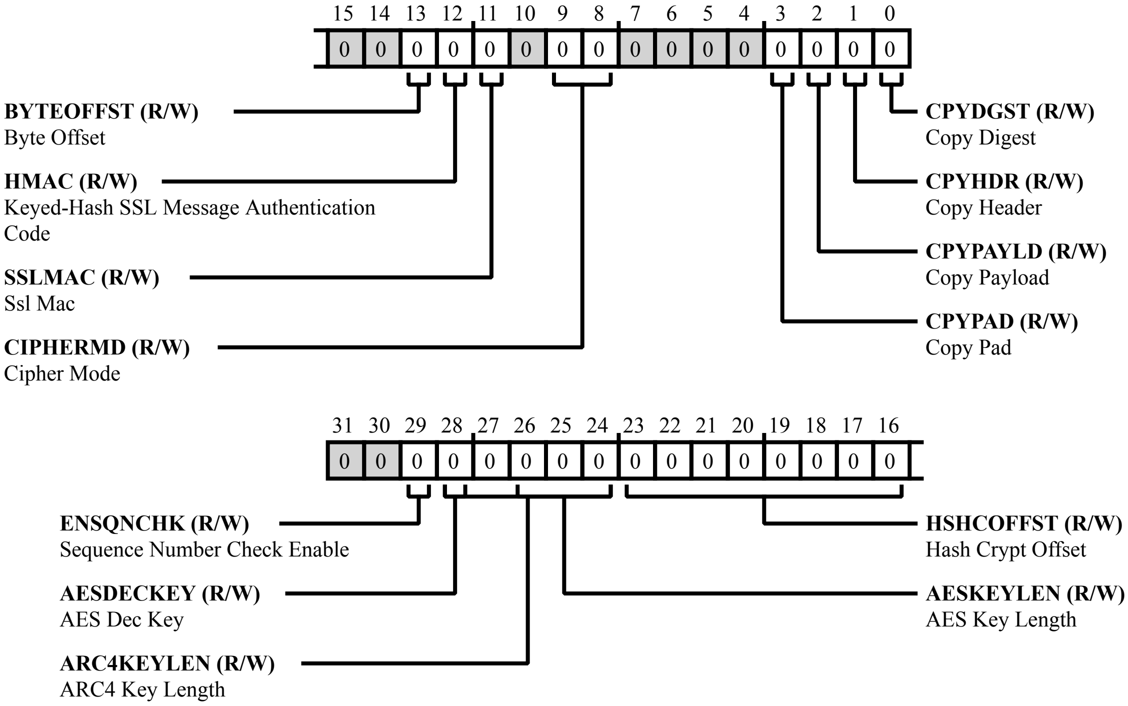

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BYTEOFFST (R/W)
Byte Offset
HMAC (R/W)
Keyed-Hash SSL Message Authentication Code
SSLMAC (R/W)
Ssl Mac
CIPHERMD (R/W)
Cipher Mode
CPYDGST (R/W)
Copy Digest
CPYHDR (R/W)
Copy Header
CPYPAYLD (R/W)
Copy Payload
CPYPAD (R/W)
Copy Pad
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ENSQNCHK (R/W)
Sequence Number Check Enable
AESDECKEY (R/W)
AES Dec Key
ARC4KEYLEN (R/W)
ARC4 Key Length
HSHCOFFST (R/W)
Hash Crypt Offset
AESKEYLEN (R/W)
AES Key Length

Figure 44-43: PKTE_SA_CMD1 Register Diagram

Table 44-69: PKTE_SA_CMD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	ENSQNCHK	Sequence Number Check Enable. The PKTE_SA_CMD1.ENSQNCHK bit defines that the key in the SA key field is an AES encrypt key or an AES decrypt key.	
		0	Disable sequence number check
		1	Enable sequence number check
28 (R/W)	AESDECKEY	AES Dec Key. If the PKTE_SA_CMD1.AESDECKEY bit is set, the key in loaded in the PKTE_SA_KEY[n] registers are expected to be the key from the last round from key expansion. If not set, the key loaded in the PKTE_SA_KEY[n] registers are expected to be the same key used during the encryption process.	
		0	AES key is an encrypt key.
		1	AES key is a decrypt key.

Table 44-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26:24 (R/W)	AESKEYLEN	AES Key Length. The `PKTE_SA_CMD1.AESKEYLEN` bit field select the size of the key used for the AES algorithm in increments of 64 bits.	
		0-1	Reserved
		2	128 Bits
		3	192 Bits
		4	256 Bits
		5-7	Reserved
28:24 (R/W)	ARC4KEYLEN	ARC4 Key Length.	
23:16 (R/W)	HSHCOFFST	Hash Crypt Offset. For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the `PKTE_SA_CMD1.HSHCOFFST` bit field specifies the offset between the hash data and the encrypt/decrypt data. The data to be hashed is assumed to come first, with an offset to the beginning of encrypt/decrypt data. When `PKTE_SA_CMD1.BYTEOFFST`, bit 13, is zero, then the offset is defined in 32-bit words. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 0 to 255. When `PKTE_SA_CMD1.BYTEOFFST`, bit 13, is one, then the offset is defined in 8-bit bytes. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 4 to 255. (The IV size is two words for DES, Triple-DES and AES-CTR and four words for AES-CBC and AES-ICM operations). Other operations do not use these bits (a default value is applied by the packet engine).	
13 (R/W)	BYTEOFFST	Byte Offset. The `PKTE_SA_CMD1.BYTEOFFST` bit defines how the `PKTE_SA_CMD1.HSHCOFFST`, bits of this register are used.	
		0	HASH_CRYPT_OFFSET is defined in 32-bit words
		1	HASH_CRYPT_OFFSET is defined in 8-bit bytes
12 (R/W)	HMAC	Keyed-Hash SSL Message Authentication Code. For basic operations that include hashing, the `PKTE_SA_CMD1.HMAC` bit enables the HMAC processing, which calls for an extra outer hash operation.	
		0	Standard Hash
		1	HMAC Processing

Table 44-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	SSLMAC	Ssl Mac.	
		0	Standard Hash
		1	SSL-MAC processing
9:8 (R/W)	CIPHERMD	Cipher Mode. The `PKTE_SA_CMD1.CIPHERMD` bit field selects the crypto mode to be used for the cipher algorithm.	
		0	Electronic Code Book (ECB) used for DES and AES
		1	Cipher Block Chaining (CBC) used for DES and AES
		2	AES Counter Mode (CTR) for IPsec using a 32-bit counter
		3	AES Integer Counter Mode (ICM) for SRTP using a 16-bit counter. Note: This is implemented as a 32-bit counter, the Host must check for an overflow from the value 0xFF to zero and then refresh the IV.
3 (R/W)	CPYPAD	Copy Pad. The `PKTE_SA_CMD1.CPYPAD` bit indicates that the padding data for an inbound operation is copied to the output buffer and saved in memory.	
		0	Do not copy the padding to output
		1	Copy padding to output
2 (R/W)	CPYPAYLD	Copy Payload. The `PKTE_SA_CMD1.CPYPAYLD` bit indicates that the payload data is copied to the output buffer and saved in memory.	
		0	Do not copy the payload to output
		1	Copy payload to output
1 (R/W)	CPYHDR	Copy Header. The `PKTE_SA_CMD1.CPYHDR` bit indicates that the protocol header is copied to the output buffer and saved in memory. For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the header is defined as the Hash/Crypt Offset data (authenticated only).	
		0	Do not copy the header to output
		1	Copy header to output

Table 44-69: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	CPYDGST	Copy Digest. The `PKTE_SA_CMD1.CPYDGST` bit copies the hash result is to the output buffer and saves in memory. The length of the hash result is defined by the `PKTE_SA_CMD0.DIGESTLEN` field.	
		0	Do not copy hash result to output
		1	Copy hash result to output, when the command descriptor `PKTE_CTL_STAT.HASHFINAL` bit is set.

SA Inner Hash Digest Registers

The `PKTE_SA_IDIGEST[n]` registers are a set of eight 32-bit read/write registers.

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers are used to enter a start hash state, and to read the interim or final hash digest.

For IPsec, TLS and DTLS operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with basic hash or HMAC authentication with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the pre-computed inner hash digest. This is the hash of the hash-key padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the inner hash pre-compute; this is the hash of the MAC_WRITE_SECRET padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the MAC_WRITE_SECRET. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the hash-key pad (0x36 hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

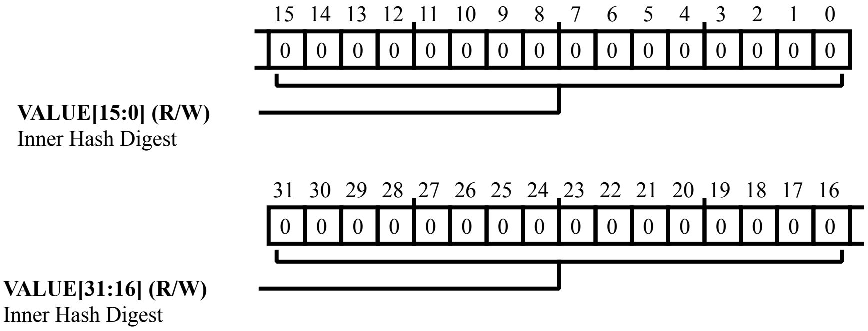


Figure 44-44: PKTE_SA_IDIGEST[n] Register Diagram

Table 44-70: PKTE_SA_IDIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Hash Digest.

SA Key Registers

These are the `PKTE_SA_KEY[n]` registers for DES, Triple-DES, ARC4 and AES: A set of eight 32-bit write only registers. The reset value of these registers is zero.

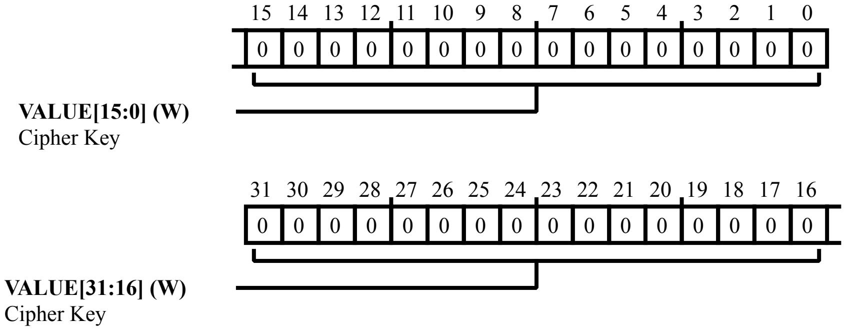

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (W)
Cipher Key
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (W)
Cipher Key

Figure 44-45: PKTE_SA_KEY[n] Register Diagram

Table 44-71: PKTE_SA_KEY[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Cipher Key.

SA Initialization Vector Register

The `PKTE_SA_NONCE` register is used for operations that make use of the IV value loaded from the SA record. This register is used both to enter a starting IV state, as well as for reading the interim or final IV. For IPsec outbound operations, it is recommended that the automatic IV insertion mode be used, this register is not needed. For IPsec inbound operations, the IV is extracted from the header of the packet.

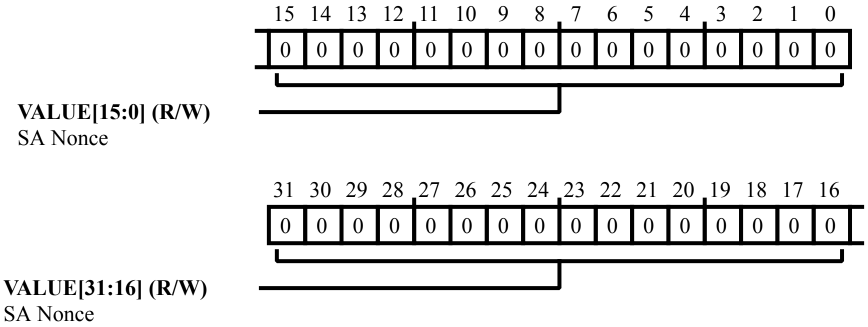

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
SA Nonce
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
SA Nonce

Figure 44-46: PKTE_SA_NONCE Register Diagram

Table 44-72: PKTE_SA_NONCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Nonce.

SA Outer Hash Digest Registers

The `PKTE_SA_ODIGEST[n]` registers are a set of five eight 32-bit write-only registers.

For write operations, these registers contain the pre-computed outer hash digest for IPsec operations with basic HMAC operations with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA).

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers hold a start hash state, or the interim outer hash digest. They are only used for HMAC processing.

For IPsec, SSL, TLS, DTLS and SRTP operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with HMAC authentication with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the pre-computed outer hash digest. This is the hash of the hash-key padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the outer hash pre-compute; this is the hash of the MAC_WRITE_SECRET padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA), these registers hold the MAC_WRITE_SECRET. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the required hash-key pad (0x5C hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

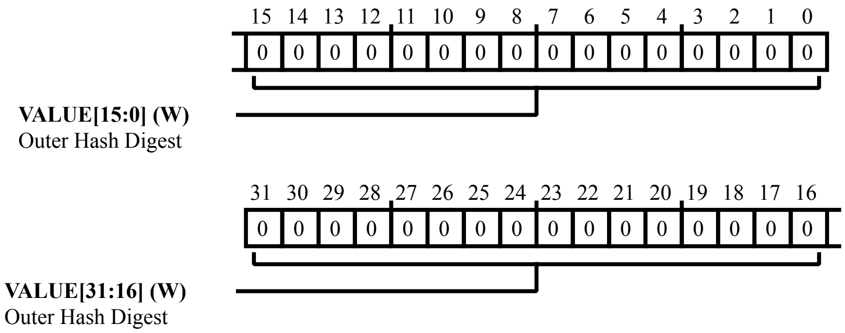


Figure 44-47: PKTE_SA_ODIGEST[n] Register Diagram

Table 44-73: PKTE_SA_ODIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Outer Hash Digest.

SA Ready Indicator

In direct host mode, a write to the `PKTE_SA_RDY` register triggers the packet engine to start processing using the command descriptor, SA record and state record in the packet engine registers. This register MUST be written for all direct host mode packet operations. It is intended that this register is written in sequence; as the entire SA record is written.

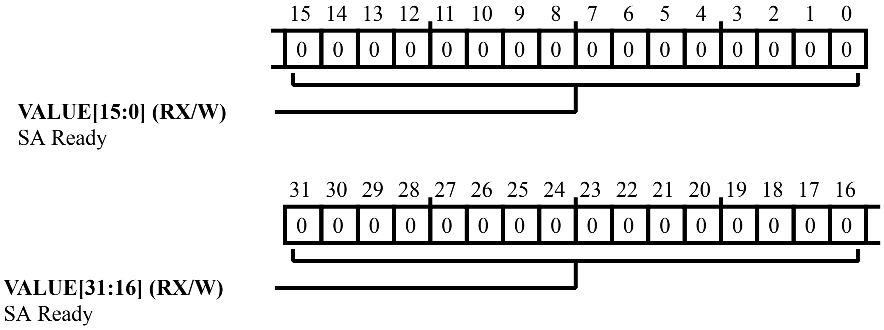

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (RX/W)
SA Ready
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (RX/W)
SA Ready

Figure 44-48: PKTE_SA_RDY Register Diagram

Table 44-74: PKTE_SA_RDY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	SA Ready.

SA Sequence Number Register

The `PKTE_SA_SEQNUM[n]` registers are a set of two read/write registers and are used for IPsec ESP, SSL, TLS, DTLS operations to specify the anti-replay sequence number value that is to be placed in the ESP header (outbound), or to be checked against for inbound packets. The packet engine manages this counter value for both inbound and outbound operations.

Outbound: The host writes the counter value stored in the SA record to this register to start an IPsec, SSL, TLS, DTLS operation. The packet engine automatically increments the count if header processing is selected. Upon successful completion, the host reads back this value and writes it to the SA record.

Inbound: The host writes the counter value stored in the SA record to this register to start an IPsec or DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the expected count value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Note: The description is only for the direct host mode. The sequence number for autonomous ring mode and target command mode are updated by the packet engine.

The reset value of this register is zero.

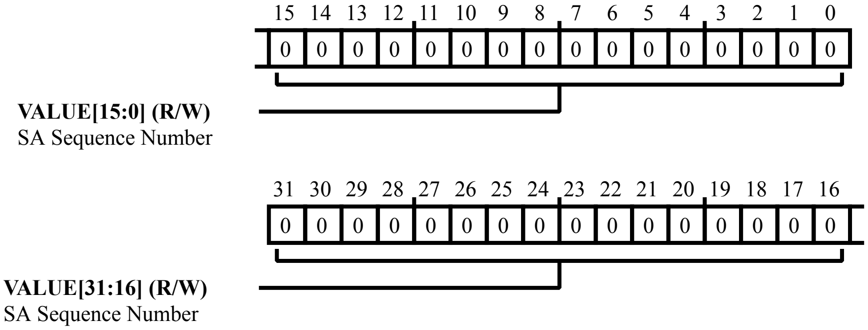


Figure 44-49: PKTE_SA_SEQNUM[n] Register Diagram

Table 44-75: PKTE_SA_SEQNUM[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA Sequence Number Mask Registers

The `PKTE_SA_SEQNUM_MSK[n]` registers are a set of two read/write registers and are used for IPsec ESP and DTLS operations to specify the anti-replay sequence number mask value for inbound operations. The packet engine manages this counter value automatically.

Inbound: The host writes the counter value stored in the SA record into this register upon starting an IPsec, DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the new mask value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Outbound: not used.

Note that the above description only applies to the direct host mode, for autonomous ring mode and target command mode the packet engine extracts the sequence number mask from the SA record.

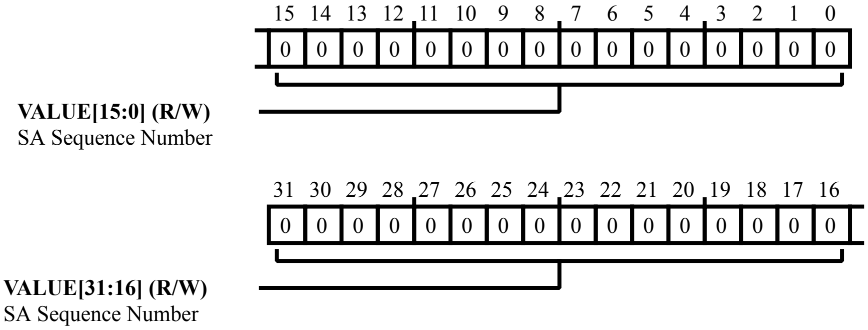

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
SA Sequence Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
SA Sequence Number

Figure 44-50: PKTE_SA_SEQNUM_MSK[n] Register Diagram

Table 44-76: PKTE_SA_SEQNUM_MSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA SPI Register

For IPsec operations, the `PKTE_SA_SPI` register is written with the SPI (Security Parameters Index) associated with the inbound or outbound flow.

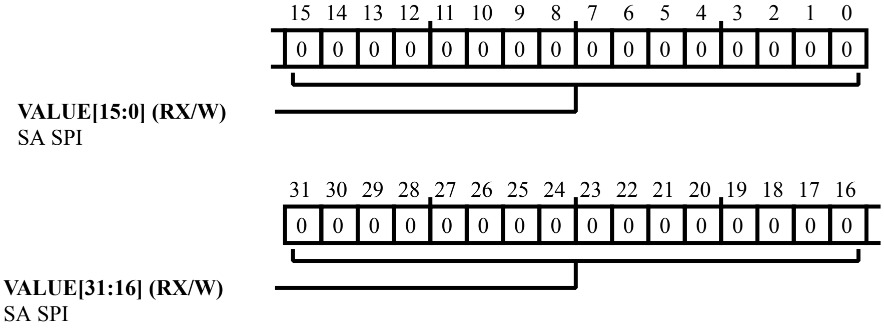

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (RX/W)
SA SPI
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (RX/W)
SA SPI

Figure 44-51: PKTE_SA_SPI Register Diagram

Table 44-77: PKTE_SA_SPI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	SA SPI. The `PKTE_SA_SPI.VALUE` bit field is used for IPsec ESP operations to specify the Security. Parameters Index (SPI) value that is to be placed in the ESP header. There is no need to read back this value at the end of an operation, since the Packet Engine does not change it. For SSL, TLS and DTLS this register stores the 8-bit TYPE field in bits [23:16], and the 16-bit Version field in bits [15:0] that are part of the protocol header.

Packet Engine Source Address

The `PKTE_SRC_ADDR` register holds the starting (byte) address for the packet to be processed.

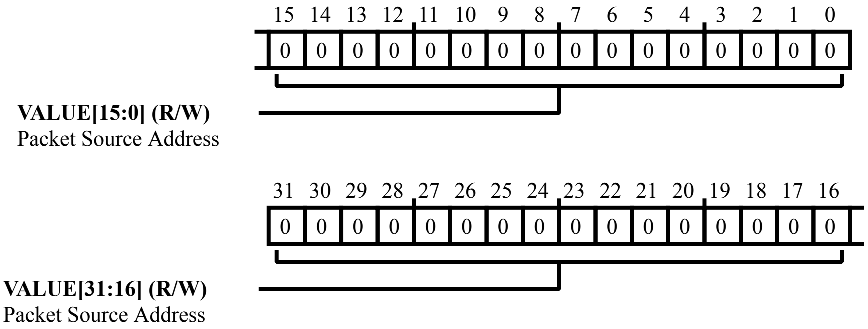

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Packet Source Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Packet Source Address

Figure 44-52: PKTE_SRC_ADDR Register Diagram

Table 44-78: PKTE_SRC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Source Address. The `PKTE_SRC_ADDR.VALUE` bit field holds the starting (byte) address for the packet to be processed.

Packet Engine Status Register

The PKTE_STAT register is used to provide the status of the packet engine. This register is useful in the direct host mode to determine when data must be written to or read from the packet engine, or for debugging the software when errors occur. This register can be ignored in autonomous ring mode and target command mode where the DMA engine controls the packet data I/O. This is a read-only register. A write to any of the bits has no effect.

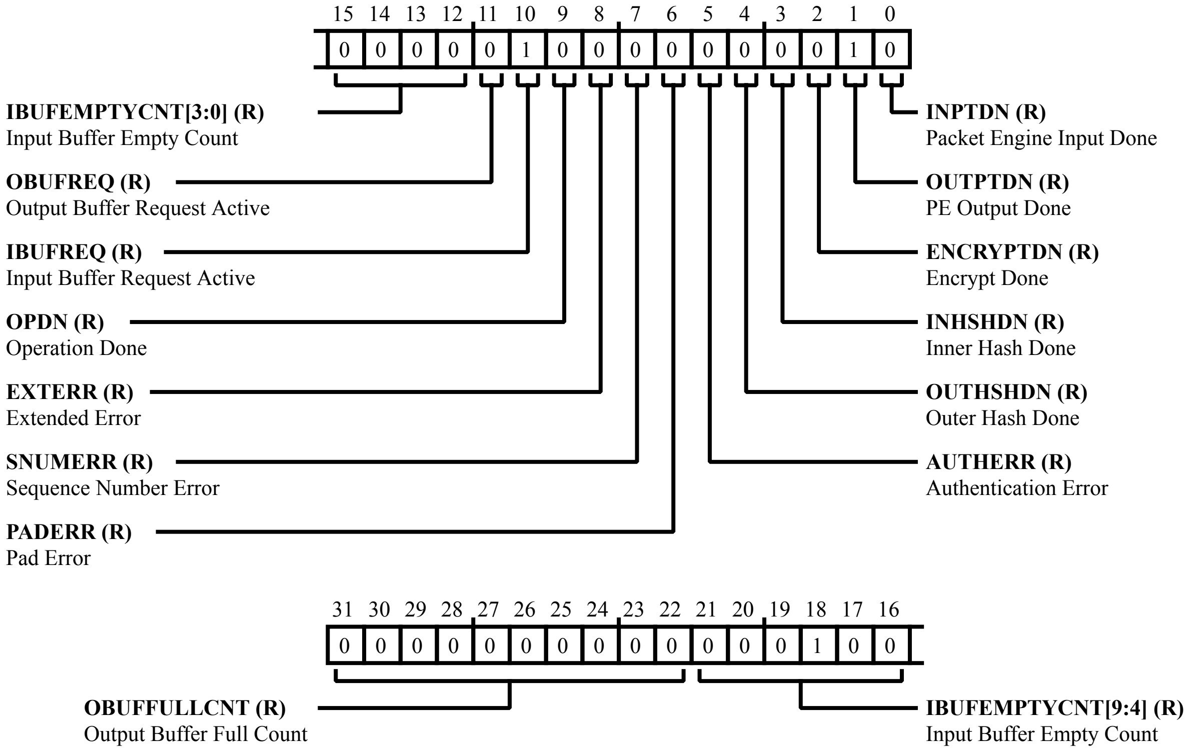

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
IBUFEMPTYCNT[3:0] (R)
Input Buffer Empty Count
OBUFREQ (R)
Output Buffer Request Active
IBUFREQ (R)
Input Buffer Request Active
OPDN (R)
Operation Done
EXTERR (R)
Extended Error
SNUMERR (R)
Sequence Number Error
PADERR (R)
Pad Error
INPTDN (R)
Packet Engine Input Done
OUTPTDN (R)
PE Output Done
ENCRYPTDN (R)
Encrypt Done
INHSHDN (R)
Inner Hash Done
OUTHSHDN (R)
Outer Hash Done
AUTHERR (R)
Authentication Error
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
OBUFFULLCNT (R)
Output Buffer Full Count
IBUFEMPTYCNT[9:4] (R)
Input Buffer Empty Count

Figure 44-53: PKTE_STAT Register Diagram

Table 44-79: PKTE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:22 (R/NW)	OBUFFULLCNT	Output Buffer Full Count. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of 32-bit words that are available in the packet engine output buffer. It works in conjunction with bit 11 from this register. When bit 11 is asserted, to indicate a request for output, the word count matches the specified output buffer threshold setting in the PKTE_BUF_THRESH register. For the last output for a given packet, any value from 1 dword to the full output buffer threshold can be seen. Transfers must be a multiple of full dwords. The application must read the PKTE_LEN field in the result descriptor to determine the exact byte-length of the result.

Table 44-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:12 (R/NW)	IBUFEMPTYCNT	Input Buffer Empty Count. The `PKTE_STAT.IBUFEMPTYCNT` bit field indicates the number of 32-bit empty spaces that are available in the packet engine input buffer. It works in conjunction with the `PKTE_STAT.IBUFREQ` bit (10) from this register. The value in the register is deducted from the specified packet length, so will never exceed the number of dwords that remain in the packet. For packets smaller than the buffer size, this register typically indicates that buffer space is available for the entire packet (rounded up to the nearest dword). For very large packets, these bits usually have a value around the maximum buffer size, indicating that the full input buffer is available.	
11 (R/NW)	OBUFREQ	Output Buffer Request Active. The `PKTE_STAT.OBUFREQ` bit indicates that the packet engine requests output data to be read from the output buffer.	
		0	No request for output data
		1	Request for output data
10 (R/NW)	IBUFREQ	Input Buffer Request Active. The `PKTE_STAT.IBUFREQ` bit indicates that the packet engine requests input data to be written to the input buffer.	
		0	No request for input data
		1	Request for input data
9 (R/NW)	OPDN	Operation Done. The `PKTE_STAT.OPDN` bit indicates that the packet engine has finished processing a packet when in direct host mode. This bit is zero in autonomous ring mode and target command mode.	
		0	Packet engine is idle
		1	Packet engine has finished processing a packet
8 (R/NW)	EXTERR	Extended Error. The `PKTE_STAT.EXTERR` bit indicates that an extended error occurred for this packet. For more information, refer to table Extended Error Codes - Status Encoding.	
		0	No extended error
		1	Extended error

Table 44-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SNUMERR	Sequence Number Error. For an inbound operation, the `PKTE_STAT.SNUMERR` bit indicates that there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition. For more information, refer to table Extended Error Codes - Status Encoding.	
		0	No sequence number error
		1	Input done, all bytes written to input buffer
6 (R/NW)	PADERR	Pad Error. The `PKTE_STAT.PADERR` bit indicates that an inbound crypto pad fault is detected. For more information about pad verification, refer to the Pad Verification and Consumption section.	
		0	No pad error
		1	Pad error
5 (R/NW)	AUTHERR	Authentication Error. The `PKTE_STAT.AUTHERR` bit indicates that an inbound ICV (for IPsec) or TAG (for SRTP) or MAC (for SSL/TLS/DTLS) fault is detected; the value carried within the packet did not match the value just computed.	
		0	No authentication error
		1	Authentication error
4 (R/NW)	OUTHSHDN	Outer Hash Done. The `PKTE_STAT.OUTHSHDN` bit indicates that the outer hash processing for this packet is finished.	
		0	Outer hash busy
		1	Outer hash done
3 (R/NW)	INHSHDN	Inner Hash Done. The `PKTE_STAT.INHSHDN` bit indicates that the inner hash processing for this packet is finished.	
		0	Inner hash busy
		1	Inner hash done
2 (R/NW)	ENCRYPTDN	Encrypt Done. The `PKTE_STAT.ENCRYPTDN` bit indicates that the encryption or decryption for this packet is finished.	
		0	Encryption or decryption busy
		1	Encryption or decryption done

Table 44-79: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	OUTPTDN	PE Output Done. The `PKTE_STAT.OUTPTDN` bit indicates that the output data for the current packet is read from the packet engine output buffer.	
		0	Output not done, more output bytes available
		1	Output done, all bytes read from the output buffer
0 (R/NW)	INPTDN	Packet Engine Input Done. The `PKTE_STAT.INPTDN` bit indicates that the number of bytes specified in the command descriptor `PKTE_LEN` field is written into the packet engine input buffer.	
		0	Input not done, more input bytes expected
		1	Input done, all bytes written to input buffer

Packet Engine State Record Address

The `PKTE_STATE_ADDR` register holds the start address of the SA state record.

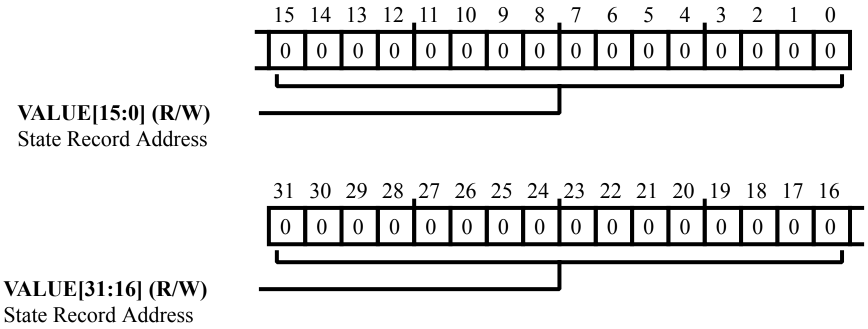

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
State Record Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
State Record Address

Figure 44-54: PKTE_STATE_ADDR Register Diagram

Table 44-80: PKTE_STATE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Record Address. The `PKTE_STATE_ADDR.VALUE` bit field holds the start address of the SA state record.

State Hash Byte Count Registers

The `PKTE_STATE_BYTE_CNT[n]` registers are used to enter a starting hash byte count, as well as to read the interim or final byte count.

For some hash operations, these registers are ignored and the byte count is internally set to 64 (0x40 hex) to indicate that the first 64 bytes (512 bits) hash block has been processed using a pre-computed hash state. These operations are:

All IPsec, SSL, TLS, DTLS and SRTP operations that use authentication; the "pre-computed" inner and outer hash digests are loaded from SA words 10 - 19.

Basic operations with `PKTE_SA_CMD0.HASHSRC` bits = 00 (from SA) specified. For Basic Hash with no HMAC, a pre-computed digest is loaded from SA words 10 - 14. For Basic Hash with HMAC, the inner and outer digests are loaded from SA words 10 - 19.

Note: Protocol operations can not be suspended in mid-packet and resumed later, therefore protocol operations do not use these registers.

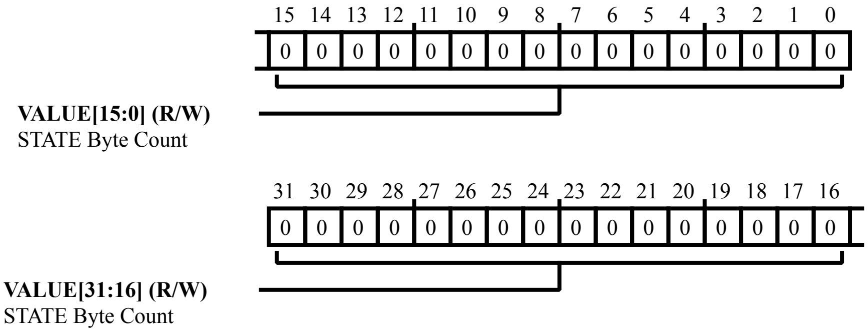

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
STATE Byte Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
STATE Byte Count

Figure 44-55: PKTE_STATE_BYTE_CNT[n] Register Diagram

Table 44-81: PKTE_STATE_BYTE_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	STATE Byte Count.

State Inner Digest Registers

The `PKTE_STATE_IDIGEST[n]` registers consist of eight 32-bit registers. These read/write registers are used to read the interim or final hash digest. The `PKTE_STATE_IDIGEST[n]` registers are only used with basic operations involving basic hash, and are typically used for operations that must be suspended and resumed in the middle of a hash. The interim hash state can be read from these registers along with the hash byte-count from the previous register. Both can be restored when resuming the hash. The appropriate save hash state (`PKTE_SA_CMD0.SVHASH`=1) and load hash from state (`PKTE_SA_CMD0.HASHSRC`=0b10) settings must be used. These registers are a mirror of the SA record inner hash digest register.

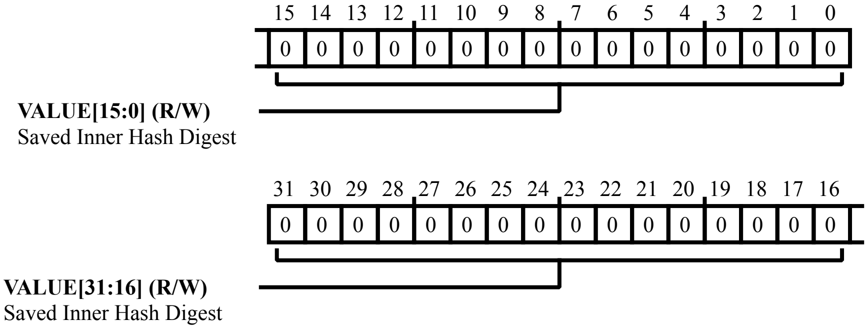

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Saved Inner Hash Digest
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Saved Inner Hash Digest

Figure 44-56: PKTE_STATE_IDIGEST[n] Register Diagram

Table 44-82: PKTE_STATE_IDIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Saved Inner Hash Digest.

State Initialization Vector Registers

The `PKTE_STATE_IV[n]` consists of four 32-bit registers. These registers are used to enter a starting IV state and to read the interim or final IV. PKTE_STATE_IV0 and PKTE_STATE_IV1 are used with DES/3DES cipher while PKTE_STATE_IV0 to PKTE_STATE_IV3 are used with AES cipher. The reset value of these registers is zero.

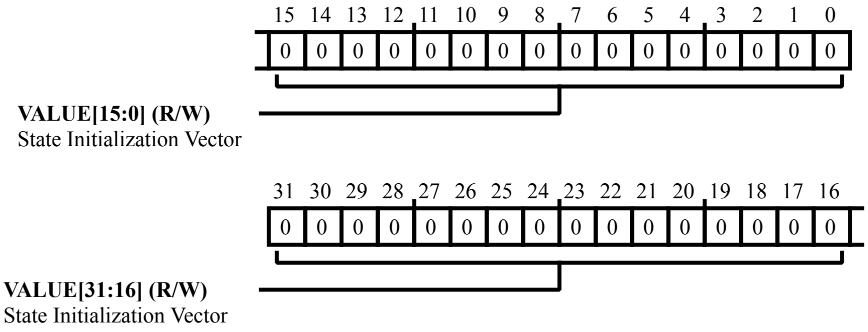

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
State Initialization Vector
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
State Initialization Vector

Figure 44-57: PKTE_STATE_IV[n] Register Diagram

Table 44-83: PKTE_STATE_IV[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Initialization Vector. The `PKTE_STATE_IV[n].VALUE` bit field is used to enter a starting IV state and to read the interim or final IV.

Packet Engine User ID

The PKTE_USERID register is a read/write register that gives identification to a command descriptor and the resultant result descriptor. The host is free to use this field for its own purpose. The host can write a unique identifier to the register in direct host mode or includes it as part of the command descriptor in autonomous ring mode. The PKTE_USERID register value passes though the packet engine without alteration to the result descriptor to be read back by the host.

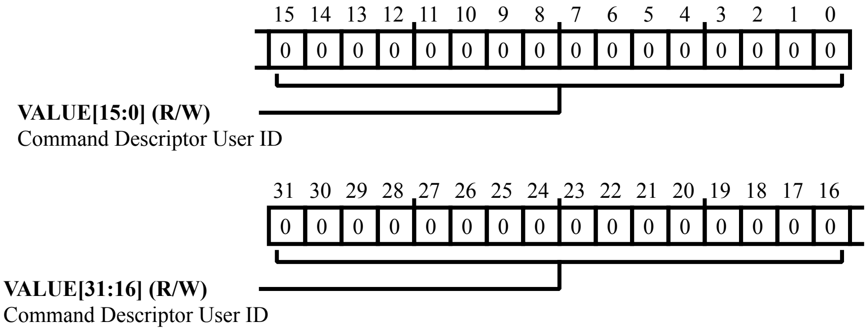

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Command Descriptor User ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Command Descriptor User ID

Figure 44-58: PKTE_USERID Register Diagram

Table 44-84: PKTE_USERID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor User ID. The PKTE_USERID.VALUE bit field gives identification to a command descriptor and the resultant result descriptor.

45 Public Key Accelerator (PKA)

The PKA helps offload computationally-intensive operations commonly found in public key cryptography algorithms.

PKA Features

The PKA engine provides the following basic operations:

- Large vector addition, subtraction, and combined addition/subtraction
- Large vector shift right or left
- Large vector multiplication, division (with and without quotient)
- Large vector compare and copy

The PKA engine provides the following complex operations:

- Large vector unsigned value modular exponentiation
- Large vector unsigned value modular exponentiation using the ‘Chinese Remainders Theorem’ (CRT) method with pre-calculated Q inverse vector
- Modular inversion: Given A and M, calculate B such that ((A × B) MOD M) = 1
- ECC point addition/doubling on elliptic curve $y^2 = x^3 + ax + b \pmod{p}$ with prime number *p* and input values *a* and *b* to the operation. Adding two identical points automatically performs point doubling.
- ECC point multiplication on elliptic curve $y^2 = x^3 + ax + b \pmod{p}$ with prime number *p* and input values *a* and *b* to the operation. A version of the ‘Montgomery ladder’ algorithm is used to provide side channel attack resistance.

The PKA also contains hardware logic to automatically zero out the PKA RAM buffer to clear out any information that is considered sensitive or secure.

PKA Functional Description

The following sections provide details on the function of the PKA module.

ADSP-SC58x PKA Register List

The Public Key Accelerator module (PKA) provides security-related features. A set of registers governs PKA operations. For more information on PKA functionality, see the PKA register descriptions.

Table 45-1: ADSP-SC58x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Definitions

The following definitions are helpful when using the PKA module.

Elliptic Curve Cryptography (ECC)

A form of public key cryptography based on elliptic curves over finite fields.

RSA

An acronym for Ron Rivest, Adi Shamir, and Leonard Adleman. It is another form of a public key cryptosystem.

Chinese Remainder Theorem (CRT)

A mathematical theorem used for simplifying time-consuming arithmetic used in public key algorithm computations.

Addition Chaining Table (ACT)

A method of speeding up exponentiation by repeatedly squaring the input and storing the result and reusing the result as input. ACT2 uses a table with 2 address bits (4 entries) and ACT4 uses a table with 4 address bits (16 entries).

PKA Architectural Concepts

The following sections describe the PKA architecture.

Public Key Co-Processor (PKCP)

The Public Key Co-Processor (PKCP) handles the basic large vector processing such as addition, subtraction, multiplication, etc.

Sequencer

The sequencer is small processor that is part of the PKA which handles the more complicated vector processing for public key algorithms. Algorithms include modular exponentiation and the ECC addition and ECC multiply used in Elliptic Curve Cipher algorithms. It executes instructions stored from an internal pre-programmed ROM that handles these operations.

RAM

Input and output vectors are stored in a 4 kB RAM buffer that is part of the MMR space. The address of PKA_RAM is the beginning of the RAM space. This memory is also used as a scratchpad or workspace for the sequencer and PKCP. Programs must place the vectors appropriately following the constraints described in the ***Functional Description*** section of this chapter.

PKA Block Diagram

The ***PKA Block Diagram*** shows the top-level block diagram of the PKA engine. The PKA engine is comprised of five parts:

1. Registers for input, output, status, and control
2. Public Key Co-processor (PKCP) module which performs the basic suite of big number (vector) operations typically found in public key cryptography applications
3. Sequencer which controls modular exponentiation, elliptic curve cryptography, and modular inversion operations.
4. Program ROM associated with the PKA engine exclusively for the sequencer
5. PKA RAM holds the large input and output values as well as the workspace/scratchpad required from the sequencer and PKCP for operations.

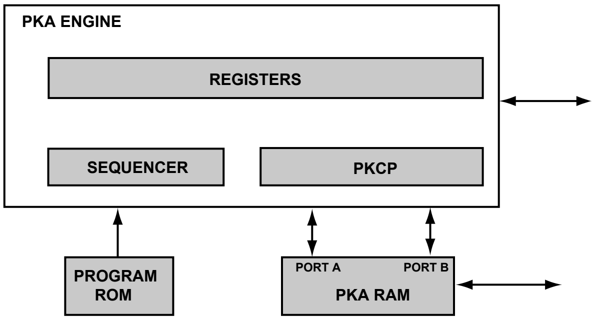


Figure 45-1: PKA Block Diagram

PKCP Vector Operations

The ***Summary of PKCP Vector Operations*** table lists the arguments and results for each PKCP vector operation.

Table 45-2: Summary of PKCP Vector Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
Multiply	A x B → C	Multiplicand	Multiplier	Product	N/A
Add	A + B → C	Addend	Addend	Sum	N/A
Subtract	A - B → C	Minuend	Subtracthend	Difference	N/A
AddSub	A + C - B → D	Addend	Subtracthend	Addend	Result
Right Shift	A >> Shift → C	Input	N/A	Result	N/A
Left Shift	A << Shift → C	Input	N/A	Result	N/A
Divide	A mod B → C, A div B → D	Dividend	Divisor	Remainder	Quotient
Modulo	A mod B → C	Dividend	Divisor	Remainder	N/A
Compare	A = B, A < B, A > B	Input 1	Input 2	N/A	N/A
Copy	A → C	Input	N/A	Result	N/A

To obtain correct result, the input vectors must meet the requirements presented in the ***Operational Restrictions on Input Vectors for PKCP Operations*** table.

Note the following:

- The PKCP does not check input restrictions
- A_Len and B_Len indicate the size of vectors A and B in (32-bit) words
- Max_Len equals 128 (32-bit) words, for example, the standard maximum vector size is 4096 bit

Table 45-3: Operational Restrictions on Input Vectors for PKCP Operations

Function	Requirement
Multiply	0 < A_Len, B_Len <= Max_Len
Add	0 < A_Len, B_Len <= Max_Len
Subtract	0 < A_Len, B_Len <= Max_Len Result must be positive (A>=B)
AddSub	0 < A_Len <= Max_Len (B and C operands have A_Len as length, B_Len ignored) Result must be positive ((A + C) >= B)
Right Shift	0 < A_Len <= Max_Len
Left Shift	0 < A_Len <= Max_Len
Divide, Modulo	1 < B_Len <= A_Len <= Max_Len Most significant 32-bit word of B operand cannot be zero
Compare	0 < A_Len <= Max_Len (B operand has A_Len as length, B_Len ignored)
Copy	0 < A_Len <= Max_Len

The host processor is responsible for allocating a block of contiguous memory in PKA RAM for the result vectors. The ***PKCP Result Vector Memory Allocation*** table indicates how much memory is allocated for the result vectors.

Table 45-4: PKCP Result Vector Memory Allocation

Function	Result Vector	Result Vector Length (in 32-bit words)
Multiply	C	A_Len + B_Len + 6 (the 6 ‘scratchpad’ words should be discarded)
Add	C	Max(A_Len, B_Len) + 1
Subtract	C	Max(A_Len, B_Len)
AddSub	D	A_Len + 1
Right Shift	C	A_Len
Left Shift	C	A_Len + 1 (when Shift Value is non-zero) A_Len (when Shift Value is zero)
Divide	C	Remainder → B_Len + 1 (one ‘scratchpad’ word should be discarded)
	D	Quotient → A_Len – B_Len + 1
Modulo	C	Remainder → B_Len + 1 (one ‘scratchpad’ word should be discarded)
Compare	None	Compare updates the PKA_COMPARE register
Copy	C	A_Len

Input vectors for an operation are always allowed to overlap in memory (partially or completely). The ***PKCP Result Vector/Input Overlap Restrictions*** table gives restrictions for the overlap of output and input vectors of the operations.

Table 45-5: PKCP Result Vector/Input Overlap Restrictions

Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed
Add, Subtract	C	May overlap with A and/or B vector, provided the start address of the C vector does not lie above the start address of the vectors with which it overlaps
AddSub	D	May overlap with A, B and/or C vector, provided the start address of the D vector does not lie above the start address of the vectors with which it overlaps
Right Shift, Left Shift	C	May overlap with A vector, provided the start address of the C vector does not lie above the start address of the A vector
Divide	C	No overlap with A, B, or D vectors allowed
	D	No overlap with A, B, or C vectors allowed
Modulo	C	No overlap with A or B vectors allowed
Compare	None	Compare does not write a result vector
Copy	None	Same restrictions as for right or left shift, copy of a vector to a lower address is always allowed even if source and destination overlap†

†The copy operation can be used to fill memory by breaking the overlap restrictions, but it requires setting up TWO initial (32-bit) words. To zero a block of memory, set the A vector pointer to the block start, set the C vector pointer two words higher and the A vector length to the block length minus two (words). Fill the first two words of the block with constant zero and perform a PKCP copy operation to zero the remainder of the block.

Modular Exponentiation Operations

The ***Summary of ExpMod Operations*** table summarizes the modular exponentiation operations that the PKA supports.

Table 45-6: Summary of ExpMod Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$C^A \bmod B \rightarrow D$	Exponent, length = A_Len	Modulus, length = B_Len	Base, length = B_Len	Result and Workspace
ExpMod-CRT	See below	Exp P followed by Exp Q at next higher even word address†, both A_Len long	Mod P + buffer word followed by Mod Q at next higher even word address‡, both B_Len long	Q inverse, length = B_Len	Input, Result (both 2xB-Len long) and Workspace

† If A_Len is even, Exp Q follows Exp P immediately – if A_Len is odd, there is one empty word between Exp Q and Exp P.

‡ If B_Len is even, there are two empty words between Mod P and Mod Q – if B_Len is odd, there is one empty (buffer) word between Mod Q and Mod P. Note that the engine may zero the words following Mod P and Mod Q.

The ExpMod-CRT operation performs the following computation steps. (These steps implement Garner's recombination algorithm after the basic exponentiations.)

- X <- (Input mod Mod P)Exp P mod Mod P
- Y <- (Input mod Mod Q)Exp Q mod Mod Q
- Z <- ((((X – Y) mod Mod P) · Q inverse) mod Mod P) · Mod Q
- Result <- Y + Z

The ExpMod-ACT2, -ACT4, and -variable functions implement the same mathematical operation but with a differently sized table with pre-calculated *odd powers*. The ExpMod-ACT2 function uses a table with two entries whereas ExpMod-ACT4 uses a table with eight entries. The ACT4 version gives better performance but needs more memory. ExpMod-variable and ExpMod-CRT operations allow the selection of a variable number (from 1 up to and including 16) of odd powers through the register normally used to specify the number of bits to shift for shift operations.

The exponentiation functions are extensions of the set of PKA functions. Input and result vectors are passed the same way as basic PKCP operations. The ***Restrictions on Input Vectors for ExpMod Operations*** table shows the restrictions on the input and result vectors for the exponentiation operations.

Table 45-7: Restrictions on Input Vectors for ExpMod Operations

Function	Requirements
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	0 < A_Len <= Max_Len 1 < B_Len <= Max_Len Modulus B must be odd (for example, the least significant bit must be ONE) Modulus B > 2^{32} Base C < Modulus B Vectors B and C must be followed by an empty 32-bit *buffer* word
ExpMod-CRT	0 < A_Len <= Max_Len 1 < B_Len <= Max_Len Mod P and Mod Q must be odd (for example, the least significant bits must be ONE) Mod P > Mod Q > 2^{32}(note that Mod P must be larger than Mod Q) Mod P and Mod Q must be co-prime (their GCD must be 1) 0 < Exp P < (Mod P – 1) 0 < Exp Q < (Mod Q – 1) (Q inverse · Mod Q) 1 (modulo Mod P) Input < (Mod P · Mod Q) Mod P and Mod Q must be followed by an empty 32-bit *buffer* word

The ***ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR*** table shows the required scratchpad sizes for the exponentiation operations. These sizes depend on the PKA type. The 'M_Len' used in the table is the 'real' Modulus length in 32-bit words, for example, without trailing zero words at the end. (This description also applies to Mod P in an ExpMod-CRT operation and Modulus B in the other operations.) If the last word of the modulus vector as given is non-zero, 'M_Len' equals B_Len.

Table 45-8: ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR

Function	Scratchpad Area Size (in 32-bit words), Result Vector is either M_Len or 2xM_Len 32-bit words long
ExpMod-ACT2	5 x (M_Len + 2)
ExpMod-ACT4	11 x (M_Len + 2)
ExpMod-variable	(# odd powers + 3) x (M_Len + 2)
ExpMod-CRT	(# odd powers + 3) x (M_Len + 2) + (M_Len + 2 – (M_Len MOD 2))

NOTE: During execution of an ExpMod-ACT2, -ACT4 or -variable operation, the last 34 bytes of the PKA RAM are used as the general scratchpad for the sequencer program execution. The ExpMod-CRT operation requires the last 72 bytes of the PKA RAM as the scratchpad. These (fixed location) areas may not overlap with any of the input vectors and/or the D vector scratchpad area. They can be used freely when executing basic PKCP operations.

Table 45-9: ExpMod Scratchpad Area / Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	D	Scratchpad area starting at D may not overlap with any of the other vectors, except that base C may be co-located with result vector D to save space (for example,`PKA_CPTR`= `PKA_DPTR` is allowed).
ExpMod-CRT	D	Scratchpad area starting at D may not overlap with any of the other vectors. This area is also the location of the main input vector (with length 2 x B_Len)

The ***Maximum Number of Odd Powers*** table indicates the maximum number of odd powers that can be used for different standard PKA RAM sizes and PKA types (non-CRT operations using `PKA_CPTR` = `PKA_DPTR`). As a rule of thumb, for optimal performance, use one odd power for *Verify* operations and 4 (or as many as the implemented PKA RAM size allows) for *Sign* operations. Note the following points about odd powers:

- Using more than eight odd powers is not advisable as the speed advantage for each extra odd power decreases rapidly (and can even become negative for short exponent vector lengths due to the extra pre-processing required).
- The maximum number of odd powers is 16 (limited by the firmware). All '16 odd powers' entries in the table above hit this limit – they are not limited by the PKA RAM size.

Table 45-10: Maximum Number of Odd Powers

Operation	Modulus and Exponent Sizes	Maximum Number of Odd Powers
Non-CRT	1024 bits	16
	2048 bits	10
	4096 bits	2
CRT	2 × 512 bits	16
	2 ×1024 bits	16
	2 × 2048 bits	6

The ***2K-bit Modular Exponentiation PKA RAM Allocation Examples*** table shows example PKA RAM vector allocations for modular exponentiation operations with and without using CRT. The *free space* start address is the first free byte following the vector workspace. The sequencer execution scratchpad of 34 bytes (non-CRT) or 72 bytes (using CRT) must fit between this address and the end of the PKA RAM. Note that the non-CRT operations use `PKA_CPTR` = `PKA_DPTR` to save space.

Table 45-11: 2K-bit Modular Exponentiation PKA RAM Allocation Examples

Operation	(sub-)vector	Start address (Byte Offset)	Size (words)	Buffer (words)
non-CRT (`PKA_ALEN` = 0x040, `PKA_BLEN` = 0x040, 4 odd-powers)	Exponent	0x000 (`PKA_APTR` = 0x000)	64	0
	Modulus	0x100 (`PKA_BPTR` = 0x040)	64	2
	Base	0x208 (`PKA_CPTR` = 0x082)	64	2
	Result	0x208 (`PKA_DPTR` = 0x082)	64	2
	Vector Workspace	0x208 (= Result)	7 × (64+2)=462	0
	Free space	0x940 (2368 bytes used)	-	-
using CRT (`PKA_ALEN` = 0x020, `PKA_BLEN` = 0x020, 4 odd-powers)	Exp P	0x000 (`PKA_APTR` = 0x000)	32	0
	Exp Q	0x080	32	0
	Mod P	0x100 (`PKA_BPTR` = 0x040)	32	2
	Mod Q	0x188	32	2
	Q inverse	0x210 (`PKA_CPTR` = 0x084)	32	0
	Input, Result	0x290 (`PKA_DPTR` = 0x0A4)	64	0
	Vector workspace	0x290 (= Result)	7 × (32+2)+32+2-0 = 272	0
	Free space	0x6D0 (1744 bytes used)	-	-

The following example in pseudo-code describes the execution of a non-CRT modular exponentiation operation using a 512 bits modulus and a 160 bits exponent, using actual test vectors:

```
// Perform a 512/160 bits
modular exponentiation without CRT (using 4 ‘odd-powers’)
// Exponent equals value 0x8FD84098_8A0930CC_9CDC1E8A_B246EB46_2D39F064
// write as vector A to PKA
```

```
RAM Byte offset 0x000:
Write PKA_RAM_BASE+0x000+0x00
0x2D39F064
Write PKA_RAM_BASE+0x000+0x04
0xB246EB46
Write PKA_RAM_BASE+0x000+0x08
0x9CDC1E8A
Write PKA_RAM_BASE+0x000+0x0C
0x8A0930CC
Write PKA_RAM_BASE+0x000+0x10
0x8FD84098
// Modulus equals value 0xF42F559D
1877CA5F_449492B9_42DC7C01_...
// A3C9085B_7236A085_2102B000_A093C6B4_...
// 9D0EDA0C_292DE841_29C23723_4048BDA3_...
// 373C4C9F_45CF15A7_5F049ABF_D8A01B9B
// write as vector B to PKA
RAM Byte offset 0x018 (following exp at next aligned 64-bit word):
Write PKA_RAM_BASE+0x018+0x00
0xD8A01B9B
Write PKA_RAM_BASE+0x018+0x04
0x5F049ABF
Write PKA_RAM_BASE+0x018+0x08
0x45CF15A7
Write PKA_RAM_BASE+0x018+0x0C
0x373C4C9F
Write PKA_RAM_BASE+0x018+0x10
0x4048BDA3
Write PKA_RAM_BASE+0x018+0x14
0x29C23723
Write PKA_RAM_BASE+0x018+0x18
0x292DE841
Write PKA_RAM_BASE+0x018+0x1C
0x9D0EDA0C
Write PKA_RAM_BASE+0x018+0x20
0xA093C6B4
Write PKA_RAM_BASE+0x018+0x24
0x2102B000
Write PKA_RAM_BASE+0x018+0x28
0x7236A085
Write PKA_RAM_BASE+0x018+0x2C
0xA3C9085B
Write PKA_RAM_BASE+0x018+0x30
0x42DC7C01
Write PKA_RAM_BASE+0x018+0x34
0x449492B9
Write PKA_RAM_BASE+0x018+0x38
0x1877CA5F
Write PKA_RAM_BASE+0x018+0x3C
0xF42F559D
```

```
// Base equals value 0x3D291F48_49064887_1149594B_67935110_...
// 14EB8FF0_AB291F3A_54A1B4D1_5E611E44_...
// C989251B_44904B45_0B060482_317F8352_...
// 18CE440E_9BF509F1_6EAF26F2_95F19F12
// write as vector C to PKA
RAM Byte offset 0x060 (following mod after buffer + align words):
Write PKA_RAM_BASE+0x060+0x00
0x95F19F12
Write PKA_RAM_BASE+0x060+0x04
0x6EAF26F2
Write PKA_RAM_BASE+0x060+0x08
0x9BF509F1
Write PKA_RAM_BASE+0x060+0x0C
0x18CE440E
Write PKA_RAM_BASE+0x060+0x10
0x317F8352
Write PKA_RAM_BASE+0x060+0x14
0x0B060482
Write PKA_RAM_BASE+0x060+0x18
0x44904B45
Write PKA_RAM_BASE+0x060+0x1C
0xC989251B
Write PKA_RAM_BASE+0x060+0x20
0x5E611E44
Write PKA_RAM_BASE+0x060+0x24
0x54A1B4D1
Write PKA_RAM_BASE+0x060+0x28
0xAB291F3A
Write PKA_RAM_BASE+0x060+0x2C
0x14EB8FF0
Write PKA_RAM_BASE+0x060+0x30
0x67935110
Write PKA_RAM_BASE+0x060+0x34
0x1149594B
Write PKA_RAM_BASE+0x060+0x38
0x49064887
Write PKA_RAM_BASE+0x060+0x3C
0x3D291F48
// The result value and scratchpad
(vector D) may be co-located with the base vector C for
// a normal modular exponentiation,
so these are located at PKA RAM Byte offset 0x060 too.
// Load pointer and length
registers:
Write PKA_APTR 0x000>>2 //
Exponent pointer
Write PKA_BPTR 0x018>>2 //
Modulus pointer
Write PKA_CPTR 0x060>>2 //
Base pointer
```

```
Write PKA_DPTR 0x060>>2 //
Result/scratchpad pointer
Write PKA_ALENGTH 0x00000005
// Exponent length in 32-bit words
Write PKA_BLENGTH 0x00000010
// Mod/base/result length in 32-bit words
// Start modular exponentiation
and wait until it’s done:
Write PKA_SHIFT 0x00000004
// Number of ‘odd powers’
Write PKA_FUNCTION 0x0000E000
// ‘Run’ bit set, ‘Sequencer Operations’ = 0b110
Wait PKA_FUNCTION[15] == ‘0’
// ‘Run’ bit clears itself - Host can also use interrupt!
// Result value equals 0xA497BF8B_DB729088_954005B0_B5CA6691_...
// A3EC491B_091A3D62_03C24214_0863A389_...
// 0C7C03CD_2333E231_35EC10ED_8F91281C_...
// 30F4253B_FE38FAFB_BB4A39DB_C14F2661
// written as vector D at
PKA RAM Byte offset 0x060:
Check PKA_RAM_BASE+0x060+0x00
== 0xC14F2661
Check PKA_RAM_BASE+0x060+0x04
== 0xBB4A39DB
Check PKA_RAM_BASE+0x060+0x08
== 0xFE38FAFB
Check PKA_RAM_BASE+0x060+0x0C
== 0x30F4253B
Check PKA_RAM_BASE+0x060+0x10
== 0x8F91281C
Check PKA_RAM_BASE+0x060+0x14
== 0x35EC10ED
Check PKA_RAM_BASE+0x060+0x18
== 0x2333E231
Check PKA_RAM_BASE+0x060+0x1C
== 0x0C7C03CD
Check PKA_RAM_BASE+0x060+0x20
== 0x0863A389
Check PKA_RAM_BASE+0x060+0x24
== 0x03C24214
Check PKA_RAM_BASE+0x060+0x28
== 0x091A3D62
Check PKA_RAM_BASE+0x060+0x2C
== 0xA3EC491B
Check PKA_RAM_BASE+0x060+0x30
== 0xB5CA6691
Check PKA_RAM_BASE+0x060+0x34
== 0x954005B0
Check PKA_RAM_BASE+0x060+0x38
== 0xDB729088
```

```
Check PKA_RAM_BASE+0x060+0x3C
== 0xA497BF8B
```

Modular Inversion

Besides modular exponentiation, the sequencer also controls modular inversion operations.

Table 45-12: Summary of ModInv Operation

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ModInv	A^{-1} mod B → D	NumToInvert, length = A_Len	Modulus, length = B_Len	Not Used	Result and Workspace

The above function appears to be an extension of the set of basic PKCP functions with the following exceptions:

- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register field is used to return info on the operation's result.

Table 45-13: PKA_SHIFT Result Values for ModInv Operation

Function	PKA_SHIFT Register Field Value At Conclusion
ModInv	0 → success; VectorD holds result 7 → no inverse exists (GCD(A, B) != 1, for example, A and B have common factors); result undefined 31 → error, modulus even; result undefined other values are reserved

The following tables list the restrictions on the input and result vectors for the ModInv operation:

Table 45-14: Operational Restrictions on Input Vectors for the ModInv Operation

Function	Requirements
ModInv	0 < A_Len <= Max_Len 0 < B_Len <= Max_Len Modulus B must be odd (for example, the least significant bit must be ONE) Modulus B may not have value 1 (result is undefined, no error indicated) The highest word of the modulus vector, as indicated by B_Len, may not be zero.

Table 45-15: ModInv Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ModInv	D	Scratchpad area starting at D may not overlap with any of the other vectors

The following table shows the required scratchpad sizes for the ModInv Operation:

Table 45-16: ModInv Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	Scratchpad area size (in 32-bit words), Result Vector is B_Length 32-bit words long
ModInv	5 x (M + ε(M)), with M = Max(A_Length, B_Length) ε(n) = 2 + (n MOD 2), for example, 2 (for n even) or 3 (for n odd)

NOTE: During execution of a ModInv operation, the last 34 bytes of the PKA RAM are used as general scratchpad for the sequencer program execution. This (fixed location) area may not overlap with any of the input vectors and/or the D vector scratchpad area during execution.

Modular Inversion with an Even Modulus

The ModInv operation requires the modulus to be odd. At first, this requirement appears to make the operation useless in the case of RSA key generation where the private key exponent *d* is derived from a chosen public exponent *e* as follows:

d = ModInv(e, φ); where φ = (p-1) × (q-1) and *p* and *q* both prime

Note that φ is even. However, since *e* must be odd (otherwise no inverse exists), *d* can be calculated as:

d = (1 + (φ × (e – ModInv(φ, e))) / e

With four more basic PKCP operations, ModInv can also be used to find inverse values in case the modulus is even.

Modular Inversion with a Prime Modulus

Modular inversion can be performed with a modular exponentiation using the modulus value minus two as exponent, provided that the modulus value is a prime. This is due to the following:

$(A^{M}) \bmod M = A \Rightarrow$

$(A^{M-1}) \bmod M = 1 \Rightarrow$

$(A^{M-2}) \bmod M = A{-}1 \pmod M$

Under the constraint that M is a prime value.

Especially with the large PKA engines containing an LNME, it is worthwhile to check whether this method is faster than using the ModInv operation directly. The modulus values for the ECC curves supported by this PKA engine must be prime, so this method can be used in ECDSA operations.

ECC Operations

Besides modular exponentiation and modular inversion, the sequencer also controls ECC operations.

Table 45-17: Summary of ECC Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ECC-ADD	Point addition/ doubling† on elliptic curve: $y^2 = x^3 + ax + b \pmod{p}$ pntA + pntC → pntD	pntA.x followed‡ by pntA.y both B_Len long (A_Len *not* used)	Curve parameter *p* followed‡ by *a* (*b* is not needed) all B_Len long	pntC.x followed by pntC.y both B_Len long	Result, for example, pntD.x followed‡ by pntD.y and workspace
ECC-MUL	Point multiplication on elliptic curve: $y^2 = x^3 + ax + b \pmod{p}$ *k* x pntC → pntD	Scalar *k* A_Len long	Curve parameter *p* followed‡ by *a*and *b* all B_Len long.	pntC.x followed‡ by pntC.y both B_Len long	Result, for example pntD.x followed‡ by pntD.y and workspace

† If pntA = pntC, a point doubling operation is performed automatically.

‡ All input components must be located on a 64-bit boundary and must have extra 'buffer' words (of 32 bits each) after their most significant word. ε must be 3 (B_Len odd) or 2 (B_Len even). Each result component (for example, pntD.x, pntD.y) is followed by ε buffer (zero) words.

The above functions appear to be extensions of the set of PKCP basic functions with the following exceptions:

- Input and result vectors can now be composite (for example, consist of two or three equal-sized subvectors)
- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register is used to return info on the result of the operation

Table 45-18: PKA_SHIFT Result Values for ECC Operations

Function	PKA_SHIFT Register Field Value at Conclusion
ECC-ADD ECC-MUL	0 → success; VectorD holds result point 7 → result is point-at-infinity; VectorD result point undefined 31 → error, (p not odd, p too short, etc); VectorD result point undefined Other values are reserved

The following tables below list the restrictions on the input and result vectors for the ECC operations.

Table 45-19: Operational Restrictions on Input Vectors for ECC Operations

Function	Requirements
ECC-ADD	1 < B_Len <= 24 (maximum vector length is 768 bits) Modulus p must be a prime > 2^{63} Effective modulus size (in bits) must be a multiple of 32† The highest word of the modulus vector, as indicated by B_Len, may not be zero. a< p and b< p pntA and pntC must be on the curve (this condition is not checked) Neither pntA nor pntC can be the point-at-infinity, although ECC-ADD can return this point as a result
ECC-MUL	0 < A_Len <= 24 (maximum vector length is 768 bits) 1 < B_Len <= 24 (maximum vector length is 768 bits) Modulus p must be a prime > 2^{63} Effective modulus size (in bits) must be a multiple of 32† The highest word of the modulus vector, as indicated by B_Len, may not be zero. a< p and b< p pntC must be on the curve (this condition is not checked) pntC cannot be the point-at-infinity, although ECC-MUL can return this point as a result

† Modulus lengths of 112 and 521 bits are exceptions to this rule.

Table 45-20: ECC Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ECC-ADD ECC-MUL	D	Scratchpad area starting at D may not overlap with any of the other vectors

The >***ECC Result Vector/Scratchpad Area Memory Allocation*** table shows the required scratchpad sizes for the ECC operations:

Table 45-21: ECC Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	Scratchpad area size (in 32-bit words), Result Vector is 2x(B_Length+ ε †(B_Length)) 32-bit words long
ECC-ADD	2 × L + 5 x M, where L = B_Length + ε(B_Length) M = B_Length + 1 + ε(B_Length + 1)
ECC-MUL	18 x L + Max(8, L), where L = (B_Length + ε(B_Length))

† $\varepsilon(n) = 2 + (n \text{ MOD } 2)$, for example, 2 (for n even) or 3 (for n odd)

NOTE: During execution of an ECC-ADD or ECC-MUL operation, the last 72 bytes of the PKA RAM are used as a general scratchpad for the sequencer program execution. These (fixed location) areas must not overlap with any of the input vectors and the D vector scratchpad area during execution.

The ***ECC Point Multiplication PKA RAM Allocation Examples*** table shows example PKA RAM vector allocations for ECC point multiplication operations. The *free space* start address is the first free byte following the vector scratchpad. The sequencer execution scratchpad of 72 bytes must fit between this address and the end of the PKA RAM. Because of this requirement, a 521-bit ECC point multiplication cannot be performed with 2K byte PKA RAM.

Table 45-22: ECC Point Multiplication PKA RAM Allocation Examples

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
192 bits (=6 words, `PKA_ALEN`=0x006, `PKA_BLEN`=0x006)	Scalar *k*	0x000 (`PKA_APTR` = 0x000)	6	0
	p	0x018 (`PKA_BPTR` = 0x006)	6	2
	a	0x038	6	2
	b	0x058	6	2
	PntC.x (base)	0x078 (`PKA_CPTR` = 0x01E)	6	2
	PntC.y (base)	0x098	6	2
	PntD.x (result)	0x0B8 (`PKA_DPTR` = 0x02E)	6	2
	PntD.y (result)	0x0D8	6	0
	Vector scratchpad	0x0B8 (= PntD.x)	(18 × 8) + 8 = 152	0
	Free space	0x318 (792 bytes used)	-	-
384 bits (=12 words, ALENGTH=0x00C, BLENGTH=0x00C)	Scalar *k*	0x000 (`PKA_APTR` = 0x000)	12	0
	p	0x030 (`PKA_BPTR` = 0x00C)	12	2
	a	0x068	12	2
	b	0x0A0	12	2
	PntC.x (base)	0x0D8 (`PKA_CPTR` = 0x036)	12	2
	PntC.y (base)	0x110	12	2
	PntD.x (result)	0x148 (`PKA_DPTR` = 0x052)	12	2
	PntD.y (result)	0x180	12	0
	Vector scratchpad	0x148 (= PntD.x)	(18 × 14) + 14 = 266	0
	Free space	0x570 (1392 bytes used)	-	-
521 bits (=17 words, ALENGTH=0x011, BLENGTH=0x011)	Scalar *k*	0x000 (`PKA_APTR` = 0x000)	17	1 (to align *p*)
	p	0x048 (`PKA_BPTR` = 0x012)	17	3
	a	0x098	17	3
	b	0x0E8	17	3
	PntC.x (base)	0x138 (`PKA_CPTR` = 0x04E)	17	3
	PntC.y (base)	0x188	17	3
	PntD.x (result)	0x1D8 (`PKA_DPTR` = 0x076)	17	3

Table 45-22: ECC Point Multiplication PKA RAM Allocation Examples (Continued)

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
	PntD.y (result)	0x228	17	0
	Vector scratchpad	0x1D8 (= PntD.x)	(18 × 20) + 20 = 380	0
	Free space	0x7C8 (1992 bytes used)	-	-

The following example in pseudo-code describes the execution of a 192 bits ECC point multiplication, using actual test vectors (the curve parameters and generator point are from standard curve ‘secp192r1’).

```
// Perform a 192 bits ECC point multiplication using PKA RAM layout from table
above.
// Scalar ‘k’ equals value 0x8D98D058_9EFD018A_C9BCF3CF_2C33AEC0_24867D7F_6ADACBFF
// write as vector A to PKA RAM Byte offset 0x000:

    Write PKA_RAM_BASE+0x000+0x00 0x6ADACBFF
    Write PKA_RAM_BASE+0x000+0x04 0x24867D7F
    Write PKA_RAM_BASE+0x000+0x08 0x2C33AEC0
    Write PKA_RAM_BASE+0x000+0x0C 0xC9BCF3CF
    Write PKA_RAM_BASE+0x000+0x10 0x9EFD018A
    Write PKA_RAM_BASE+0x000+0x14 0x8D98D058
// Curve parameter ‘p’ equals value
0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFF
// write as 1st part of vector B immediately following vector A at PKA RAM Byte
offset 0x018
// (no buffer word needed after ‘k’ vector, 64-bit alignment is OK):
    Write PKA_RAM_BASE+0x018+0x00 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x04 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x08 0xFFFFFFFE
    Write PKA_RAM_BASE+0x018+0x0C 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x10 0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x14 0xFFFFFFFF
// Curve parameter ‘a’ equals value
0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFC
// write as 2nd part of vector B after one buffer word and one re-alignment word
at 0x038:
    Write PKA_RAM_BASE+0x038+0x00 0xFFFFFFFC
    Write PKA_RAM_BASE+0x038+0x04 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x08 0xFFFFFFFE
    Write PKA_RAM_BASE+0x038+0x0C 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x10 0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x14 0xFFFFFFFF
// Curve parameter ‘b’ equals value
0x64210519_E59C80E7_0FA7E9AB_72243049_FEB8DEEC_C146B9B1
// write as 3rd part of vector B after one buffer word and one re-alignment word
at 0x058:
    Write PKA_RAM_BASE+0x058+0x00
0xC146B9B1
```

```
    Write PKA_RAM_BASE+0x058+0x04 0xFEB8DEEC
    Write PKA_RAM_BASE+0x058+0x08 0x72243049
    Write PKA_RAM_BASE+0x058+0x0C 0x0FA7E9AB
    Write PKA_RAM_BASE+0x058+0x10 0xE59C80E7
    Write PKA_RAM_BASE+0x058+0x14 0x64210519
// X-coord of generator point is value
0x188DA80E_B03090F6_7CBF20EB_43A18800_F4FF0AFD_82FF1012
// write as 1st part of vector C following vector B after buffer + alignment
words at 0x078:
    Write PKA_RAM_BASE+0x078+0x00 0x82FF1012
    Write PKA_RAM_BASE+0x078+0x04 0xF4FF0AFD
    Write PKA_RAM_BASE+0x078+0x08 0x43A18800
    Write PKA_RAM_BASE+0x078+0x0C 0x7CBF20EB
    Write PKA_RAM_BASE+0x078+0x10 0xB03090F6
    Write PKA_RAM_BASE+0x078+0x14 0x188DA80E
// Y-coord of generator point is value
0x07192B95_FFC8DA78_631011ED_6B24CDD5_73F977A1_1E794811
// write as 2nd part of vector C after one buffer word and one re-alignment word
at 0x098:
    Write PKA_RAM_BASE+0x098+0x00 0x1E794811
    Write PKA_RAM_BASE+0x098+0x04 0x73F977A1
    Write PKA_RAM_BASE+0x098+0x08 0x6B24CDD5
    Write PKA_RAM_BASE+0x098+0x0C 0x631011ED
    Write PKA_RAM_BASE+0x098+0x10 0xFFC8DA78
    Write PKA_RAM_BASE+0x098+0x14 0x07192B95
// The result point and scratchpad (vector D) follow vector C after one buffer
word and one
// re-alignment word, so these are located at PKA RAM Byte offset 0x0B8.
// Load pointer and length registers:
    Write PKA_APTR 0x000>>2 // Scalar ‘k’ pointer
    Write PKA_BPTR 0x018>>2 // Curve parameters ‘p’, ‘a’ & ‘b’ pointer
    Write PKA_CPTR 0x078>>2 // Generator point X & Y coordinates pointer
    Write PKA_DPTR 0x0B8>>2 // Result point X & Y coordinates/scratchpad pointer
    Write PKA_ALENGTH 0x00000006 // Scalar ‘k’ length in 32-bit words
    Write PKA_BLENGTH 0x00000006 // Curve parameters and coordinate lengths in 32-
bit words
// Start ECC point multiplication and wait until it’s done:
    Write PKA_FUNCTION 0x0000D000
// ‘Run’ bit set, ‘Sequencer Operations’ = 0b101
    Wait PKA_FUNCTION[15] == ‘0’
// ‘Run’ bit clears itself – Host can also use interrupt!
    Check PKA_SHIFT == 0x00000000
// Shift field value 0 indicates success – check this
// X-coord of result point is value
0x759B9F39_0E81D268_18C82BB9_CB42BCF5_0E0AE958_85BA3097
// written as 1st part of vector D at PKA RAM Byte offset 0x0B8:
    Check PKA_RAM_BASE+0x0B8+0x00== 0x85BA3097
    Check PKA_RAM_BASE+0x0B8+0x04== 0x0E0AE958
    Check PKA_RAM_BASE+0x0B8+0x08== 0xCB42BCF5
    Check PKA_RAM_BASE+0x0B8+0x0C== 0x18C82BB9
    Check PKA_RAM_BASE+0x0B8+0x10== 0x0E81D268
    Check PKA_RAM_BASE+0x0B8+0x14== 0x759B9F39
// Y-coord of result point is value
0xECA14640_F92EFF07_CAF2BD55_3FBE28EF_D043F28E_1CC3D238
// written as 2nd part of vector D at PKA RAM Byte offset 0x0D8:
    Check PKA_RAM_BASE+0x0D8+0x00== 0x1CC3D238
    Check PKA_RAM_BASE+0x0D8+0x04== 0xD043F28E
    Check PKA_RAM_BASE+0x0D8+0x08== 0x3FBE28EF
    Check PKA_RAM_BASE+0x0D8+0x0C== 0xCAF2BD55
    Check PKA_RAM_BASE+0x0D8+0x10== 0xF92EFF07
    Check PKA_RAM_BASE+0x0D8+0x14== 0xECA14640
```

ADSP-SC58x PKA Register Descriptions

Public Key Accelerator (PKA) contains the following registers.

Table 45-23: ADSP-SC58x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Vector_A Length

During execution of basic PKCP operations, the PKA_ALEN register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_ALEN register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

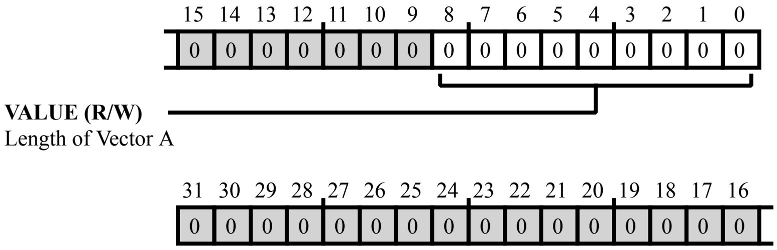

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Length of Vector A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-2: PKA_ALEN Register Diagram

Table 45-24: PKA_ALEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector A. Length (in 32-bit words) of Vector A.

PKA Vector_A Address

During execution of basic PKCP operations, the PKA_APTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_APTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

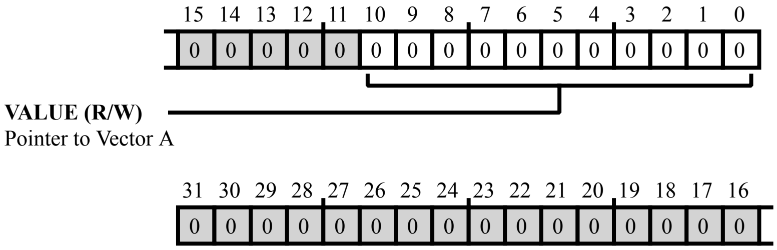

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Pointer to Vector A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-3: PKA_APTR Register Diagram

Table 45-25: PKA_APTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector A. The `PKA_APTR.VALUE` bit field is the location of Vector A within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Vector_B Length

During execution of basic PKCP operations, the `PKA_BLEN` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_BLEN` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

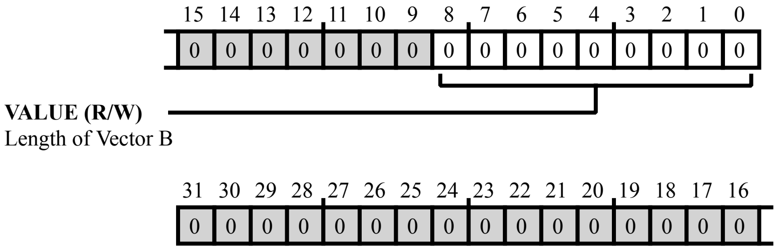

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Length of Vector B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-4: PKA_BLEN Register Diagram

Table 45-26: PKA_BLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector B. Length (in 32-bit words) of Vector B.

PKA Vector_B Address

During execution of basic PKCP operations, the `PKA_BPTR` register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the `PKA_BPTR` register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

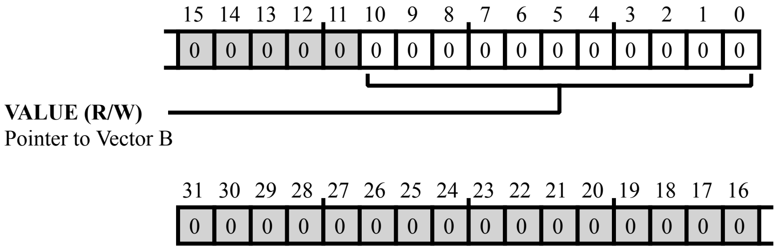

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Pointer to Vector B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-5: PKA_BPTR Register Diagram

Table 45-27: PKA_BPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector B. The `PKA_BPTR.VALUE` bit field is the location of Vector B within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Compare Result

The `PKA_COMPARE` register provides the result of a basic PKCP Compare operation. It is updated when the `PKA_FUNC.RUN` bit is reset at the end of that operation. The status after a complex sequencer operation is unknown.

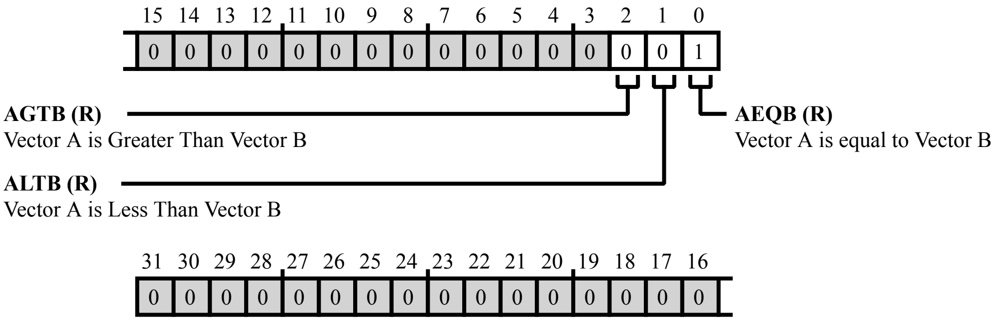

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
AGTB (R)
Vector A is Greater Than Vector B
ALTB (R)
Vector A is Less Than Vector B
AEQB (R)
Vector A is equal to Vector B
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-6: PKA_COMPARE Register Diagram

Table 45-28: PKA_COMPARE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	AGTB	Vector A is Greater Than Vector B. The `PKA_COMPARE.AGTB` bit shows the result of the basic compare operation is PKCP Vector_A is greater than Vector_B.
1 (R/NW)	ALTB	Vector A is Less Than Vector B. The `PKA_COMPARE.ALTB` bit shows the result of the basic compare operation is Vector_A is less than Vector_B.
0 (R/NW)	AEQB	Vector A is equal to Vector B. The `PKA_COMPARE.AEQB` bit shows the result of the basic compare operation is Vector_A is equal to Vector_B.

PKA Vector_C Address

During execution of basic PKCP operations, the PKA_CPTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_CPTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

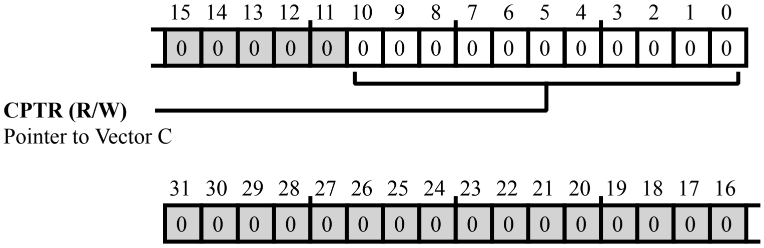

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CPTR (R/W)
Pointer to Vector C
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-7: PKA_CPTR Register Diagram

Table 45-29: PKA_CPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	CPTR	Pointer to Vector C. The PKA_CPTR.CPTR bit field is the location of Vector C within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Most-Significant-Word of Divide Remainder

The `PKA_DIVMSW` register indicates the (32-bit word) address in the PKA RAM where the most significant non-zero 32-bit word of the Remainder result for the basic Divide and Modulo operations is stored. Bits [4:0] are loaded with the bit number of the most significant non-zero bit in the most significant non-zero word when MS one control bit is set. For Divide, Modulo and MS one reporting, this register is updated when the `PKA_FUNC.RUN` bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating bits [4:0] of this register with the actual result's most significant bit location is done near the end of the operation. Note that the result is only meaningful if no errors were detected and that for ECC operations, the `PKA_DIVMSW` register provides information for the x-coordinate of the result point only.

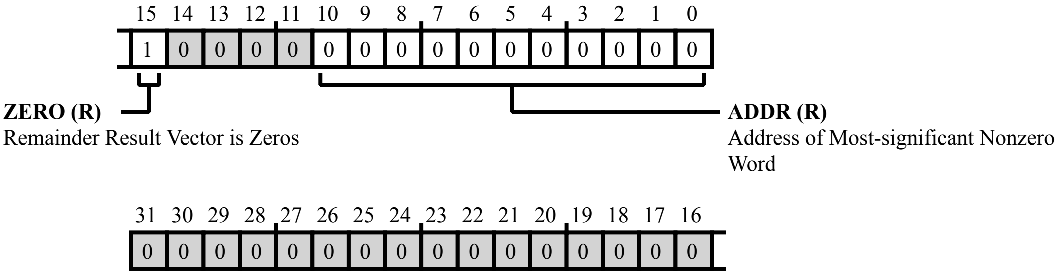

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ZERO (R)
Remainder Result Vector is Zeros
ADDR (R)
Address of Most-significant Nonzero Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-8: PKA_DIVMSW Register Diagram

Table 45-30: PKA_DIVMSW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Remainder Result Vector is Zeros. The `PKA_DIVMSW.ZERO` bit shows the remainder result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The `PKA_DIVMSW.ADDR` bit shows the address of the most significant non-zero 32-bit word of the remainder result vector in PKA RAM.

PKA Vector_D Address

During execution of basic PKCP operations, the PKA_DPTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_DPTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

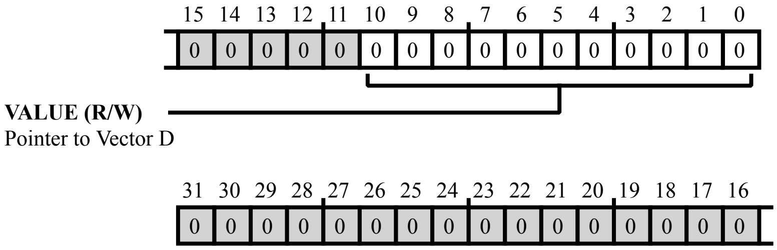

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Pointer to Vector D
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-9: PKA_DPTR Register Diagram

Table 45-31: PKA_DPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector D. The PKA_DPTR.VALUE bit field is the location of Vector D within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Function

The `PKA_FUNC` register contains the control bits to start basic PKCP as well as complex sequencer operations. The `PKA_FUNC.RUN` bit can be used to poll for the completion of the operation. Modifying bits [11:0] is made impossible during the execution of a basic PKCP operation.

During the execution of Sequencer controlled complex operations, this register is modified - the `PKA_FUNC.RUN` and `PKA_FUNC.STALLRSLT` bits are set to zero at the conclusion, but other bits are undefined.

Continuously reading this register to poll the `PKA_FUNC.RUN` bit is NOT allowed when executing complex sequencer operations (the sequencer cannot access the PKCP when this is done).

Leave at least one SCLK cycle between poll operations.

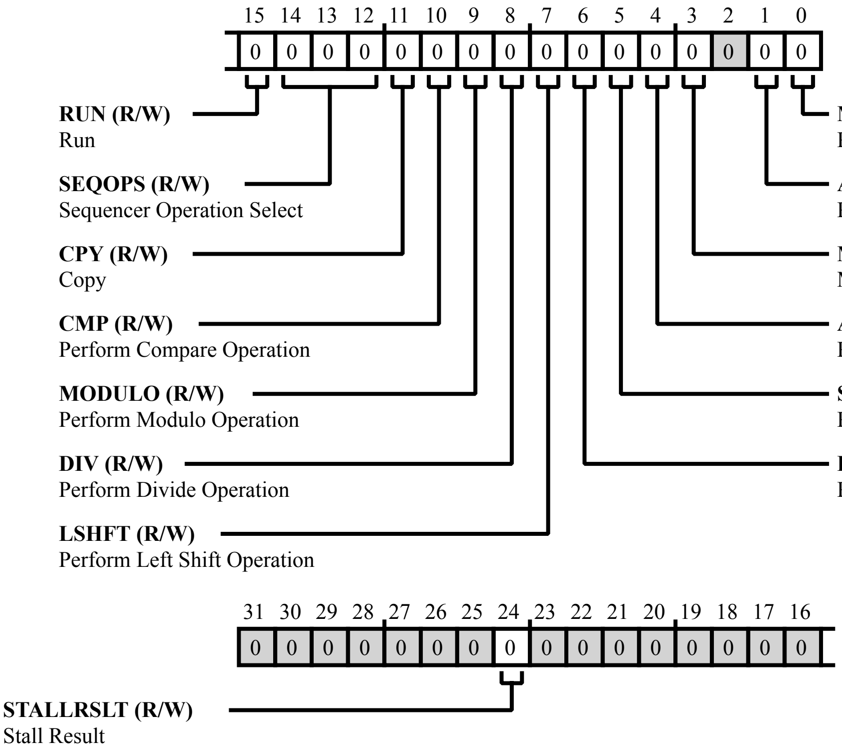

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RUN (R/W)
Run
SEQOPS (R/W)
Sequencer Operation Select
CPY (R/W)
Copy
CMP (R/W)
Perform Compare Operation
MODULO (R/W)
Perform Modulo Operation
DIV (R/W)
Perform Divide Operation
LSHFT (R/W)
Perform Left Shift Operation
MULT (R/W)
Perform Multiply Operation
ADDSUB (R/W)
Perform Combined Add/Subtract Operation
MSONE (R/W)
Most Significant One
ADD (R/W)
Perform Add Operation
SUB (R/W)
Perform Subtract Operation
RSHFT (R/W)
Perform Right Shift Operation
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STALLRSLT (R/W)
Stall Result

Figure 45-10: PKA_FUNC Register Diagram

Table 45-32: PKA_FUNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	STALLRSLT	Stall Result. When the `PKA_FUNC.STALLRSLT` bit is set, updating the `PKA_COMPARE`, `PKA_RESULTMSW` and `PKA_DIVMSW` registers, as well as resetting the Run bit, is stalled beyond the point that a running operation is actually finished. Use this to allow software enough time to read results from a previous operation when the newly started operation is known to take only a short amount of time. If a result is waiting, the result registers is updated and the Run bit is reset in the clock cycle following writing the Stall Result bit back to 0. The Stall result function may only be used for basic PKCP operations.
15 (R/W)	RUN	Run. Set the `PKA_FUNC.RUN` bit to instruct the PKA module to begin processing the basic PKCP or complex Sequencer operation. This bit is reset low automatically when the operation is complete. The complement of this bit is output as the pkaint1 interrupt. After a reset, the Run bit is always set to 1b but the first Sequencer firmware instruction sets this bit to 0 immediately after the hardware reset is released. A few clock cycles are needed before the first instruction is executed and the Run bit state has been propagated.
14:12 (R/W)	SEQOPS	Sequencer Operation Select. The `PKA_FUNC.SEQOPS` bit field select the complex Sequencer operation to perform. 0: None 1: ExpMod-CRT 2: ExpMod-ACT4 3: ECC-ADD 4: ExpMod-ACT2 5: ECC-MUL 6: ExpMod-variable 7: ModInv
11 (R/W)	CPY	Copy. Setting the `PKA_FUNC.CPY` bit performs the copy operation. For more information, see the "PKCP Vector Operations" section.
10 (R/W)	CMP	Perform Compare Operation. Setting the `PKA_FUNC.CMP` bit performs the compare operation. For more information, see the "PKCP Vector Operations" section.

Table 45-32: PKA_FUNC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	MODULO	Perform Modulo Operation. Setting the `PKA_FUNC.MODULO` bit performs the modulo operation. For more information, see the "PKCP Vector Operations" section.
8 (R/W)	DIV	Perform Divide Operation. Setting the `PKA_FUNC.DIV` bit performs the divide operation. For more information, see the "PKCP Vector Operations" section.
7 (R/W)	LSHFT	Perform Left Shift Operation. Setting the `PKA_FUNC.LSHFT` bit performs the Left shift operation. For more information, see the "PKCP Vector Operations" section.
6 (R/W)	RSHFT	Perform Right Shift Operation. Setting the `PKA_FUNC.RSHFT` bit performs the right shift operation. For more information, see the "PKCP Vector Operations" section.
5 (R/W)	SUB	Perform Subtract Operation. Setting the `PKA_FUNC.SUB` bit performs the subtract operation. For more information, see the "PKCP Vector Operations" section.
4 (R/W)	ADD	Perform Add Operation. Setting the `PKA_FUNC.ADD` bit performs the add operation. For more information, see the "PKCP Vector Operations" section.
3 (R/W)	MSONE	Most Significant One. Setting the `PKA_FUNC.MSONE` bit loads the location of the Most Significant one bit within the result word indicated in the `PKA_RESULTMSW` register into bits [4:0] of the `PKA_DIVMSW` register can only be used with basic PKCP operations, except for Divide, Modulo and Compare.
1 (R/W)	ADDSUB	Perform Combined Add/Subtract Operation. Setting the `PKA_FUNC.ADDSUB` bit performs the combined Add/Subtract operation. For more information, see the "PKCP Vector Operations" section.
0 (R/W)	MULT	Perform Multiply Operation. Setting the `PKA_FUNC.MULT` bit performs the multiply operation. For more information, see the "PKCP Vector Operations" section.

Start of PKA RAM space

The `PKA_RAM` register provides the the starting location of the RAM space to hold the input, output and other vectors.

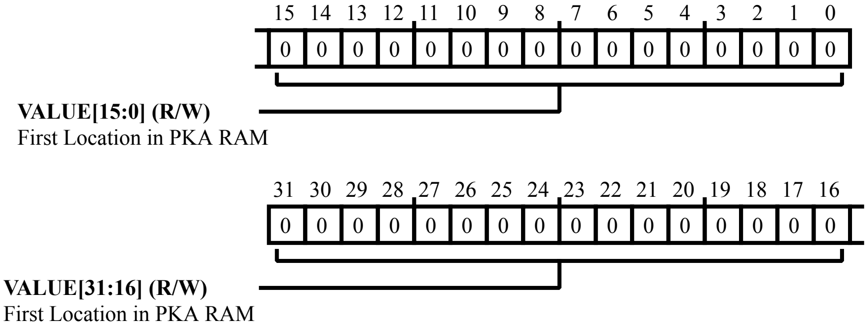

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
First Location in PKA RAM
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
First Location in PKA RAM

Figure 45-11: PKA_RAM Register Diagram

Table 45-33: PKA_RAM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	First Location in PKA RAM. The `PKA_RAM.VALUE` bit field provides the the starting location of the RAM space to hold the input, output and other vectors.

PKA Most-Significant-Word of Result Vector

The PKA_RESULTMSW register indicates the (word) address in the PKA RAM where the most significant non-zero 32-bit word of the result is stored and should be ignored for modulo operations. For basic PKCP operations, the PKA_RESULTMSW register is updated when the `PKA_FUNC.RUN` bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating the final value matching the actual result is done near the end of the operation. Note that the result is only meaningful if no errors are detected and that for ECC operations, the PKA_DIVMSW register provides information for the x-coordinate of the result point only.

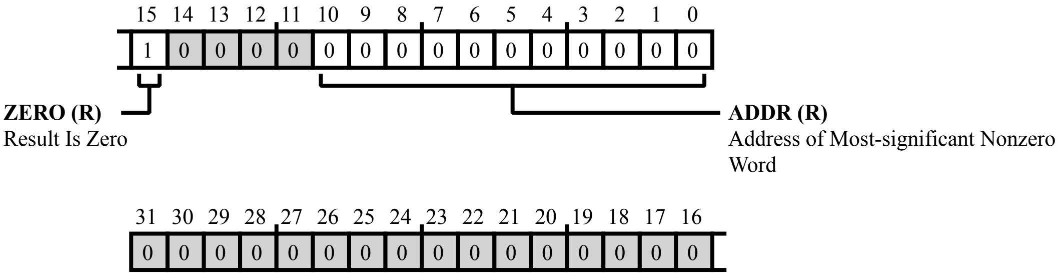

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ZERO (R)
Result Is Zero
ADDR (R)
Address of Most-significant Nonzero Word
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-12: PKA_RESULTMSW Register Diagram

Table 45-34: PKA_RESULTMSW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Result Is Zero. The `PKA_RESULTMSW.ZERO` bit indicates the result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The `PKA_RESULTMSW.ADDR` bit is the address of the most significant non-zero 32-bit word of the result vector in PKA RAM.

PKA Bit Shift Value

For basic PKCP operations, modifying the contents of the PKA_SHIFT register is made impossible while the operation is being performed. For the ExpMod-variable and ExpMod-CRT operations, the PKA_SHIFT register is used to indicate the number of odd powers to use (directly as a value in the range 1-16). For the ModInv and ECC operations, this register is used to hold a completion code.

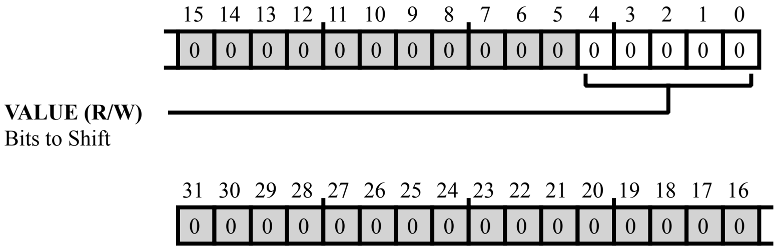

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Bits to Shift
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 45-13: PKA_SHIFT Register Diagram

Table 45-35: PKA_SHIFT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Bits to Shift. The `PKA_SHIFT.VALUE` bit field is the number of bits to shift the input vector (in the range 0-31) during a Rshift or Lshift operation.

46 Public Key Interrupt Controller (PKIC)

The Public Key Accelerator (PKA) and the True Random Number Generator (TRNG) share a common interrupt controller, the Public Key Interrupt Controller (PKIC). The host processor configures the PKIC to generate interrupts when certain operations are complete or interrupts are caused by errors.

PKIC Functional Description

The main purpose and function of the PKIC is to capture the interrupts from different sources, either from the PKA or the TRNG and combine them to one interrupt output. The interrupt controller is managed using the following register groups:

- Control for polarity, edge, and level detection and enabling of individual interrupts
- Acknowledgment (to clear edge detected interrupts)
- Status:
  - A raw source status register after edge detection, if edge selected.
  - A status register after masking with the interrupt enable control bits.

ADSP-SC58x PKIC Register List

The Public Key Processor Interrupt Controller (PKIC) provides security-related features. A set of registers governs PKIC operations. For more information on PKIC functionality, see the PKIC register descriptions.

Table 46-1: ADSP-SC58x PKIC Register List

Name	Description
PKIC_ACK	Acknowledge Register
PKIC_EN_CLR	Enable Clear Register
PKIC_EN_CTL	Enable Control Register
PKIC_EN_SET	Enable Set Register
PKIC_EN_STAT	Enabled Status Register
PKIC_POL_CTL	Polarity Control Register

Table 46-1: ADSP-SC58x PKIC Register List (Continued)

Name	Description
PKIC_RAW_STAT	Raw Status Register
PKIC_TYPE_CTL	Type Control Register

ADSP-SC58x PKIC Interrupt List

Table 46-2: ADSP-SC58x PKIC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
129	PKIC0_IRQ	PKIC0 Interrupt	Level	

PKIC Programming Model

The following sections provide information on how to program the PKIC.

Enabling/Disabling and Status

The PKIC_EN_STAT register provides the mask to which interrupt source is enabled. There are two status registers, PKIC_RAW_STAT and PKIC_EN_STAT. They allow the host processor to read the status of the interrupt source before and after the mask is applied.

Level or Edge

All of the interrupt sources are level or edge events. The PKIC_TYPE_CTL register configures each interrupt to either level or edge. The PKIC_POL_CTL register controls the polarity of the signal.

These interrupts are latched at both status registers in case edge detection is selected. The edge detectors are reset by clearing the interrupts using the PKIC_ACK registers.

PKIC Programming Concepts

The following concepts help with proper programming for the PKIC module.

Interrupt Handling

When an interrupt is triggered, the handler must first examine this module, the PKIC to determine what triggered the interrupt. By reading the PKIC_EN_STAT, the bits that are set are the pending interrupts of the ones that were not masked. After determining the source of the interrupt, the appropriate action must be taken to service the interrupt from the corresponding module, the PKA, or the TRNG. After handling the interrupt in a particular module, the corresponding interrupt must be acknowledged and cleared in the PKIC to allow further interrupts.

While handling an interrupt, any events that would cause another interrupt would happen without triggering another interrupt.

Overlapping Registers

There are two sets of overlapping registers in the PKIC. The `PKIC_EN_STAT` and `PKIC_ACK` registers share one address. If read, the register tells which enabled interrupts are pending. If written to (W1C), the interrupt is acknowledged and cleared. The `PKIC_RAW_STAT` and `PKIC_EN_SET` registers are another pair that share the address. When read, the register tells which interrupts are pending and if written to will enabled certain interrupts. This register cannot be used to disable any interrupts.

ADSP-SC58x PKIC Register Descriptions

Public Key Processor Interrupt Controller (PKIC) contains the following registers.

Table 46-3: ADSP-SC58x PKIC Register List

Name	Description
`PKIC_ACK`	Acknowledge Register
`PKIC_EN_CLR`	Enable Clear Register
`PKIC_EN_CTL`	Enable Control Register
`PKIC_EN_SET`	Enable Set Register
`PKIC_EN_STAT`	Enabled Status Register
`PKIC_POL_CTL`	Polarity Control Register
`PKIC_RAW_STAT`	Raw Status Register
`PKIC_TYPE_CTL`	Type Control Register

Acknowledge Register

The `PKIC_ACK` register is used to acknowledge the interrupt and clear the corresponding interrupt bit in the status register.

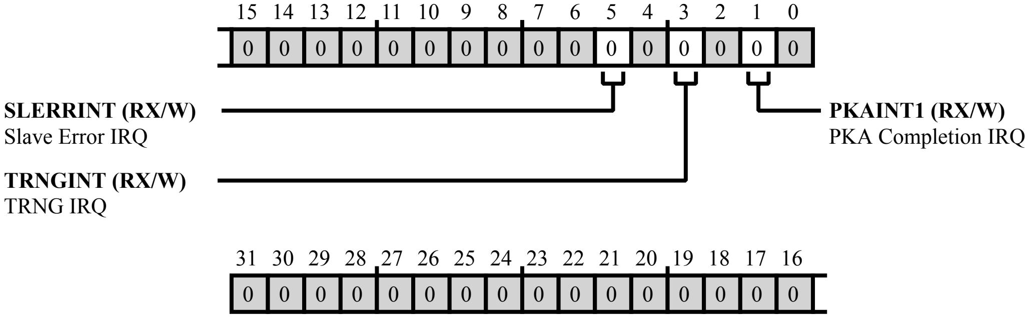

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (RX/W)
Slave Error IRQ
TRNGINT (RX/W)
TRNG IRQ
PKAINT1 (RX/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-1: PKIC_ACK Register Diagram

Table 46-4: PKIC_ACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The `PKIC_ACK.SLERRINT` bit is the acknowledge bit for the Slave Error interrupt. When set =1 the `PKIC_ACK.SLERRINT` bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	
		0	Do not acknowledge interrupt and clear status bit
		1	Acknowledge interrupt and clear status bit
3 (RX/W)	TRNGINT	TRNG IRQ. The `PKIC_ACK.TRNGINT` bit is the acknowledge bit for the TRNG interrupt. When set =1 the `PKIC_ACK.TRNGINT` bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	
		0	Do not acknowledge interrupt and clear status bit
		1	Acknowledge interrupt and clear status bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The `PKIC_ACK.PKAINT1` bit is the acknowledge bit for the PKA completion interrupt. When set =1 the `PKIC_ACK.PKAINT1` bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	

Enable Clear Register

The PKIC_EN_CLR register allows the user to disable certain interrupts without enabling others. The disabled interrupts are also reflected in PKIC_EN_CTL register.

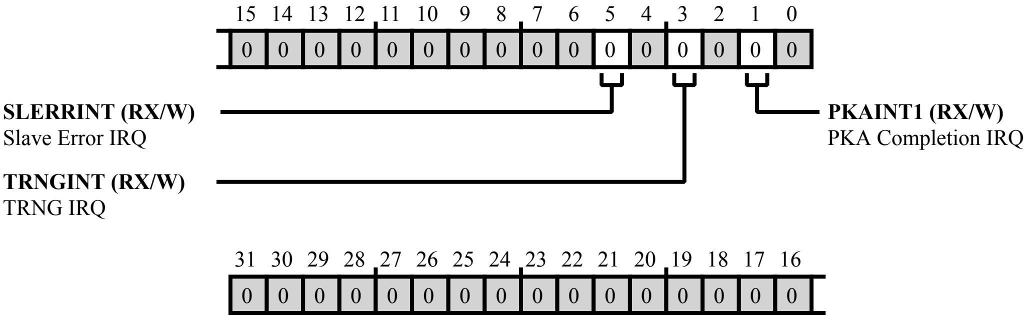

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (RX/W)
Slave Error IRQ
TRNGINT (RX/W)
TRNG IRQ
PKAINT1 (RX/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-2: PKIC_EN_CLR Register Diagram

Table 46-5: PKIC_EN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The PKIC_EN_CLR.SLERRINT bit is the individual disable for the Slave Error interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically.	
		0	No action
		1	Clear/reset corresponding CTL bit
3 (RX/W)	TRNGINT	TRNG IRQ. The PKIC_EN_CLR.TRNGINT bit is the individual disable for the TRNG interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.	
		0	No action
		1	Clear/reset corresponding CTL bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The PKIC_EN_CLR.PKAINT1 bit is the individual disable for the PKA Completion interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.	
		0	No action
		1	Clear/reset corresponding CTL bit

Enable Control Register

The PKIC_EN_CTL register provides individual enable bits for the interrupt sources.

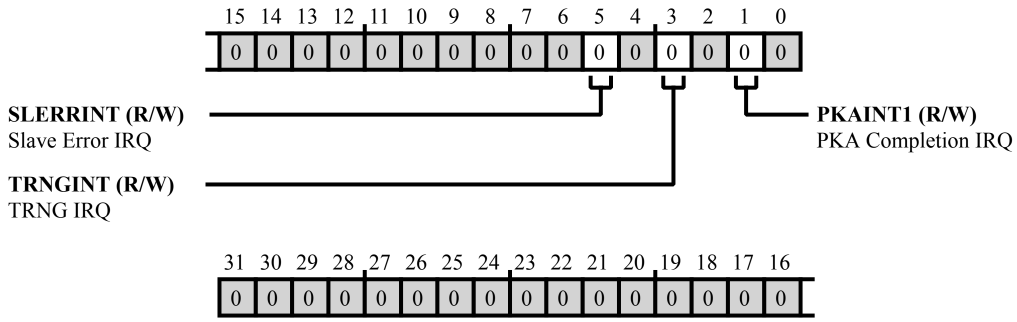

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (R/W)
Slave Error IRQ
TRNGINT (R/W)
TRNG IRQ
PKAINT1 (R/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

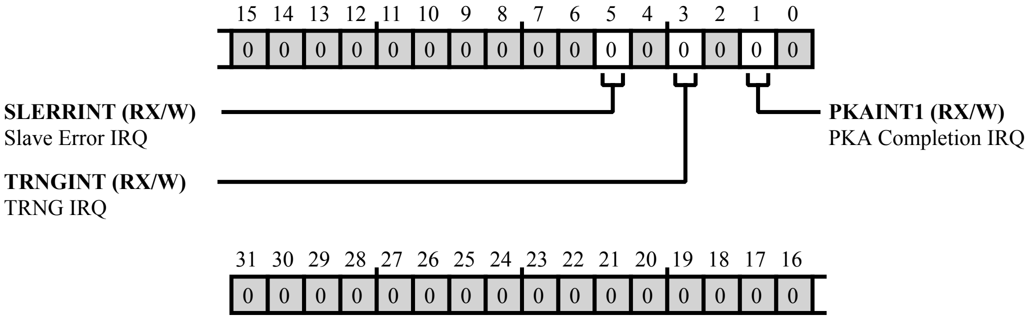
Figure 46-3: PKIC_EN_CTL Register Diagram

Table 46-6: PKIC_EN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The `PKIC_EN_CTL.SLERRINT` bit enables control for the Slave Error interrupt.	
		0	Disable interrupt
		1	Enable interrupt
3 (R/W)	TRNGINT	TRNG IRQ. The `PKIC_EN_CTL.TRNGINT` bit enables control for the TRNG interrupt.	
		0	Disable interrupt
		1	Enable interrupt
1 (R/W)	PKAINT1	PKA Completion IRQ. The `PKIC_EN_CTL.PKAINT1` bit enables control for the PKA completion interrupt.	
		0	Disable interrupt
		1	Enable interrupt

Enable Set Register

The PKIC_EN_SET register allows the user to only set certain interrupt sources without disabling any others. The enabled interrupts are reflected in PKIC_EN_CTL register.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (RX/W)
Slave Error IRQ
TRNGINT (RX/W)
TRNG IRQ
PKAINT1 (RX/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-4: PKIC_EN_SET Register Diagram

Table 46-7: PKIC_EN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The PKIC_EN_SET.SLERRINT bit is the individual enable for the Slave Error interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically. 0=no effect	
		0	No effect
		1	Enable interrupt
3 (RX/W)	TRNGINT	TRNG IRQ. The PKIC_EN_SET.TRNGINT bit is the individual enable for the TRNG interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically.	
		0	No effect
		1	Enable interrupt
1 (RX/W)	PKAINT1	PKA Completion IRQ. The PKIC_EN_SET.PKAINT1 bit is the individual enable for the PKA Completion interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically.	
		0	No effect
		1	Enable interrupt

Enabled Status Register

The PKIC_EN_STAT register is used to tell the status of the interrupts after the gating with the PKIC_EN_CTL register.

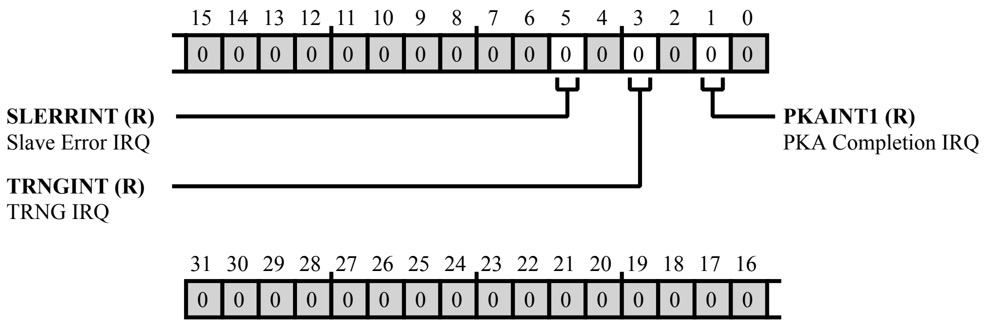

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (R)
Slave Error IRQ
TRNGINT (R)
TRNG IRQ
PKAINT1 (R)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-5: PKIC_EN_STAT Register Diagram

Table 46-8: PKIC_EN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	SLERRINT	Slave Error IRQ. The PKIC_EN_STAT.SLERRINT bit provides the status of the Slave Error interrupt (after masking from the PKIC_EN_CTL register). 0 Interrupt is inactive 1 Interrupt is pending
3 (R/NW)	TRNGINT	TRNG IRQ. The PKIC_EN_STAT.TRNGINT bit provides the status of the TRNG interrupt (after masking from the PKIC_EN_CTL register). 0 Interrupt is inactive 1 Interrupt is pending
1 (R/NW)	PKAINT1	PKA Completion IRQ. The PKIC_EN_STAT.PKAINT1 bit provides the status of the PKA Completion interrupt (after masking from the PKIC_EN_CTL register). 0 Interrupt is inactive 1 Interrupt is pending

Polarity Control Register

The `PKIC_POL_CTL` register is used to configure the signal polarity for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to (high level/rising edge or low level/falling edge).

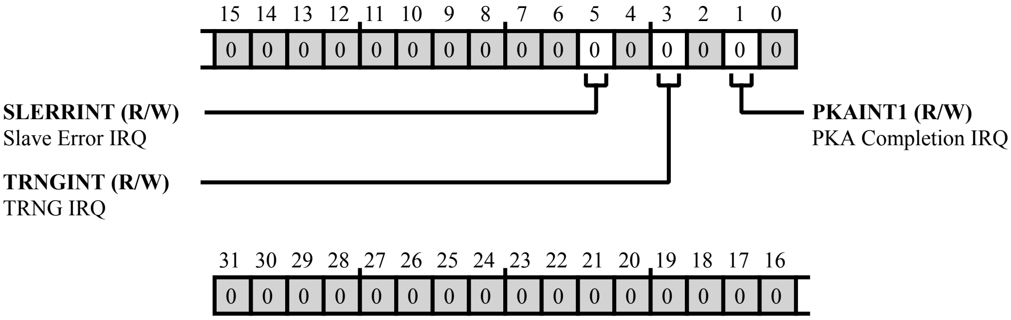

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (R/W)
Slave Error IRQ
TRNGINT (R/W)
TRNG IRQ
PKAINT1 (R/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-6: PKIC_POL_CTL Register Diagram

Table 46-9: PKIC_POL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The `PKIC_POL_CTL.SLERRINT` bit provides polarity control for the Slave Error interrupt.	
		0	Low level/falling edge
		1	High level/rising edge
3 (R/W)	TRNGINT	TRNG IRQ. The `PKIC_POL_CTL.TRNGINT` bit provides polarity control for the TRNG interrupt.	
		0	Low level/falling edge
		1	High level/rising edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The `PKIC_POL_CTL.PKAINT1` bit provides polarity control for PKA completion interrupt.	
		0	Low level/falling edge
		1	High level/rising edge

Raw Status Register

The `PKIC_RAW_STAT` register reflects the status of the individual interrupts before masking with the `PKIC_EN_CTL` register.

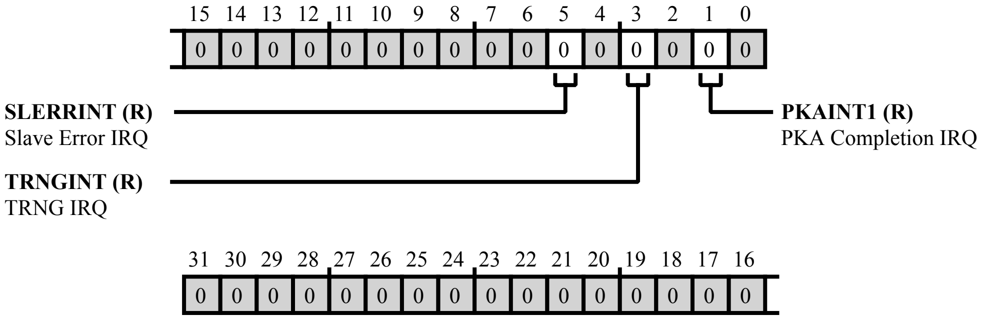

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (R)
Slave Error IRQ
TRNGINT (R)
TRNG IRQ
PKAINT1 (R)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-7: PKIC_RAW_STAT Register Diagram

Table 46-10: PKIC_RAW_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	SLERRINT	Slave Error IRQ. The `PKIC_RAW_STAT.SLERRINT` bit provides the raw status of the Slave Error interrupt.	
		0	Inactive interrupt
		1	Pending interrupt
3 (R/NW)	TRNGINT	TRNG IRQ. The `PKIC_RAW_STAT.TRNGINT` bit provides the raw status of the TRNG interrupt where 1=pending and 0=inactive.	
		0	Inactive interrupt
		1	Pending interrupt
1 (R/NW)	PKAINT1	PKA Completion IRQ. The `PKIC_RAW_STAT.PKAINT1` bit provides raw status of the PKA completion interrupt where 1=pending and 0=inactive.	
		0	Inactive interrupt
		1	Pending interrupt

Type Control Register

The `PKIC_TYPE_CTL` register is used to configure the signal type for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to level or edge.

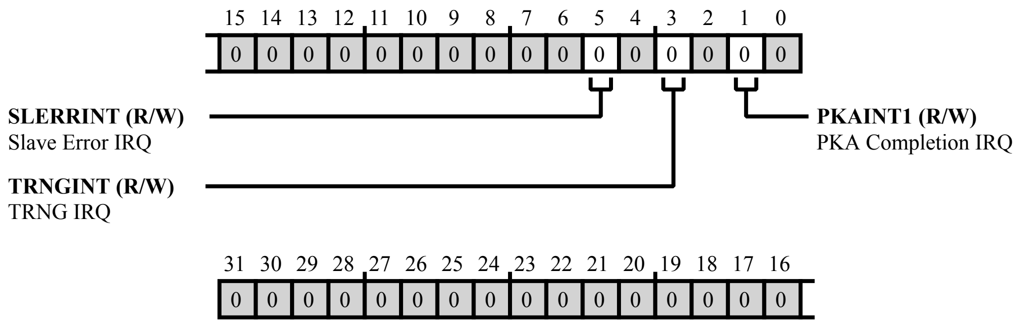

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SLERRINT (R/W)
Slave Error IRQ
TRNGINT (R/W)
TRNG IRQ
PKAINT1 (R/W)
PKA Completion IRQ
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 46-8: PKIC_TYPE_CTL Register Diagram

Table 46-11: PKIC_TYPE_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The `PKIC_TYPE_CTL.SLERRINT` bit provides signal type control for the Slave Error interrupt.	
		0	Level
		1	Edge
3 (R/W)	TRNGINT	TRNG IRQ. The `PKIC_TYPE_CTL.TRNGINT` bit provides signal type control for the TRNG interrupt.	
		0	Level
		1	Edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The `PKIC_TYPE_CTL.PKAINT1` bit provides signal type control for the PKA completion interrupt.	
		0	Level
		1	Edge

47 True Random Number Generator (TRNG)

The TRNG engine provides a true, non-deterministic, noise source for generating keys, Initialization Vectors (IVs), and other random number requirements. Other non-cryptographic purposes include statistical sampling, retry timers for communications protocols and noise generation.

TRNG Features

The TRNG features include:

- Hardware-based non-deterministic random number generator
- ANSI X9.31 postprocessing (depending on processor)
- Redundant 'fail-safe' design with self-test circuits
- Reliable shot noise oscillator implementation with auto-tuning
- Debug output to allow monitoring of internal operation
- Alarm count overflow and auto-tuning error interrupts
- Buffer to allow generation of large blocks of random data in the background

TRNG Functional Description

The following sections provide details on the function of the TRNG module.

ADSP-SC58x TRNG Register List

The True Random Number Generator (TRNG) provides random number generation, intended mainly for security-related applications. A set of registers governs TRNG operations. For more information on TRNG functionality, see the TRNG register descriptions.

Table 47-1: ADSP-SC58x TRNG Register List

Name	Description
`TRNG_ALMCNT`	TRNG Alarm Counter Register

Table 47-1: ADSP-SC58x TRNG Register List (Continued)

Name	Description
TRNG_ALMMSK	TRNG Alarm Mask Register
TRNG_ALMSTP	TRNG Alarm Stop Register
TRNG_CFG	TRNG Configuration Register
TRNG_CNT	Counter Register
TRNG_CTL	TRNG Control Register
TRNG_FRODETUNE	TRNG FRO De-tune Register
TRNG_FROEN	TRNG FRO Enable Register
TRNG_INPUT[n]	TRNG Input Registers
TRNG_INTACK	TRNG Interrupt Acknowledge Register
TRNG_LFSR_H	TRNG LFSR Access Register
TRNG_LFSR_L	TRNG LFSR Access Register
TRNG_LFSR_M	TRNG LFSR Access Register
TRNG_MONOBITCNT	TRNG Monobit Test Result Register
TRNG_OUTPUT[n]	TRNG Output Registers
TRNG_POKER[n]	TRNG Poker Test Result Registers
TRNG_RUNCNT	TRNG Run Count Registers
TRNG_RUN[n]	TRNG Run Test State and Result Registers
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register

Random Number Generation

The random numbers that the TRNG generates are produced by sampling Free Running Oscillators (FRO). The (`TRNG_CTL.STARTUPCYC`) bit field along with `TRNG_CFG.MINREFCYC` (minimum refill cycles) and `TRNG_CFG.MAXREFCYC` (maximum refill cycles) bit fields determine the number of samples taken to generate the first random value and subsequent random values.

1. After enabling the TRNG (`TRNG_CTL.TRNGEN` bit =1), a number of FRO output samples defined by the `TRNG_CTL.STARTUPCYC` bit field are gathered in the main linear-feedback shift register (LFSR) before taking a snapshot of the LFSR and storing that snapshot in the random data buffer (after optional post-processing).
2. After taking a snapshot of the LFSR, a number of FRO output samples defined by the `TRNG_CFG.MINREFCYC` bit field, are gathered in the main LFSR. If the random data buffer is full, sample-taking continues until the number of samples (counting from the snapshot) matches the number of samples defined by the `TRNG_CFG.MAXREFCYC` bit field. At that point, the TRNG switches off the FROs and powers down.

3. If, after the `TRNG_CFG.MINREFCYC` sampling period, the random data buffer is not completely filled, a new snapshot of the LFSR is taken and stored in the random data buffer (after optional postprocessing). Control branches back to point step 2 above and a new `TRNG_CFG.MINREFCYC` sample period starts.

Locking Detection and Prevention

Lock detection in functional mode uses the sampled outputs of the individual FROs. A FRO alarm event is declared when a repeating pattern (of up to four sample lengths) is detected continuously for the number of samples defined in the alarm threshold `TRNG_ALMCNT.ALMTHRESH` bit field. The alarm event is logged by setting the bit that corresponds to the FRO that caused the alarm in the `TRNG_ALMMSK` register. If that bit was already set, the corresponding bit in the `TRNG_ALMSTP` register is set. The FRO is switched off to prevent further alarm events from that FRO. If the `TRNG_ALMMSK` register bit was not yet set, the FRO is restarted automatically in an attempt to break locking.

The shutdown count field in the alarm count `TRNG_ALMCNT.SHDNCNT` register monitors the number of FROs switched off. (It counts the number of 1 bits in the `TRNG_ALMSTP` register.) The shutdown threshold field (`TRNG_ALMCNT.SHDNTHRESH`) can be configured to generate the shutdown overflow interrupt (`TRNG_STAT.SHDNOVR`). When the shutdown count in the `TRNG_ALMCNT.SHDNCNT` bit field exceeds the shutdown threshold in the `TRNG_ALMCNT.SHDNTHRESH` bit field, the shutdown overflow bit (`TRNG_STAT.SHDNOVR`) is set to 1 (which can be used to generate an interrupt).

Software can use two strategies for the TRNG operation:

- *Monitored Operation*. Software checks the `TRNG_ALMMSK` register at regular intervals (on the order of seconds). If a bit is set in that register, then the program must also check the `TRNG_ALMSTP` register to determine if a FRO was shut down due to multiple alarm events. If no FROs are shut down, the program clears the `TRNG_ALMMSK` register to remove the incidental alarm events. If one or more FROs are shut down, the host processor can modify the delay selection of those FROs using the `TRNG_FRODETUNE` register to prevent further locking. For this type of operation, the shutdown threshold is normally set to a low value (two, for example). The shutdown overflow interrupt can then be used to signal abnormal operation conditions or the breakdown of FROs.
- *Unmonitored Operation.* Software sets the shutdown threshold to the acceptable number of FROs to be shut down before taking corrective actions. It then uses the shutdown overflow interrupt to initiate corrective actions (clearing the `TRNG_ALMMSK` and `TRNG_ALMSTP` registers, toggling bits in the `TRNG_FRODETUNE` register). The software must monitor the time interval between these interrupts. If they occur too often (for example, within a minute after each other), this frequency indicates abnormal operating conditions or the breakdown of FROs.

Run Testing

Run Test

The TRNG block counts the number of consecutive zeros and ones (runs) in the data stream shifted into the main LFSR. The run length and bit value is then used to increment a specific bucket counter for these values. After

20,000 bits, the bucket counters must be within specified limits for this test to pass. If not, a run fail interrupt (RUNFAIL) is generated.

Table 47-2: Allowable Limits on Runs of 0's and 1's

Run Count	Bit Value	Min (inclusive)	Max (inclusive)
1	0	2267	2733
	1	2267	2733
2	0	1079	1421
	1	1079	1421
3	0	502	748
	1	502	748
4	0	233	402
	1	233	402
5	0	90	223
	1	90	223
6 and up	0	90	233
	1	90	233

Long Run Test

The long run test fails immediately when a run longer than 33 bits is found and a long run fail (`TRNG_STAT.LRUNFAIL`) interrupt is generated.

Noise Source Test

A noise source failure is declared when a run of 48 or more identical bits is found and a noise fail interrupt (`TRNG_STAT.NOISEFAIL`) is generated.

The status and counts are stored in the `TRNG_RUNCNT` and `TRNG_RUN[n]` registers. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), the run fail acknowledge (`TRNG_INTACK.RUNFAIL`), or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

Monobit Testing

The TRNG block performs the monobit test on blocks of 20,000 bits (in parallel with the run test and poker test). It monitors the number of zeros and ones in the data stream shifted into the main LFSR. At the start of the block, the counter is initialized to 10,000. Each 1 value increments the counter, each 0 value decrements the counter. After 20,000 bits, the counter value must be within 9310–10690 (inclusive) for this test to pass. If not, a monobit fail interrupt (`TRNG_STAT.MBITFAIL`) is generated. The AIS-31 standard (test T1, ref 0) specifies this run time testing of the TRNG and the parameters.

NOTE: The actual limits stated here are different than the limits stated in the AIS-31 standard due to the implementation. The circuitry in the TRNG uses an up-down counter while the standard just evaluates the sum of the 20,000 bits.

When the continue poker (`TRNG_TEST.CONTPKR`) bit is set to 1, the test is not stopped after 20,000 bits. The counter keeps incrementing and decrementing (protected against overflow and underflow). But, no actual limit checking happens. The offset from starting value 10,000 indicates the balance of 0 and 1 bits that were checked since the start of the continuous test. The offset is twice the number of missing or extra 1 bits. (An extra 1 bit adds an increment operation and removes one decrement operation. So, having 10,001 1 bits in the block gives a counter value of 10,002.)

Poker Testing

The poker test is run in parallel with the monobit test and run test. Counters in the `TRNG_POKER[n]` registers are used to count the occurrences of one specific 4-bit value in the data stream fed into the main LFSR. All of the counters are decremented by one every 64 data bits and reset to their start value every 20,000 bits. All counters start at a value of –1 and are decremented 312 times during the 20,000-bit test run.

Each 8-bit counter holds a two's complement value and does not overflow past the range of –128 to +127. At the end of the 20,000-bits block, the values of the counters that contain a single 1 bit appended at the least significant bit side are individually squared and then added. The poker test fails with a poker fail (PKRFAIL) interrupt when:

- the resulting sum (accumulated in the `TRNG_MONOBITCNT` register) is outside the range 1288 – 71750 (inclusive), or
- one of the counters tries to increment or decrement outside its limit range

NOTE: The poker test fails when the 4-bit values of the data stream are distributed too evenly (with eight counters having incremented 312 times and the others incremented 313 times). This failure is intentional. The minimum mean deviation from the expected value of 312.5 is 4.5. Failure at counter overflow is not an official part of the poker test as specified in the AIS-31 standard (ref 0). It can be shown that the maximum deviation for one counter's value from the mean value of 312.5 (without the poker test failing) is 129.5. Since this deviation is more than 40% of the mean value, it indicates that something is wrong. Here, the counter overflow failure is combined with the official poker test failure.

When the continue poker (`TRNG_TEST.CONTPKR`) bit is set to 1, the test does not stop after 20,000 bits. The counters keep incrementing and decrementing (the latter every 64 bits). A PKRFAIL interrupt is generated when one of the counters tries to increment or decrement outside its limit range.

Data for Tests

For the monobit test, run test and poker test circuits self-test, the test data written to the `TRNG_INPUT[n]`0 register is used for the monobit test and run test bit-by-bit. It executes from bit 31 down to bit 0. For the poker test, the written test data is used in eight blocks of 4 bits, starting from bits 31:28 down to bits 3:0.

When the `TRNG_TEST.CONTPKR` bit =0 during run test and poker test circuits self-tests, the state of the poker and runs test status registers is frozen after all calculations are made and after inputting 20,000 test bits. This state allows time to read the test results. These calculations take around 20 clock cycles to complete. The status registers are reset to their starting states when the first word for the next test block of test bits is written to the `TRNG_INPUT[n]`0 register. Then, the contents of this word are processed.

TRNG Block Diagram

The *System Block Diagram of the TRNG* diagram shows the system block for the TRNG. The system includes:

- Free Running Oscillators (FROs) that are the source of sampled bits
- A post-processing unit (processor dependent) for standards compliance
- Test circuitry to detect non-randomness due to failures in the system

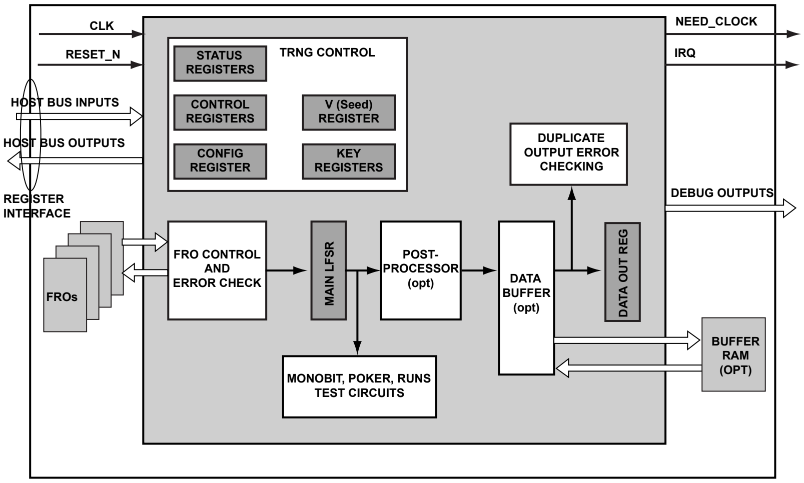

CLK
RESET_N
HOST BUS INPUTS
HOST BUS OUTPUTS
REGISTER INTERFACE
STATUS REGISTERS
TRNG CONTROL
CONTROL REGISTERS
V (Seed) REGISTER
CONFIG REGISTER
KEY REGISTERS
DUPLICATE OUTPUT ERROR CHECKING
NEED_CLOCK
IRQ
DEBUG OUTPUTS
FROs
FRO CONTROL AND ERROR CHECK
MAIN LFSR
POST-PROCESSOR (opt)
DATA BUFFER (opt)
DATA OUT REG
BUFFER RAM (OPT)
MONOBIT, POKER, RUNS TEST CIRCUITS

Figure 47-1: System Block Diagram of the TRNG

TRNG Architectural Concepts

The random numbers are accessible to the host processor in four 32-bit registers allowing a single burst read of a 128-bit random number. Acknowledging the data ready (interrupt) state causes the TRNG to move a new value (when available in the data buffer) to the TRNG output register. The TRNG always tries to keep the data buffer completely full. Pulling out data starts the regeneration of a new number by:

- Enabling the FROs
- Capturing their outputs in the LFSR, and
- Cryptographically postprocessing the values from the LFSR

The process produces new random values to replenish the buffer.

The major functional blocks of the TRNG module are:

- The actual TRNG core with control and test circuits, optional post-processor, and optional data buffer control logic

- Free Running Oscillators (FROs) instantiated outside the TRNG core
- A 128-byte buffer RAM instantiated outside the TRNG core

In the TRNG core, the true entropy source uses FROs as the basic building block. The accumulation of timing jitter, caused (for the largest part) by shot noise, creates uncertainty intervals for the output transitions of each FRO. Sampling within an uncertainty interval generates a single bit of entropy, which is 'accumulated' in a LFSR. As the uncertainty interval is very narrow compared to the cycle time of a FRO, the mean amount of entropy generated per sample is small (less than 1/100 bit per sample). To increase the entropy generation rate, multiple FROs are used in parallel.

The FROs are asynchronous to one another and asynchronous to the sampling clock to make their behavior truly non-deterministic.

The output signals of the FROs are sampled at regular intervals (in general, at the TRNG core module clock frequency). The samples feed into an error detection circuit that checks for repeating patterns coming out of a FRO. If a repeating pattern persists for a configurable number of samples, the FRO is suspect of having synchronized to (a harmonic of) the sampling interval. This activity drastically reduces the amount of entropy generated by that FRO. The error detection circuit signals this activity as a FRO error event.

Error events can occur during normal operation. The FRO control circuit attempts to restart a FRO that on a first error event. A second error event causes an automatic shutdown of the FRO. Because there are multiple FROs, shutting down a FRO reduces the amount of entropy generated, but it does not immediately jeopardize the TRNG operation. A limit can be configured below which the number of operational FROs does not drop. If this limit is crossed, an interrupt can be generated on the host processor. Software on the host processor can then attempt to prevent frequent locking of a FRO by *de-tuning* it to a slightly different frequency.

An XOR tree combines the sampled outputs and feeds them into an 81-bit LFSR to accumulate entropy and whiten the random bits stream.

NOTE: Here, *whitening* means balancing the number of one and zero bits in the stream.

TRNG Operating Modes

The TRNG has the following operating modes.

Normal Reading Mode

In normal reading mode, random data can only be read out of the output registers of the TRNG module when the ready bit (`TRNG_STAT.RDY`) =1. Acknowledging the data by writing a 1 to the ready acknowledge bit in the `TRNG_INTACK.RDY` register clears the ready bit from the status register and clears the output registers. The registers remain at zero until the next 128-bit data block is available.

Secure Reading Mode

An attacker can try to read the output registers (without acknowledging the data) to obtain a copy of data to be read later by an application. To block this attack, secure reading mode is used. In this mode, enable reading from the output registers (by writing 0x00 to the open read gate bits (`TRNG_INTACK.OPENRDGATE`) before it is possible to access the output registers. Enabling the reading starts a timeout (controlled by the `TRNG_CFG.RDTIMEOUT` bit field). When this timeout expires, reading is disabled and the offered data is acknowledged so that it is not offered again. The host processor must set this timeout such that there is enough time to read the output registers and perform a normal data acknowledge (which aborts the timeout).

Test Mode

In addition to the test circuitry that operates during normal operation, the TRNG has a test mode that allows further diagnosis when errors occur. In test mode, programs have access to the main LFSR through the `TRNG_LFSR_L`, `TRNG_LFSR_M`, and `TRNG_LFSR_H` registers. Programs can also control the finite state machine sample counter using the `TRNG_CNT` register.

In test mode, the TRNG can be configured to test individual or a chosen set of FROs. It can also be configured to feed test patterns to the delay chain.

TRNG Data Transfer Modes

The host processor reads four 32-bit registers to access the random numbers. Once the registers are read and the data ready interrupt has been acknowledged, the TRNG moves more data from the internal buffer to the output registers (`TRNG_OUTPUT[n]`).

TRNG Event Control

There are eight events that the TRNG generates. The events are common error events from the run-time testing of the TRNG. While the TRNG is operating and generating random bits, test circuitry is also running statistical tests. The tests determine if the sources of the random bits have started to fail and are not truly producing random bits. There is also a single event to signal when data is ready in the output registers.

These events are captured in the `TRNG_STAT` register and can generate a single interrupt in the Public Key Interrupt Controller (PKIC). The individual events can be masked so the interrupt is not triggered. This configuration uses the `TRNG_CTL` register. The event and associated interrupt can be acknowledged in the `TRNG_INTACK` register.

The ***TRNG Interrupt Signals*** table lists all events or interrupts that the TRNG can generate.

TRNG Interrupt Signals

The TRNG provides a total of eight interrupts multiplexed into one output.

Table 47-3: TRNG Interrupt Signals

Interrupt	Name	Description
0	RDY	Ready. Random number is ready to be read from registers

Table 47-3: TRNG Interrupt Signals (Continued)

Interrupt	Name	Description
1	SHDNOVR	Shutdown Overflow. The number of FRO's automatically shut down due to failures or errors have gone above the threshold specified in `TRNG_ALMCNT.SHDNTHRESH`.
2	STUCKOUT	Stuck Out. Logic circuitry around the output registers has detected the same output has been provided twice.
3	NOISEFAIL	Noise Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 48 identical bits, which is considered a noise source failure.
4	RUNFAIL	Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value for at least one of the `TRNG_RUN[n]` counters after checking 20,000 bits.
5	LRUNFAIL	Long Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 34 identical bits.
6	PKRFAIL	Poker Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 `TRNG_POKER[n]` counters or an out-of-bounds sum of squares values after checking 20,000 bits.
7	MBITFAIL	Monobit Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds number of 1's after checking 20,000 bits.

ADSP-SC58x TRNG Register Descriptions

True Random Number Generator (TRNG) contains the following registers.

Table 47-4: ADSP-SC58x TRNG Register List

Name	Description
`TRNG_ALMCNT`	TRNG Alarm Counter Register
`TRNG_ALMMSK`	TRNG Alarm Mask Register
`TRNG_ALMSTP`	TRNG Alarm Stop Register
`TRNG_CFG`	TRNG Configuration Register
`TRNG_CNT`	Counter Register
`TRNG_CTL`	TRNG Control Register
`TRNG_FRODETUNE`	TRNG FRO De-tune Register
`TRNG_FROEN`	TRNG FRO Enable Register
`TRNG_INPUT[n]`	TRNG Input Registers
`TRNG_INTACK`	TRNG Interrupt Acknowledge Register
`TRNG_LFSR_H`	TRNG LFSR Access Register
`TRNG_LFSR_L`	TRNG LFSR Access Register

Table 47-4: ADSP-SC58x TRNG Register List (Continued)

Name	Description
TRNG_LFSR_M	TRNG LFSR Access Register
TRNG_MONOBITCNT	TRNG Monobit Test Result Register
TRNG_OUTPUT[n]	TRNG Output Registers
TRNG_POKER[n]	TRNG Poker Test Result Registers
TRNG_RUNCNT	TRNG Run Count Registers
TRNG_RUN[n]	TRNG Run Test State and Result Registers
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register

TRNG Alarm Counter Register

The TRNG_ALMCNT register, together with the TRNG_ALMMSK and TRNG_ALMSTP registers, can be used by the host processor to determine if the FRO/sample cycle locking is a problem. Note that incidental alarm events are expected to occur during normal operation. This register also controls the way the monobit test and poker test circuits operate (using the standard 20,000 bit blocks or running continuously).

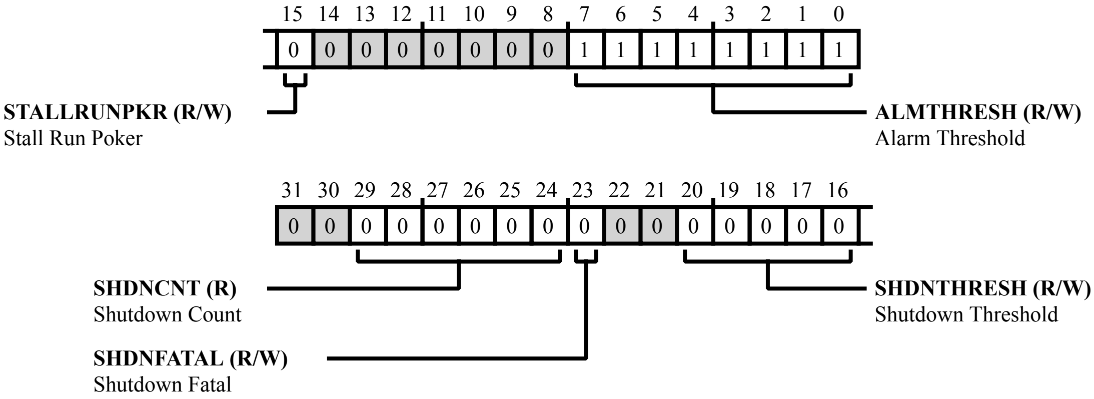

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
STALLRUNPKR (R/W)
Stall Run Poker
ALMTHRESH (R/W)
Alarm Threshold
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SHDNCNT (R)
Shutdown Count
SHDNTHRESH (R/W)
Shutdown Threshold
SHDNFATAL (R/W)
Shutdown Fatal

Figure 47-2: TRNG_ALMCNT Register Diagram

Table 47-5: TRNG_ALMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	SHDNCNT	Shutdown Count. This read-only field indicates the number of 1 bits in the TRNG_ALMSTP register, the number of FRO's that's been turned off.
23 (R/W)	SHDNFATAL	Shutdown Fatal. When the `TRNG_ALMCNT.SHDNFATAL` bit field is set, the shutdown overflow (SHDNOVR) interrupt is considered a fatal error requiring taking the complete TRNG engine off-line.
20:16 (R/W)	SHDNTHRESH	Shutdown Threshold. The `TRNG_ALMCNT.SHDNTHRESH` bit field provides the threshold setting for generating the shutdown overflow (SHDNOVR) interrupt, which is activated when the shutdown count (`TRNG_ALMCNT.SHDNCNT`) value in this register exceeds the threshold value set here.
15 (R/W)	STALLRUNPKR	Stall Run Poker. When the `TRNG_ALMCNT.STALLRUNPKR` bit is set, stalls the Monobit Test, Run Test and Poker Test circuits when either the `TRNG_STAT.MBITFAIL`, `TRNG_STAT.RUNFAIL` or `TRNG_STAT.PKRFAIL` bits =1. This allows inspection of the state of the result counters (which would otherwise be reset immediately for the next 20,000 bits block to test).

Table 47-5: TRNG_ALMCNT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	ALMTHRESH	Alarm Threshold. The `TRNG_ALMCNT.ALMTHRESH` bit field sets the alarm detection threshold for the repeating pattern detectors on each FRO. A FRO alarm event is declared when a repeating pattern (of up to four samples length) is detected continuously for the number of samples defined by this fields value. Reset value 255 (decimal) should keep the number of alarm events to a manageable level.

TRNG Alarm Mask Register

A set bit (=1) in the `TRNG_ALMMSK` register signifies an alarm event and is used by the host processor to determine which of the individual FROs generated the alarm. If a bit in this register is set, the corresponding bit in the `TRNG_ALMSTP` register is set and the FRO is turned off by clearing the corresponding bit in the `TRNG_FROEN` register. If a bit is not set, the FRO restarts automatically to try to break sample cycle locking that could have caused the alarm event.

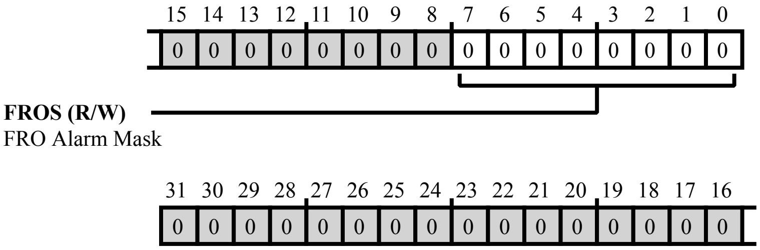

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FROS (R/W)
FRO Alarm Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 47-3: TRNG_ALMMSK Register Diagram

Table 47-6: TRNG_ALMMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Mask. The `TRNG_ALMMSK.FROS` bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced an alarm event.

TRNG Alarm Stop Register

The `TRNG_ALMSTP` register is used by the host processor to determine which of the individual FROs generated more than one alarm event in quick succession. If a FRO generates an alarm event while a previous event is still logged in the `TRNG_ALMMSK` register, the corresponding bit in this register is set (=1) and the FRO is turned off by clearing (=0) the corresponding bit in the `TRNG_FROEN` register. The `TRNG_ALMCNT.SHDNCNT` bit field keeps track of the number of bits that are set in this register.

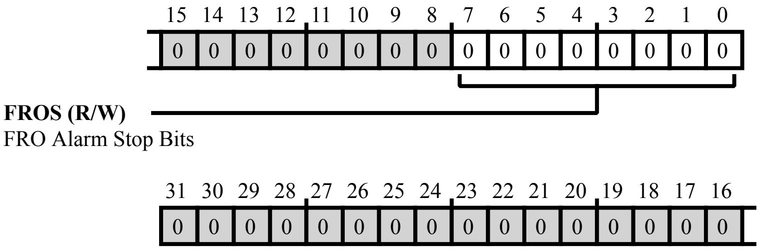

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FROS (R/W)
FRO Alarm Stop Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 47-4: TRNG_ALMSTP Register Diagram

Table 47-7: TRNG_ALMSTP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Stop Bits. The `TRNG_ALMSTP.FROS` bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced more than one alarm event in quick succession and has been turned off. A 1 in this field forces the corresponding bit in the `TRNG_FROEN` register to 0.

TRNG Configuration Register

The `TRNG_CFG` register holds the lower and upper limits of the samples taken from the FROs in order to refill the random data buffer. This register also holds the time out value used for secure reading mode.

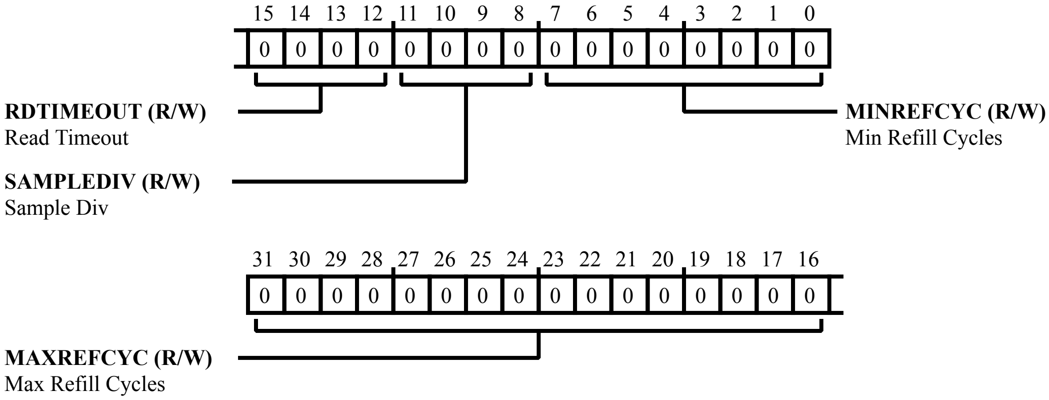

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDTIMEOUT (R/W)
Read Timeout
SAMPLEDIV (R/W)
Sample Div
MINREFCYC (R/W)
Min Refill Cycles
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MAXREFCYC (R/W)
Max Refill Cycles

Figure 47-5: TRNG_CFG Register Diagram

Table 47-8: TRNG_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	MAXREFCYC	Max Refill Cycles. The `TRNG_CFG.MAXREFCYC` bit field determines the maximum number of samples (between 2^8 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number. If the written value of this field is zero, the number of samples is 2^{24}, otherwise the number of samples equals the written value times 2^8. This field can only be modified while the `TRNG_CTL.TRNGEN` bit =0.
15:12 (R/W)	RDTIMEOUT	Read Timeout. The `TRNG_CFG.RDTIMEOUT` bit field controls the Secure Reading Mode. When this field is 0, Secure Reading Mode is disabled. Values in the range 115 enable Secure Reading and set a read gate closure timeout of approximately (read_timeout + 1) x 16 clock input cycles. This field can only be modified while the `TRNG_CTL.TRNGEN` bit =0.
11:8 (R/W)	SAMPLEDIV	Sample Div. The `TRNG_CFG.SAMPLEDIV` bit field directly controls the number of input cycles between samples taken from the FROs. The default value 0 indicates that samples are taken every cycle, maximum value 15 (decimal) takes one sample every 16 cycles. This field must be set to a value such that the slowest FRO (even under worst-case conditions) has a cycle time less than twice the sample period. The default configuration of the FROs allows this field to remain 0. This field can only be modified while `TRNG_CTL.TRNGEN`=0.

Table 47-8: TRNG_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MINREFCYC	Min Refill Cycles. The `TRNG_CFG.MINREFCYC` bit field determines the minimum number of samples (between 2^6 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number. If the value of this field is zero, the number of samples is fixed to the value determined by the maximum refill cycles (`TRNG_CFG.MAXREFCYC`) field, otherwise the minimum number of samples equals the written value times 64 (which can be up to 2^{14}). The number of samples defined here cannot be higher than the number defined by the `TRNG_CFG.MAXREFCYC` field (i.e. that field takes precedence). This field can only be modified while the `TRNG_CTL.TRNGEN` bit =0.

Counter Register

The `TRNG_CNT` register is used to access the main control Finite State Machine's (FSM) sample counter while the `TRNG_CTL.TSTMODE` bit =1.

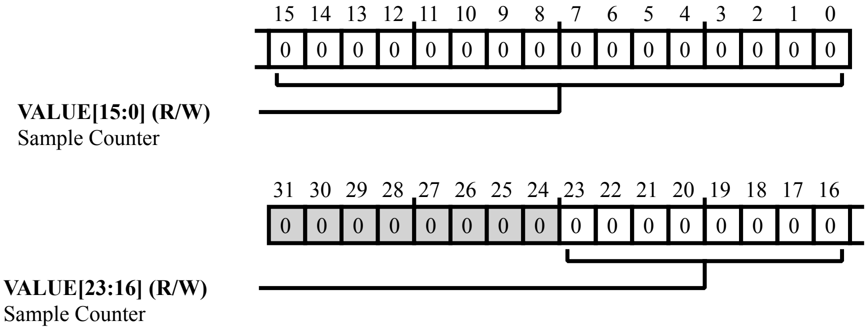

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Sample Counter
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[23:16] (R/W)
Sample Counter

Figure 47-6: TRNG_CNT Register Diagram

Table 47-9: TRNG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Sample Counter. The `TRNG_CNT.VALUE` bit field is the sample counter used by control finite state machine. This counter can only be accessed when the `TRNG_CTL.TSTMODE` bit =1.

TRNG Control Register

The `TRNG_CTL` register must be written to start accumulating entropy before random numbers can be generated. In most cases, the `TRNG_CFG` register must also be written prior to writing the `TRNG_CTL` register. To enable the TRNG, set the `TRNG_CTL.TRNGEN` bit. This register also controls post-processing (if available). Note that when the `TRNG_CTL.TRNGEN` bit =1, the start up cycles field (`TRNG_CTL.STARTUPCYC`) and the post-processing enable bit (if available) are locked. Any writes to these fields are ignored.

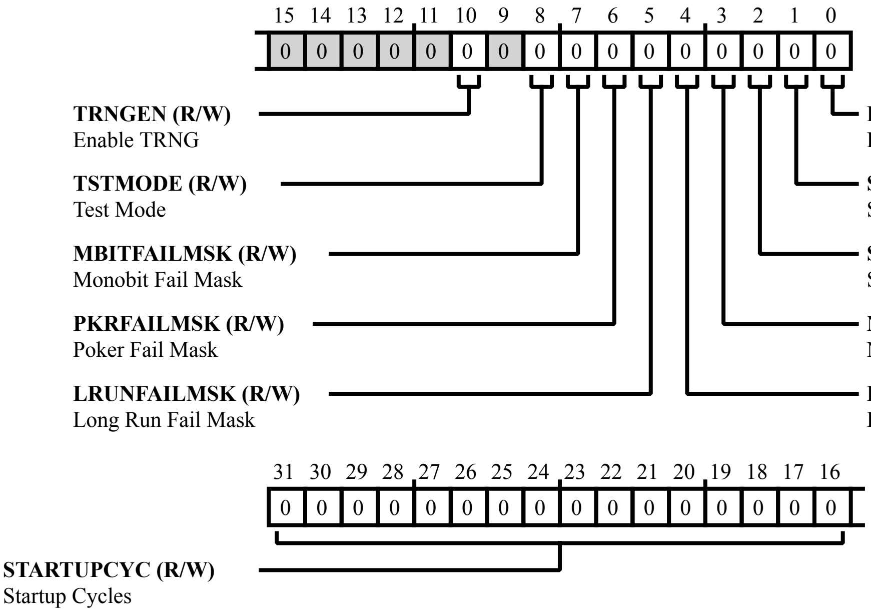

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRNGEN (R/W)
Enable TRNG
TSTMODE (R/W)
Test Mode
MBITFAILMSK (R/W)
Monobit Fail Mask
PKRFAILMSK (R/W)
Poker Fail Mask
LRUNFAILMSK (R/W)
Long Run Fail Mask
RDYMSK (R/W)
Ready Mask
SHDNOVRMSK (R/W)
Shutdown Overflow Mask
STUCKOUTMSK (R/W)
Stuck Out Mask
NOISEFAILMSK (R/W)
Noise Fail Mask
RUNFAILMSK (R/W)
Run Fail Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STARTUPCYC (R/W)
Startup Cycles

Figure 47-7: TRNG_CTL Register Diagram

Table 47-10: TRNG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	STARTUPCYC	Startup Cycles. The `TRNG_CTL.STARTUPCYC` bit field determines the number of samples (between 2^8 and 2^{24}) taken to gather entropy from the FROs during startup. If the written value of this field is zero, the number of samples is 2^{24}, otherwise the number of samples equals the written value times 2^8 . This field can only be written when `TRNG_CTL.TRNGEN`=0 before the write.
10 (R/W)	TRNGEN	Enable TRNG. Setting the `TRNG_CTL.TRNGEN` bit to 1 starts the TRNG, gathering entropy from the FROs for the number of samples determined by the value in the `TRNG_CTL.STARTUPCYC` (Startup Cycles) field. Resetting this bit to 0 forces all TRNG logic back into the idle state immediately. Resetting this bit to 0 also performs the Un-instantiate operation, clearing all internal post-processor registers.

Table 47-10: TRNG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	TSTMODE	Test Mode. When the `TRNG_CTL.TSTMODE` bit is set, access is enabled to the `TRNG_CNT` and `TRNG_LFSR_L`, `TRNG_LFSR_M` and `TRNG_LFSR_H` registers (the latter are cleared before enabling access) and sets the `TRNG_STAT.NEEDCLK` bit for testing purposes. This bit must be set to 1 before various test modes in the `TRNG_TEST` register can be enabled.
7 (R/W)	MBITFAILMSK	Monobit Fail Mask. When the `TRNG_CTL.MBITFAILMSK` bit is set, this mask allows the `TRNG_STAT.MBITFAIL` bit to activate the (active HIGH) interrupt output.
6 (R/W)	PKRFAILMSK	Poker Fail Mask. When the `TRNG_CTL.PKRFAILMSK` bit is set, this mask allows the `TRNG_STAT.PKRFAIL` bit to activate the (active HIGH) interrupt output.
5 (R/W)	LRUNFAILMSK	Long Run Fail Mask. When the `TRNG_CTL.LRUNFAILMSK` bit is set, this mask allows the `TRNG_STAT.LRUNFAIL` bit to activate the (active HIGH) interrupt output.
4 (R/W)	RUNFAILMSK	Run Fail Mask. When the `TRNG_CTL.RUNFAILMSK` bit is set, this mask allows the `TRNG_STAT.RUNFAIL` bit to activate the (active HIGH) interrupt output.
3 (R/W)	NOISEFAILMSK	Noise Fail Mask. When the `TRNG_CTL.NOISEFAILMSK` bit is set, this mask allows the `TRNG_STAT.NOISEFAIL` bit to activate the (active HIGH) interrupt output.
2 (R/W)	STUCKOUTMSK	Stuck Out Mask. When the `TRNG_CTL.STUCKOUTMSK` bit is set, this mask allows the `TRNG_STAT.STUCKOUT` bit to activate the (active HIGH) interrupt output.
1 (R/W)	SHDNOVRMSK	Shutdown Overflow Mask. When the `TRNG_CTL.SHDNOVRMSK` bit is set, this mask allows the `TRNG_STAT.SHDNOVR` bit to activate the (active HIGH) interrupt output.
0 (R/W)	RDYMSK	Ready Mask. When the `TRNG_CTL.RDYMSK` bit is set, this mask allows the `TRNG_STAT.RDY` bit to activate the (active HIGH) interrupt output.

TRNG FRO De-tune Register

The TRNG_FRODETUNE register is used by the host processor to change the frequencies of individual FROs. This can reduce the number of alarm events generated by a specific FRO.

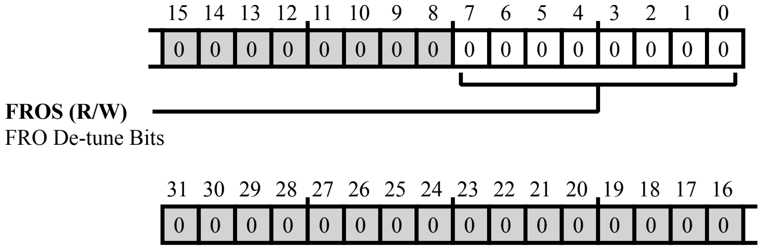

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FROS (R/W)
FRO De-tune Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 47-8: TRNG_FRODETUNE Register Diagram

Table 47-11: TRNG_FRODETUNE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO De-tune Bits. The TRNG_FRODETUNE.FROS bits De-tune the FROs. A 1 in bit [n] lets FRO n run approximately 5% faster. The value of one of these bits may only be changed while the corresponding FRO is turned off (by temporarily writing a 0 in the corresponding bit of the TRNG_FROEN register).

TRNG FRO Enable Register

The `TRNG_FROEN` register can be used by the host processor to enable and disable FROs individually. Only enabled FROs contribute to entropy generation, but require power to do so. Disabled FROs cannot generate alarm events.

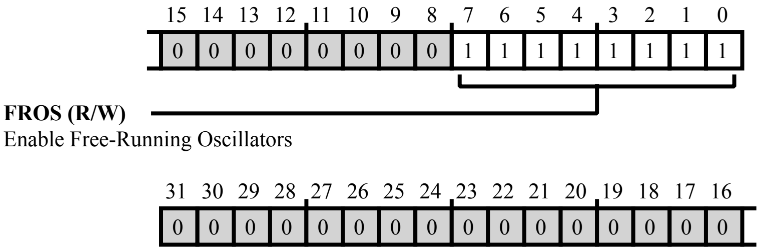

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
FROS (R/W)
Enable Free-Running Oscillators
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 47-9: TRNG_FROEN Register Diagram

Table 47-12: TRNG_FROEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	Enable Free-Running Oscillators. The `TRNG_FROEN.FROS` bits are the enables for the individual FROs. A 1 in bit [n] enables FRO n. The default state is all ones to enable all FROs after power-up. Note that the FROs are not actually started up before the `TRNG_CTL.TRNGEN` bit is set to 1. These bits are automatically forced to 0 (and cannot be written to 1) when the corresponding bit in the `TRNG_ALMSTP` register has value 1.

TRNG Input Registers

The `TRNG_INPUT[n]` registers are used as input for post-processor testing (if post processing is available) and as input for Monobit Test, Run Test and Poker Test functionality tests (TRNG_INPUT0 only). They share their addresses with the corresponding `TRNG_OUTPUT[n]` registers. The least significant word is contained in the TRNG_INPUT0 register.

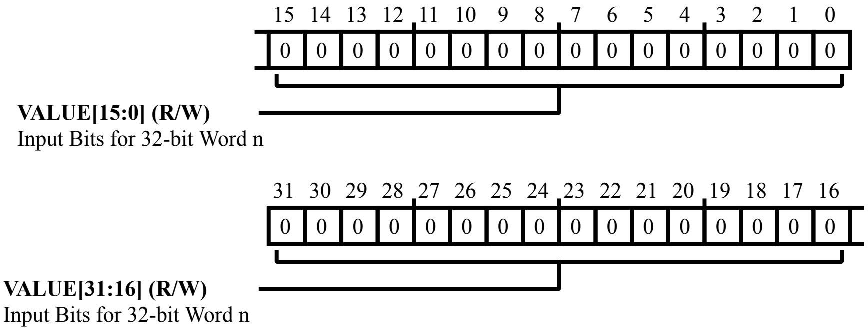

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Input Bits for 32-bit Word n
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Input Bits for 32-bit Word n

Figure 47-10: TRNG_INPUT[n] Register Diagram

Table 47-13: TRNG_INPUT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Input Bits for 32-bit Word n. The `TRNG_INPUT[n].VALUE` bit field is used to hold 32-bits of the 64-bit word of test data for 3-DES post-processor (if available). Or, the `TRNG_INPUT[n].VALUE` bit field is used to hold 32-bit data word for Run Test and Poker Test circuits self test. Can only be written to when the `TRNG_STAT.TSTRDY` bit =1.

TRNG Interrupt Acknowledge Register

The `TRNG_INTACK` register is written to acknowledge interrupts indicated in bits [7:0] of the `TRNG_STAT` register. Writing a 1 to any of the bits [7:2] has side effects in resetting various parts of the TRNG core logic which can also be used even if no interrupts are actually active.

When acknowledging the interrupts, these bits are write '1' to clear the associated bit in TRNG_STAT register. The bit in this register will also automatically be reset to zero.

When secure reading mode is enabled, write bits [7:0] of this register with zeros to enable TRNG data reads from the `TRNG_OUTPUT[n]` registers. Writing bits [7:0] also starts the (configurable) timeout counter that automatically acknowledges the TRNG data (and disables reads) if the `TRNG_INTACK.RDY` bit is not written with a 1 within that timeout period.

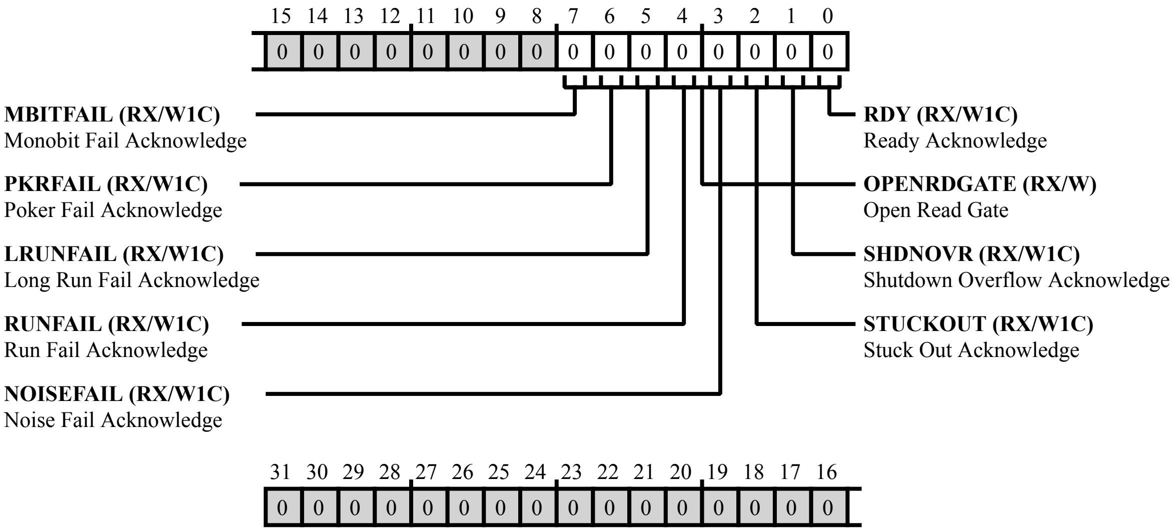

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MBITFAIL (RX/W1C)
Monobit Fail Acknowledge
PKRFAIL (RX/W1C)
Poker Fail Acknowledge
LRUNFAIL (RX/W1C)
Long Run Fail Acknowledge
RUNFAIL (RX/W1C)
Run Fail Acknowledge
NOISEFAIL (RX/W1C)
Noise Fail Acknowledge
RDY (RX/W1C)
Ready Acknowledge
OPENRDGATE (RX/W)
Open Read Gate
SHDNOVR (RX/W1C)
Shutdown Overflow Acknowledge
STUCKOUT (RX/W1C)
Stuck Out Acknowledge
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 47-11: TRNG_INTACK Register Diagram

Table 47-14: TRNG_INTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (RX/W1C)	MBITFAIL	Monobit Fail Acknowledge. Set the `TRNG_INTACK.MBITFAIL` bit to acknowledge the Monobit Fail Interrupt. This also resets all counter and state bits in the `TRNG_RUN[n]`, `TRNG_MONOBITCNT` and the `TRNG_POKER[n]` registers (except for the `TRNG_RUNCNT.LENMAX` field).
6 (RX/W1C)	PKRFAIL	Poker Fail Acknowledge. Set the `TRNG_INTACK.PKRFAIL` bit to acknowledge the Poker Fail Interrupt. This also resets all counter and state bits in the `TRNG_RUN[n]`, `TRNG_MONOBITCNT` and the `TRNG_POKER[n]` registers (except for the `TRNG_RUNCNT.LENMAX` field).

Table 47-14: TRNG_INTACK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (RX/W1C)	LRUNFAIL	Long Run Fail Acknowledge. Set the `TRNG_INTACK.LRUNFAIL` bit to acknowledge the Long Run Fail Interrupt. Also clears the `TRNG_RUNCNT.LENMAX` field.
4 (RX/W1C)	RUNFAIL	Run Fail Acknowledge. Set the `TRNG_INTACK.RUNFAIL` bit to acknowledge the Run Fail Interrupt. Also resets all counter and state bits in the `TRNG_RUN[n]`, `TRNG_MONOBITCNT` and the `TRNG_POKER[n]` registers (except for the `TRNG_RUNCNT.LENMAX` field).
3 (RX/W1C)	NOISEFAIL	Noise Fail Acknowledge. Set the `TRNG_INTACK.NOISEFAIL` bit to acknowledge the Noise Fail Interrupt. Setting this bit also clears the `TRNG_RUNCNT.LENMAX` (Run Length Max) field, the random data buffer, the `TRNG_OUTPUT[n]` registers and the `TRNG_STAT.RDY` bit.
2 (RX/W1C)	STUCKOUT	Stuck Out Acknowledge. Set the `TRNG_INTACK.STUCKOUT` bit to acknowledge the Stuck Out Interrupt. Setting this bit also clears the random data buffer, the `TRNG_OUTPUT[n]` registers and the `TRNG_STAT.RDY` bit.
1 (RX/W1C)	SHDNOVR	Shutdown Overflow Acknowledge. Set the `TRNG_INTACK.SHDNOVR` bit to acknowledge the Shutdown Overflow Interrupt.
0 (RX/W1C)	RDY	Ready Acknowledge. The `TRNG_INTACK.RDY` bit allows a new number (if it is ready in the random data buffer), to directly move into the result register. Once done, the `TRNG_STAT.RDY` bit is reset, after at most size clock cycles.
7:0 (RX/W)	OPENRDGATE	Open Read Gate. In Secure Reading Mode, the `TRNG_INTACK.OPENRDGATE` bit writes an all zeros value to bits [7:0] to enable reading of TRNG data from the `TRNG_OUTPUT[n]` registers. This starts the timeout counter that automatically acknowledges the TRNG data (and disables reading) if the `TRNG_INTACK.RDY` bit is not written with a 1 within that timeout period.

TRNG LFSR Access Register

The `TRNG_LFSR_H` register is used to access bits [80:64] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE` =1).

For security reasons, the LFSR contents are zeroed before enabling access.

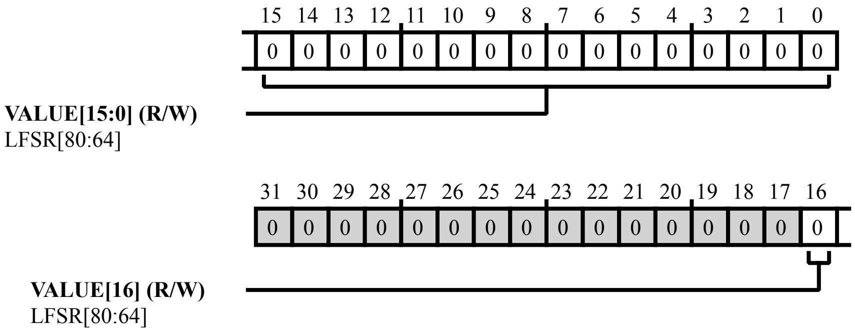

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
LFSR[80:64]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[16] (R/W)
LFSR[80:64]

Figure 47-12: TRNG_LFSR_H Register Diagram

Table 47-15: TRNG_LFSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/W)	VALUE	LFSR[80:64]. The `TRNG_LFSR_H.VALUE` bit field contains bits [80:64] of the main entropy accumulation LFSR. This field can only be accessed when the `TRNG_CTL.TSTMODE` bit =1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The `TRNG_LFSR_L` register is used to access bits [31:0] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE` =1).

For security reasons, the LFSR contents are zeroed before enabling access.

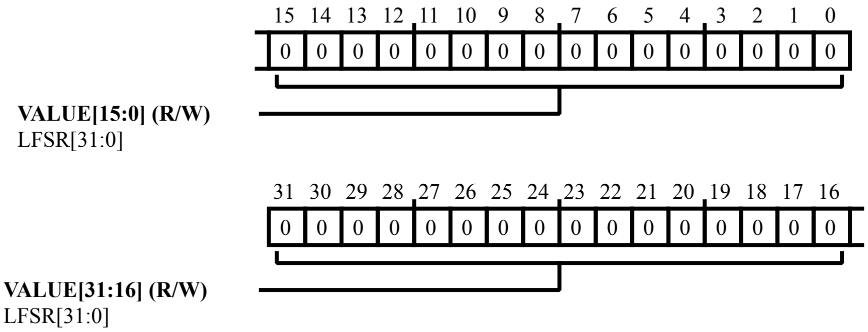

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
LFSR[31:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
LFSR[31:0]

Figure 47-13: TRNG_LFSR_L Register Diagram

Table 47-16: TRNG_LFSR_L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[31:0]. The `TRNG_LFSR_L.VALUE` bit field contains bits [31:0] of the main entropy accumulation LFSR. This field can only be accessed when the `TRNG_CTL.TSTMODE` bit =1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The `TRNG_LFSR_M` register is used to access bits [63:32] of the main entropy accumulation LFSR while in test mode (`TRNG_CTL.TSTMODE` =1).

For security reasons, the LFSR contents are zeroed before enabling access.

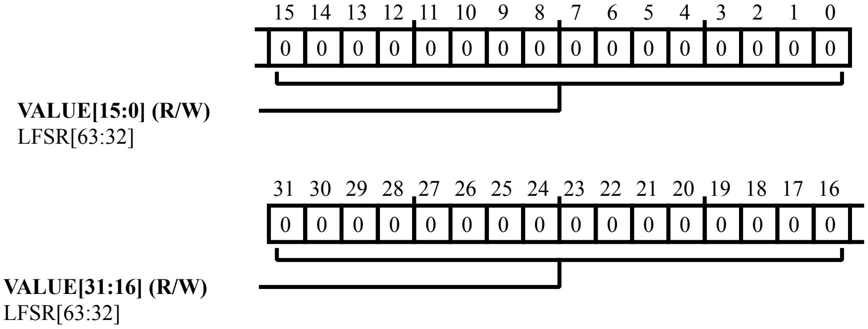

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
LFSR[63:32]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
LFSR[63:32]

Figure 47-14: TRNG_LFSR_M Register Diagram

Table 47-17: TRNG_LFSR_M Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[63:32]. The `TRNG_LFSR_M.VALUE` bit field contains bits [63:32] of the main entropy accumulation LFSR. This field can only be accessed when the `TRNG_CTL.TSTMODE` bit =1. Contents are cleared (=0) before access is enabled.

TRNG Monobit Test Result Register

The `TRNG_MONOBITCNT` register accesses the counter used perform a Monobit Test as specified by the AIS-31 standard (test T1, ref 4). This test is performed on blocks of 20,000 bits (in parallel to the run test and Poker Test).

Note: Immediately after performing the actual Monobit Test at the end of the 20,000 bits block, the counter is used to accumulate the Poker Test results. As a result, the actual Monobit Test count result value can only be read in the `TRNG_MONOBITCNT` register if the test fails and the stall run Poker (`TRNG_ALMCNT.STALLRUNPKR`) bit =1.

The monobit test result register is read-only; writing it accesses the registers mapped at the same address. The counter in this register is reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), run fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

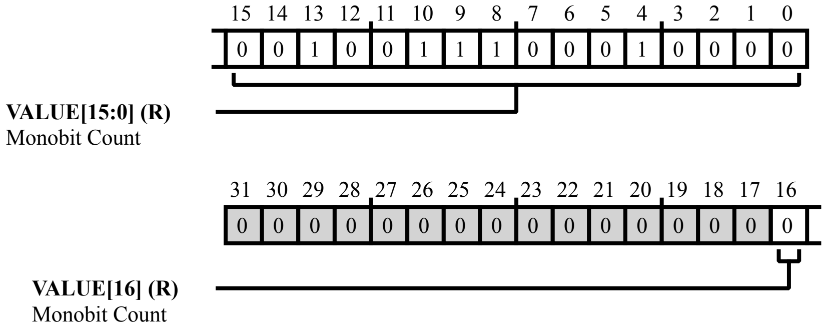


Figure 47-15: TRNG_MONOBITCNT Register Diagram

Table 47-18: TRNG_MONOBITCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/NW)	VALUE	Monobit Count. The `TRNG_MONOBITCNT.VALUE` bit field is the up/down counter which monitors 1 and 0 bits. After 20,000 bits, this counter should have a value in the range 9310 through 10690 (inclusive) to pass the Monobit Test. This counter is protected against overflow and underflow.

TRNG Output Registers

The `TRNG_OUTPUT[n]` registers provide read access to the 128-bit random number output. A subset of these registers are also used as output for post-processor testing (if available). They share their addresses with the TRNG_INPUT0 through TRNG_INPUT3 registers. The least significant word is contained in the TRNG_OUTPUT0 register.

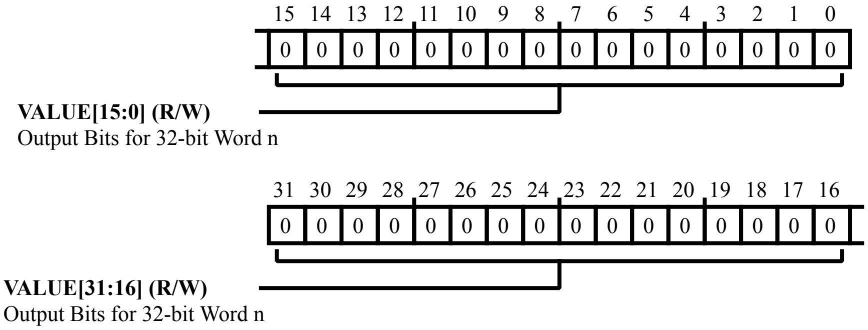

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Output Bits for 32-bit Word n
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Output Bits for 32-bit Word n

Figure 47-16: TRNG_OUTPUT[n] Register Diagram

Table 47-19: TRNG_OUTPUT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Output Bits for 32-bit Word n. The `TRNG_OUTPUT[n].VALUE` bit field is used to holds 32 bits of the 128-bit word of random data. Only valid when the `TRNG_STAT.RDY` bit =1. Alternatively, this register holds the 32-bits of the 42-bit word of result data for 3-DES post-processing testing. Only valid when the `TRNG_STAT.TSTRDY` bit =1.

TRNG Poker Test Result Registers

The `TRNG_POKER[n]` registers are used to access the 16 counters used perform a poker test on blocks of 20,000 bits (in parallel to the monobit and run tests).

Poker test result registers are read-only; writing them accesses the registers mapped at these same addresses. All counters in these registers are reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), run fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

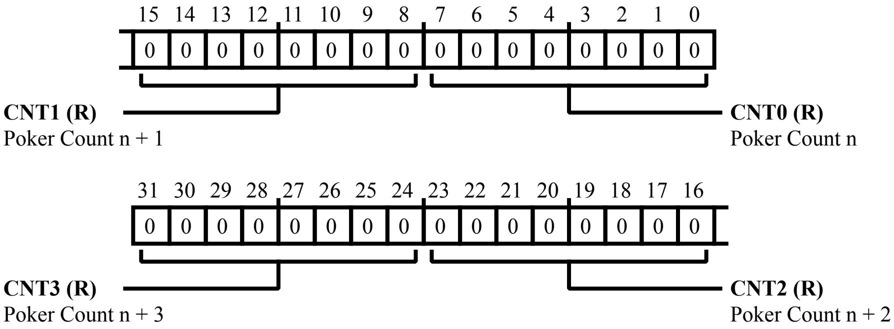

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT1 (R)
Poker Count n + 1
CNT0 (R)
Poker Count n
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CNT3 (R)
Poker Count n + 3
CNT2 (R)
Poker Count n + 2

Figure 47-17: TRNG_POKER[n] Register Diagram

Table 47-20: TRNG_POKER[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	CNT3	Poker Count n + 3. The `TRNG_POKER[n].CNT3` bit field provides the counter for 4-bit value 0x3, 0x7, 0xB, 0xF in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
23:16 (R/NW)	CNT2	Poker Count n + 2. The `TRNG_POKER[n].CNT2` bit field provides the counter for 4-bit value 0x2, 0x6, 0xA, 0xE in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
15:8 (R/NW)	CNT1	Poker Count n + 1. The `TRNG_POKER[n].CNT1` bit field provides the counter for 4-bit value 0x1, 0x5, 0x9, 0xD in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
7:0 (R/NW)	CNT0	Poker Count n. The `TRNG_POKER[n].CNT0` bit field provides the counter for 4-bit value 0x0, 0x4, 0x8, 0xC in the TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.

TRNG Run Count Registers

The TRNG_RUNCNT registers are used to access the 10 counters that perform a run test and long run test as specified by the AIS-31 standard (tests T3 and T4, ref 4). They are also used to perform the noise source failure test proposed in section E.5 of that same standard.

The TRNG_RUNCNT registers are read-only; writing them accesses the other registers which are mapped at the same addresses. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the Monobit Fail acknowledge (`TRNG_INTACK.MBITFAIL`), Run Fail acknowledge (`TRNG_INTACK.RUNFAIL`) or the Poker Fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

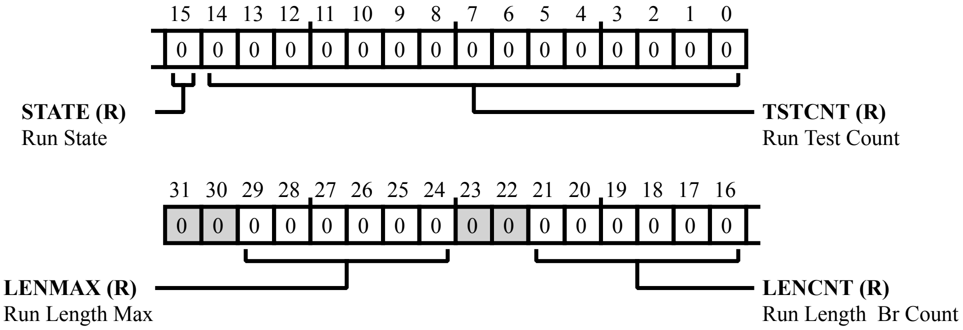


Figure 47-18: TRNG_RUNCNT Register Diagram

Table 47-21: TRNG_RUNCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	LENMAX	Run Length Max. The `TRNG_RUNCNT.LENMAX` bit field configures the maximum run length count value encountered since start of test. This value is reset back to zero when writing a 1 to either the Noise Fail acknowledge (`TRNG_INTACK.NOISEFAIL`) or the Long Run Fail acknowledge (`TRNG_INTACK.LRUNFAIL`) bits.
21:16 (R/NW)	LENCNT	Run Length Br Count. The `TRNG_RUNCNT.LENCNT` bit field configures the counter for the current run of consecutive 0/1 bits; cannot increment past its maximum value of 63.
15 (R/NW)	STATE	Run State. The `TRNG_RUNCNT.STATE` bit field provides th state of bits in the current run.
14:0 (R/NW)	TSTCNT	Run Test Count. The `TRNG_RUNCNT.TSTCNT` bit field configures the block length counter for the run and poker tests - counts up for 20,000 tested bits and then controls testing of the run_X_count_... and poker_count_X counters to contain expected values, after which they - and this counter - are reset for the next block.

TRNG Run Test State and Result Registers

The `TRNG_RUN[n]` registers holds the counts for the associated run bucket for 1's and 0's.

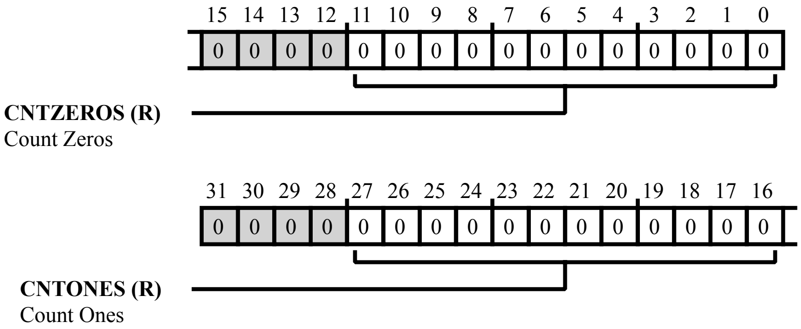


Figure 47-19: TRNG_RUN[n] Register Diagram

Table 47-22: TRNG_RUN[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:16 (R/NW)	CNTONES	Count Ones. In TRNG_RUN1, this counter is for single bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095. In TRNG_RUN2, this counter is for two bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047. In TRNG_RUN3, this counter is for three bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023. In TRNG_RUN4, this counter for four bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511. In TRNG_RUN5, this counter is for five bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255. In TRNG_RUN6, this counter for six and higher bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.

Table 47-22: TRNG_RUN[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/NW)	CNTZEROS	Count Zeros. In TRNG_RUN1, this counter is for single bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095. In TRNG_RUN2, this counter is for two bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047. In TRNG_RUN3, this counter is for three bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023. In TRNG_RUN4, this counter is for four bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511. In TRNG_RUN5, this counter is for five bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255. In TRNG_RUN6, this counter is for six and higher bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.

TRNG Status Register

The TRNG_STAT register provides status results. This register shares the same address as the Interrupt Acknowledge (TRNG_INTACK) register.

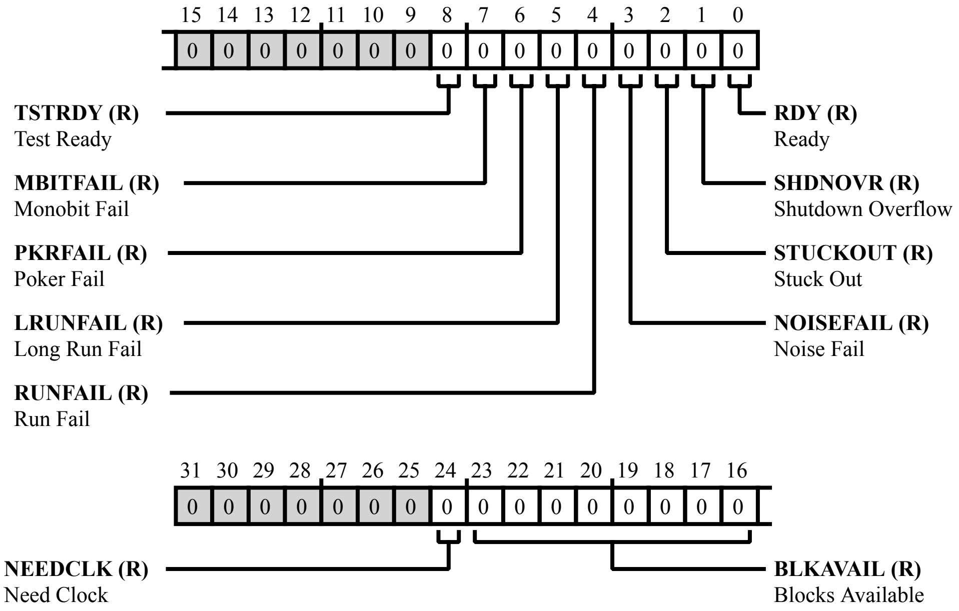

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSTRDY (R)
Test Ready
MBITFAIL (R)
Monobit Fail
PKRFAIL (R)
Poker Fail
LRUNFAIL (R)
Long Run Fail
RUNFAIL (R)
Run Fail
RDY (R)
Ready
SHDNOVR (R)
Shutdown Overflow
STUCKOUT (R)
Stuck Out
NOISEFAIL (R)
Noise Fail
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NEEDCLK (R)
Need Clock
BLKAVAIL (R)
Blocks Available

Figure 47-20: TRNG_STAT Register Diagram

Table 47-23: TRNG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/NW)	NEEDCLK	Need Clock. When the TRNG_STAT.NEEDCLK bit is set, it indicates that the TRNG is busy generating entropy or is in one of its test modes the module clock may not be turned off.
23:16 (R/NW)	BLKAVAIL	Blocks Available. This field indicates the number of 128 bits blocks of random data that are available in the random data buffer. If this value is non-zero, the output registers will be re-filled from the random data buffer immediately after acknowledging the TRNG_STAT.RDY by writing a '1' to TRNG_INTACK.RDY.
8 (R/NW)	TSTRDY	Test Ready. When the TRNG_STAT.TSTRDY bit is set, it indicates that data for known-answer tests on the Monobit Test, Run Test, Poker Test and post-processor functions can be written to the TRNG_INPUT[n] registers. When testing the post-processor, result data can be read from those same registers when this bit has become 1 again (after dropping to 0).

Table 47-23: TRNG_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	MBITFAIL	Monobit Fail. When the `TRNG_STAT.MBITFAIL` bit is set, the Monobit Test logic monitoring data, shifted into the main LFSR, detected an out-of-bounds number of 1s after checking 20,000 bits (test T1 as specified in the AIS-31 standard).
6 (R/NW)	PKRFAIL	Poker Fail. When the `TRNG_STAT.PKRFAIL` bit is set, the Poker Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 Poker Counters or an out of bounds sum of squares value after checking 20,000 bits (test T2 as specified in the AIS-31 standard).
5 (R/NW)	LRUNFAIL	Long Run Fail. When the `TRNG_STAT.LRUNFAIL` bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 34 identical bits (test T4 as specified in the AIS-31 standard).
4 (R/NW)	RUNFAIL	Run Fail. When the `TRNG_STAT.RUNFAIL` bit is set, the Run Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value for at least one of the `TRNG_RUN[n].CNTZEROS` or `TRNG_RUN[n].CNTONES` counters after checking 20,000 bits (test T3 as specified in the AIS-31 standard).
3 (R/NW)	NOISEFAIL	Noise Fail. When the `TRNG_STAT.NOISEFAIL` bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 48 identical bits, which is considered a noise source failure as proposed in section E.5 of the AIS-31 standard.
2 (R/NW)	STUCKOUT	Stuck Out. When the `TRNG_STAT.STUCKOUT` bit is set, the logic around the output data registers detected that the TRNG generates the same value twice in a row.
1 (R/NW)	SHDNOVR	Shutdown Overflow. When the `TRNG_STAT.SHDNOVR` bit is set, the number of FROs shut down after a second error event (the number of 1 bits in the `TRNG_ALMSTP` register) has exceeded the threshold set by the `TRNG_ALMCNT.SHDNTHRESH` bit field.
0 (R/NW)	RDY	Ready. When the `TRNG_STAT.RDY` bit is set, data is available in the TRNG_OUTPUT0 to TRNG_OUTPUT3 registers. If a new number is already available in the random data buffer, that number is directly moved into the result register. In this case the ready status bit is asserted again, after at most six module clock cycles.

TRNG Test Register

The TRNG_TEST register can be used by the host processor to perform a number of tests on the TRNG logic including:

- Register controlled characterization by connecting the tst_fro_clk_out output to a selected FRO clock output
- FRO logic connectivity and error event detection checking by feeding known patterns through the FRO delay line and error event detection circuits
- Direct XOR-ed FRO outputs capture by disabling the main LFSR feedback logic
- Extend the Monobit Test and Poker Test by not resetting the Monobit count and Poker Test X counters after each 20,000 bits block
- Perform known answer tests on the Run Test, Poker Test and post-processor functions.

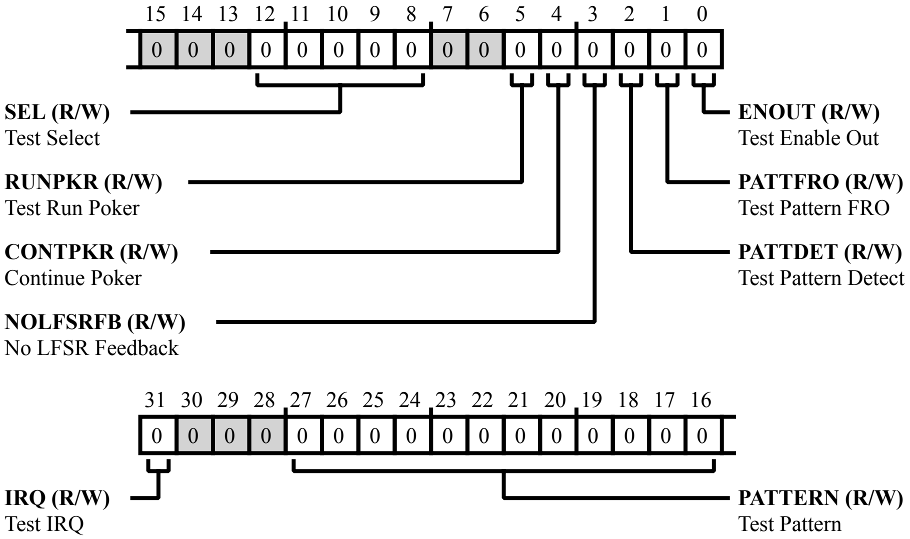

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SEL (R/W)
Test Select
RUNPKR (R/W)
Test Run Poker
CONTPKR (R/W)
Continue Poker
NOLFSRFB (R/W)
No LFSR Feedback
ENOUT (R/W)
Test Enable Out
PATTFRO (R/W)
Test Pattern FRO
PATTDET (R/W)
Test Pattern Detect
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IRQ (R/W)
Test IRQ
PATTERN (R/W)
Test Pattern

Figure 47-21: TRNG_TEST Register Diagram

Table 47-24: TRNG_TEST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	IRQ	Test IRQ. When the TRNG_TEST.IRQ bit is set force irq output HIGH for interrupt signal connectivity testing.	
		0	Do not force IRQ high
		1	Force IRQ output high

Table 47-24: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
27:16 (R/W)	PATTERN	Test Pattern. The `TRNG_TEST.PATTERN` bit field sets up a repeating sequence of bits to be fed into the selected FRO delay chain `TRNG_TEST.PATTFRO` =1 and/or the selected FRO error detection circuit `TRNG_TEST.PATTDET` =1. This field is rotated right over one bit, once every sample period, when either of these control bits is 1. Therefore, bit [16] is the actual pattern bit fed into the test target.
12:8 (R/W)	SEL	Test Select. The `TRNG_TEST.SEL` bit field configures the number of the FRO to be tested, the value should be in the range of 0 to 7.
5 (R/W)	RUNPKR	Test Run Poker. When the `TRNG_TEST.RUNPKR` bit is set, it provides direct access to the inputs of the Monobit, Run and Poker Test circuits (writing input data in chunks of 32 bits to the TRNG_INPUT0 register). While this bit is 1, the TRNG is not allowed to generate entropy but any buffered random data is preserved, to be loaded into the output registers as soon as this bit is reset to 0 again. The `TRNG_STAT.NEEDCLK` bit is forced active while this bit is 1. The Monobit, Run and Poker Test circuits are reset to their initial states on any change of this bit.
4 (R/W)	CONTPKR	Continue Poker. When the `TRNG_TEST.CONTPKR` bit is set, Monobit Test and Poker Test keep running continuously by not resetting the Monobit count (`TRNG_MONOBITCNT`) and poker counters (`TRNG_POKER[n]` register) at the end of each 20,000 bits test block. This bit can only be set to 1 when `TRNG_CTL.TSTMODE` =1. 0 Do not continue poker test 1 Continue poker test
3 (R/W)	NOLFSRFB	No LFSR Feedback. When the `TRNG_TEST.NOLFSRFB` bit is set, it removes XNOR feedback from the main LFSR, converting it into a normal shift register for the XOR-ed outputs of the FROs (shifting data in on the LSB side). A 1 also forces the LFSR to sample continuously. This bit can only be set to 1 when `TRNG_CTL.TSTMODE` =1. 0 Keep XNOR feedback 1 Remove XNOR feedback

Table 47-24: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PATTDET	Test Pattern Detect. When the `TRNG_TEST.PATTDET` bit is set, it repeatedly feeds test pattern (PATTERN) into the error detection circuit of the FRO selected by the test select (`TRNG_TEST.SEL`) field. This bit can only be set to 1 when `TRNG_CTL.TSTMODE` =1.	
		0	Do not repeat feed test pattern
		1	Repeat feed test pattern
1 (R/W)	PATTFRO	Test Pattern FRO. When the `TRNG_TEST.PATTFRO` bit is set, it repeatedly feeds test pattern (PATTERN) into the delay chain of the FRO selected by the test select (`TRNG_TEST.SEL`) field by forcing the corresponding FRO enable (FROEN) output LOW. This bit can only be set to 1 when `TRNG_CTL.TSTMODE` =1.	
		0	Do not repeat feed test pattern
		1	Repeat feed test pattern
0 (R/W)	ENOUT	Test Enable Out. When the `TRNG_TEST.ENOUT` bit is set, it enables the tst_fro_clk_out output, connecting to the FRO selected by the test select (`TRNG_TEST.SEL`) field. This bit can only be set to 1 when `TRNG_CTL.TSTMODE` =1.	
		0	Disable tst_fro_clk_out
		1	Enable tst_fro_clk_out

48 Thermal Monitoring Unit (TMU)

The TMU provides on-chip temperature measurement which is important in applications that have substantial power consumption. The TMU is integrated into the processor die and digital infrastructure using an MMR-based system access to measure the die temperature variations in real-time.

TMU Features

The TMU supports the following features:

- On-chip temperature sensing
- Programmable over-temperature and under-temperature limits
- Programmable conversion rate
- Programmable clock source selection to run the sensor off an independent local clock
- Averaging feature available
- Temperature gain and offset correction options

TMU Functional Description

Following sections provide the functional description of TMU.

ADSP-SC58x TMU Register List

Thermal monitoring unit

Table 48-1: ADSP-SC58x TMU Register List

Name	Description
`TMU_ALRT_LIM_HI`	Alert High Limit Register
`TMU_ALRT_LIM_LO`	Alert Low Limit Register
`TMU_AVG`	Averaging Register
`TMU_CTL`	TMU Control Register

Table 48-1: ADSP-SC58x TMU Register List (Continued)

Name	Description
TMU_FLT_LIM_HI	Fault High Limit Register
TMU_FLT_LIM_LO	Fault Low Limit Register
TMU_GAIN	Gain Value Register
TMU_IMSK	Interrupt Mask Register
TMU_OFFSET	Offset Register
TMU_STAT	Status Register
TMU_TEMP	Temperature Value Register

ADSP-SC58x TMU Interrupt List

Table 48-2: ADSP-SC58x TMU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
6	TMU0_FAULT	TMU0 Fault		

ADSP-SC58x TMU Trigger List

Table 48-3: ADSP-SC58x TMU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
131	TMU0_FAULT	TMU0 Fault Event	

Table 48-4: ADSP-SC58x TMU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

TMU Definitions

The following definitions are helpful when using the TMU module.

Thermal Diode

A special type of diode whose electrical properties change with the temperature in a defined way.

DTM

Dynamic thermal management is a set of techniques that adapt the run-time behavior of a processor to achieve the highest performance under thermal constraints.

TMU Block Diagram

The *Thermal Monitoring Unit Block Diagram* shows the main block inside TMU.

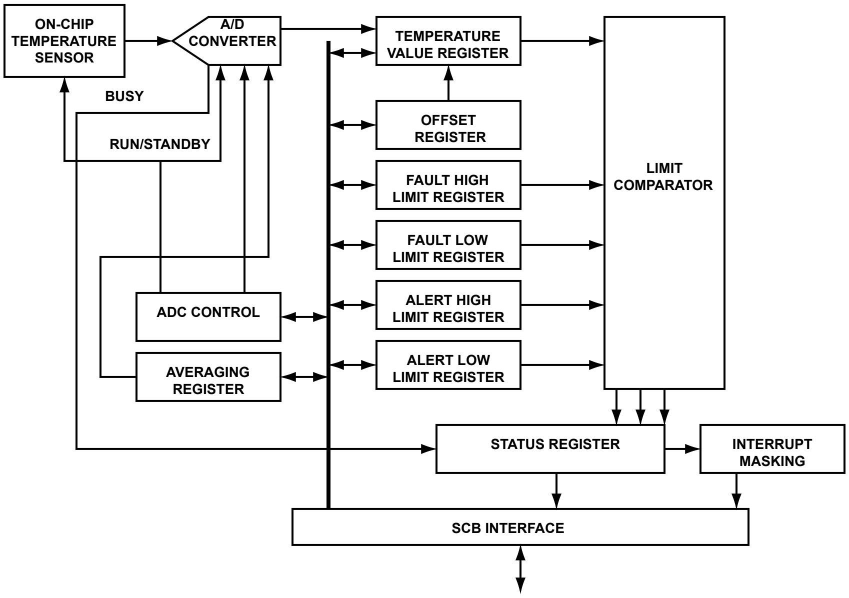

ON-CHIP TEMPERATURE SENSOR
A/D CONVERTER
TEMPERATURE VALUE REGISTER
BUSY
RUN/STANDBY
OFFSET REGISTER
FAULT HIGH LIMIT REGISTER
FAULT LOW LIMIT REGISTER
ALERT HIGH LIMIT REGISTER
ALERT LOW LIMIT REGISTER
LIMIT COMPARATOR
ADC CONTROL
AVERAGING REGISTER
STATUS REGISTER
INTERRUPT MASKING
SCB INTERFACE

Figure 48-1: Thermal Monitoring Unit Block Diagram

TMU Architectural Concepts

The following sections provide information on the architecture and system integration of the TMU.

System Integration

Dynamic thermal management (DTM) techniques adapt the run-time behavior of a processor to achieve the highest performance under thermal constraints. One of the most important aspects of DTM is to capture the run-time variations in the temperature caused by power consumption variations due to workload changes.

The accuracy of thermal measurements directly affects the efficiency of thermal management as well as the performance of the processor. Temperature estimations lower or higher than the actual temperature may cause late or early activation of DTM techniques.

- Late activation of DTM can result in degraded reliability because the temperature may exceed the designated thresholds.
- Early activation of DTM can have significant impact on performance.

The TMU is an analog thermal sensor that consists of a temperature-sensing diode, a calibrated reference current source, and a current comparator. The sensor placement error is one of the most important sources of inaccuracy in values obtained from thermal sensors.

The TMU module provides the temperature monitoring capability to the chip and implements thermal management in the end system. It provides many features which ensure the minimum load on the software and also minimal or no external components for a flexible temperature monitoring system.

Digital Thermometer

The TMU functions as a digital thermometer with an MMR-based system access. The on-die temperature value is measured and digitized through an A/D converter. The temperature value is stored and updated periodically in an MMR register. The TMU can be configured to generate an interrupt as it crosses the upper temperature limit (`TMU_FLT_LIM_HI` register). A thermal event is also routed to the interrupt port of the sensor which is routed to the SEC, to ensure that core intervention is not required in the event of overheating.

Temperature Sensor Averaging

A temperature sensor averaging feature is to enhance the accuracy of the temperature measurements. To enable the temperature sensor averaging feature, the `TMU_AVG` bits must be enabled in the control register after power-up. In this mode, the averaging reduces the effect of noise on the temperature result. The temperature is measured each time a conversion is performed and a moving average method is used to determine the result in the temperature value register. The total time to measure a temperature channel is typically 1 ms.

TMU Event Control

The TMU generates different events depending on the state of the TMU temperature measurement and the different thresholds set in thresh hold registers. These events are reported in the `TMU_STAT` register as shown below. It can generate an event for each of following conditions.

- The fault high limit is configured in the `TMU_FLT_LIM_HI` register. The interrupt is generated when the temperature value is greater than or equal to this value.
- The alert high limit is configured in the `TMU_ALRT_LIM_HI` register. The `TMU_STAT.ALRTHI` status bit is set when the temperature value is greater than or equal to this value.
- The fault low limit is configured in the `TMU_FLT_LIM_LO` register. The `TMU_STAT.FLTLO` status bit is set when the temperature value is less than or equal to this value.
- The alert low limit is configured in the `TMU_ALRT_LIM_LO` register. The `TMU_STAT.FLTLO` status bit is set when the temperature value is less than or equal to this value.

Interrupts and status conditions can be masked (disabled) or unmasked (enabled) by setting and clearing bits in the `TMU_IMSK` register.

NOTE: Only `TMU_FLT_LIM_HI` generates an interrupt to core. Other events only result in status change in the `TMU_STAT` register.

Status and Error Signals

When the measured temperature value exceeds the high or low limits that are configured in the `TMU_FLT_LIM_HI`/`TMU_FLT_LIM_LO` and the `TMU_ALRT_LIM_HI`/`TMU_ALRT_LIM_LO` registers, corresponding thermal events are triggered.

The fault and alert events are sent to the core through the status register (`TMU_STAT`). Both the `TMU_STAT.FLTHI` and `TMU_STAT.ALRTHI` bits are sticky bits and are cleared by a W1C operation by the core.

The alert and fault registers can be programmed through the MMR bus interface as shown in the TMU block diagram. A write into the `TMU_FLT_LIM_HI`/`TMU_FLT_LIM_LO` and `TMU_ALRT_LIM_HI`/`TMU_ALRT_LIM_LO` registers or the `TMU_TEMP` register is not allowed when the events are being triggered.

TMU Programming Guidelines

The following section provides basic programming information for the TMU module.

To get the best performance and accuracy from the TMU, intialize the `TMU_GAIN`, `TMU_OFFSET` and `TMU_AVG` registers before enabling the module. Contact Analog Devices, Inc for the current best values to use.

After these registers are programmed, the rest of the TMU initialization can take place. This includes setting up the fault and alert limits and then enabling the TMU.

ADSP-SC58x TMU Register Descriptions

Thermal monitoring unit (TMU) contains the following registers.

Table 48-5: ADSP-SC58x TMU Register List

Name	Description
`TMU_ALRT_LIM_HI`	Alert High Limit Register
`TMU_ALRT_LIM_LO`	Alert Low Limit Register
`TMU_AVG`	Averaging Register
`TMU_CTL`	TMU Control Register
`TMU_FLT_LIM_HI`	Fault High Limit Register
`TMU_FLT_LIM_LO`	Fault Low Limit Register
`TMU_GAIN`	Gain Value Register
`TMU_IMSK`	Interrupt Mask Register
`TMU_OFFSET`	Offset Register
`TMU_STAT`	Status Register
`TMU_TEMP`	Temperature Value Register

Alert High Limit Register

The TMU_ALRT_LIM_HI register sets the temperature alert high limit as an integer value. The value is stored in two's complement format. Asserts TMU_STAT.ALRTHI if the TMU_TEMP value is greater than or equal to TMU_ALRT_LIM_HI. The TMU_ALRT_LIM_HI value should be programmed for value greater than 8'h3C (60 degC).

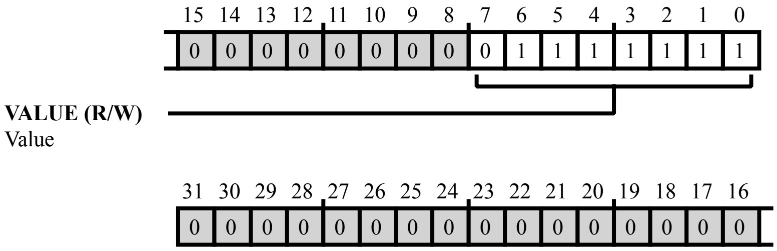

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-2: TMU_ALRT_LIM_HI Register Diagram

Table 48-6: TMU_ALRT_LIM_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Value. The TMU_ALRT_LIM_HI.VALUE bit field configures the alert temperature high limit as an integer value stored in two's complement format. If the temperature value is greater than or equal to the high-limit the TMU_STAT.ALRTHI bit is set. The limit-value should be programmed for value greater than 8h3C (60 degC).

Alert Low Limit Register

The TMU_ALRT_LIM_LO register configures the alert temperature low limit.

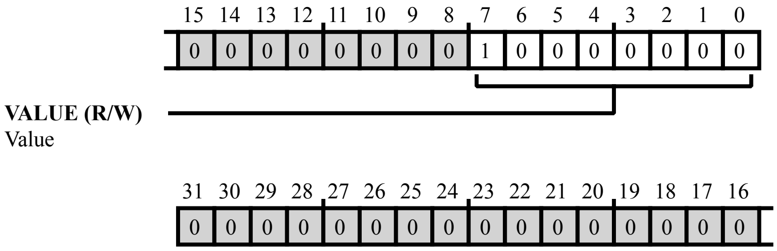

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-3: TMU_ALRT_LIM_LO Register Diagram

Table 48-7: TMU_ALRT_LIM_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Value. The `TMU_ALRT_LIM_LO.VALUE` bit field configures the alert temperature low limit as an integer value stored in two's complement format. If the temperature value is less than or equal to the low-limit the `TMU_STAT.ALRTLO` bit is set.

Averaging Register

The TMU_AVG register enables averaging on the TMU.

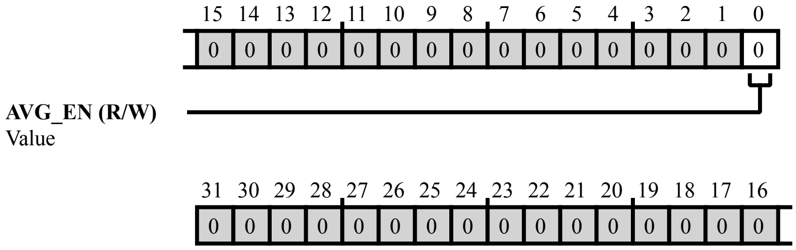

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AVG_EN (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-4: TMU_AVG Register Diagram

Table 48-8: TMU_AVG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	AVG_EN	Value. The `TMU_AVG.AVG_EN` bit enables averaging on the TMU. Averaging is done using the formula (7 x previous_avg_value + current_value)/8. Initially the current_value is taken as previous_avg_value.	
		0	No averaging
		1	Enable averaging

TMU Control Register

The TMU_CTL register contains bits that allow programs to configure and enable the TMU.

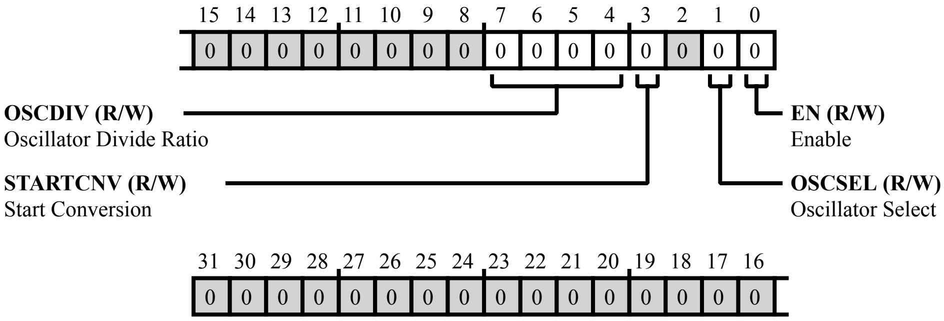

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OSCDIV (R/W)
Oscillator Divide Ratio
STARTCNV (R/W)
Start Conversion
EN (R/W)
Enable
OSCSEL (R/W)
Oscillator Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-5: TMU_CTL Register Diagram

Table 48-9: TMU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	OSCDIV	Oscillator Divide Ratio. The TMU_CTL.OSCDIV bit field configures the division ratio for the SCLK. SCLK is divided by (21 + 4 * TMU_CTL.OSCDIV). By default the SCLK is divided by 21.	
3 (R/W)	STARTCNV	Start Conversion. When the TMU_CTL.STARTCNV bit is set, ADC conversion starts.	
		0	ADC conversion not started
		1	Start ADC conversion
1 (R/W)	OSCSEL	Oscillator Select. The TMU_CTL.OSCSEL bit selects the local oscillator. By default the TMU works off the divided system clock (SCLK).	
		0	Select SCLK
		1	Select local oscillator
0 (R/W)	EN	Enable. The TMU_CTL.EN bit enables the TMU. By default the module is in power-down mode. The peripheral interface is active even in power-down mode (the TMU registers can be read/written).	
		0	Disable TMU
		1	Enable TMU

Fault High Limit Register

The `TMU_FLT_LIM_HI` register sets the temperature fault high limit as an integer value. The value is stored in two's complement format. Asserts `TMU_STAT.FLTHI` if the `TMU_TEMP` value is greater than or equal to `TMU_FLT_LIM_HI`. The `TMU_FLT_LIM_HI` value should be programmed for value greater than 8’h3C (60 degC).

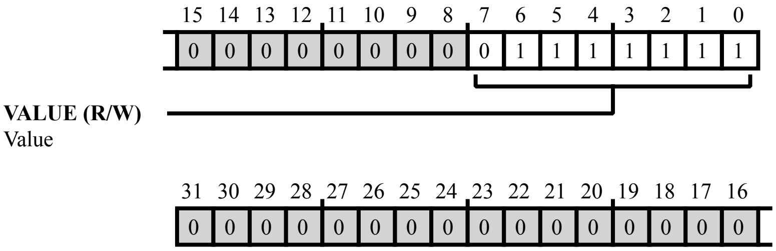

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-6: TMU_FLT_LIM_HI Register Diagram

Table 48-10: TMU_FLT_LIM_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Value. The `TMU_FLT_LIM_HI.VALUE` bit field sets the temperature high limit as an integer in two's complement format. If the limit value is greater than or equal to the high-limit the `TMU_STAT.FLTHI` bit is set. The limit-value should be programmed for value greater than 8h3C (60 degC).

Fault Low Limit Register

The `TMU_FLT_LIM_LO` register configures the fault temperature low limit.

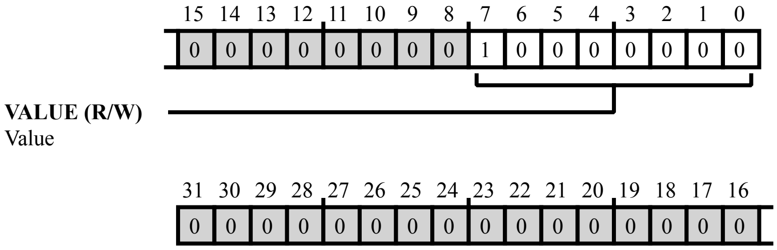

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-7: TMU_FLT_LIM_LO Register Diagram

Table 48-11: TMU_FLT_LIM_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Value. The `TMU_FLT_LIM_LO.VALUE` bit field configures the fault temperature low limit as an integer which is stored in two's complement format. If the temperature value is less than or equal to the low-limit, the `TMU_STAT.FLTLO` bit is set.

Gain Value Register

The TMU_GAIN register is used to configure the gain value in two's complement format. This value is used to correct the gain error in the TMU_TEMP register.

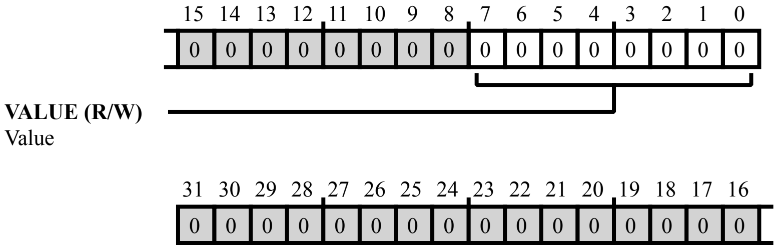

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-8: TMU_GAIN Register Diagram

Table 48-12: TMU_GAIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Value. The TMU_GAIN.VALUE bit field configures the gain value in two's complement format. This value is used to correct the gain error in the TMU_TEMP register.

Interrupt Mask Register

The TMU_IMSK register provides bit that are used to mask and unmask the interrupts associated with the TMU module.

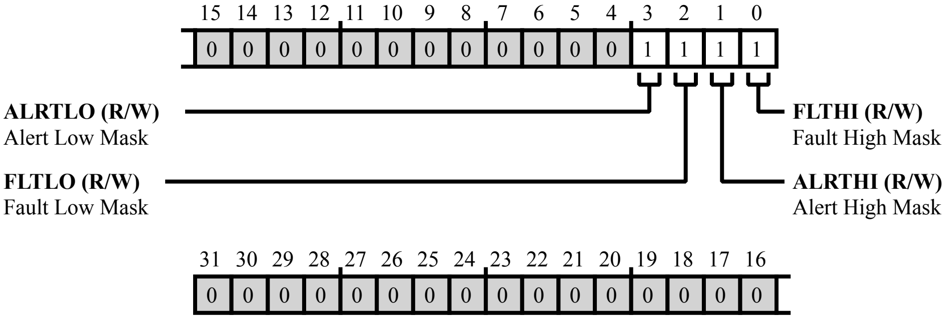

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
ALRTLO (R/W)
Alert Low Mask
FLTLO (R/W)
Fault Low Mask
FLTHI (R/W)
Fault High Mask
ALRTHI (R/W)
Alert High Mask
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-9: TMU_IMSK Register Diagram

Table 48-13: TMU_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ALRTLO	Alert Low Mask. The `TMU_IMSK.ALRTLO` bit masks or unmasks the alert low status change (`TMU_STAT.ALRTLO`).	
		0	Unmask alert low status change
		1	Mask alert low status change
2 (R/W)	FLTLO	Fault Low Mask. The `TMU_IMSK.FLTLO` bit masks or unmasks the fault low status change (`TMU_STAT.FLTLO`).	
		0	Unmask fault low status change
		1	Mask fault low status change
1 (R/W)	ALRTHI	Alert High Mask. The `TMU_IMSK.ALRTHI` bit masks or unmasks the alert high status change (`TMU_STAT.ALRTHI`).	
		0	Unmask alert high status change
		1	Mask alert high status change
0 (R/W)	FLTHI	Fault High Mask. The `TMU_IMSK.FLTHI` bit masks or unmasks the fault high interrupt (`TMU_STAT.FLTHI`).	
		0	Unmask fault high interrupt
		1	Mask fault high interrupt

Offset Register

The value programmed in the `TMU_OFFSET` register is used to correct the offset error in `TMU_TEMP` register.

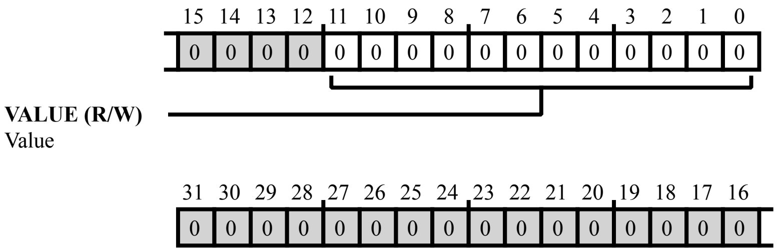

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-10: TMU_OFFSET Register Diagram

Table 48-14: TMU_OFFSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	VALUE	Value. The `TMU_OFFSET.VALUE` bit field provides the offset value which is used to correct the offset error in the `TMU_TEMP` register. This value is in a two's complement fixed point Q3.8 format where the 4 MSB's represent the integer part in two's complement format and the remaining 8 LSB's represent the decimal part. The offset value is applied after multiplying the gain.

Status Register

The TMU_STAT register bits indicate when an error or fault is detected.

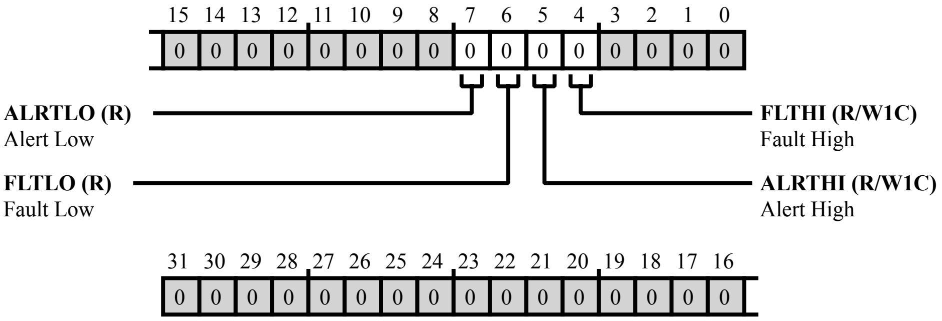


Figure 48-11: TMU_STAT Register Diagram

Table 48-15: TMU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	ALRTLO	Alert Low. The TMU_STAT.ALRTLO bit is set when the temperature value is less than or equal to the setting in the TMU_ALRT_LIM_LO register. The bit assertion is masked if TMU_IMSK.ALRTLO is HIGH.
6 (R/NW)	FLTLO	Fault Low. The TMU_STAT.FLTLO bit is set when the temperature value is less than or equal to the setting in the TMU_FLT_LIM_LO register. The bit assertion is masked if TMU_IMSK.FLTLO is HIGH.
5 (R/W1C)	ALRTHI	Alert High. The TMU_STAT.ALRTHI bit is set when the temperature value is greater than or equal to the setting in the TMU_ALRT_LIM_HI register. When Write1toClear is done on TMU_STAT.ALRTHI, it will be cleared only if the TMU_TEMP value is less than TMU_ALRT_LIM_HI value. If the bit is cleared by 'Write1toClear', The TMU_TEMP register is set to 16'h2C00 (effective temperature value of 44). The value 16'h2C00 will be changed according to gain, offset values from TMU_GAIN, TMU_OFFSET registers respectively.
4 (R/W1C)	FLTHI	Fault High. The TMU_STAT.FLTHI bit is set when the temperature value is greater than or equal to the setting in the TMU_FLT_LIM_HI register. When Write1toClear is done on TMU_STAT.FLTHI, it will be cleared only if the TMU_TEMP value is less than TMU_FLT_LIM_HI value. If the bit is cleared by 'Write1toClear', The TMU_TEMP register is set to 16'h2C00 (effective temperature value of 44). The value 16'h2C00 will be changed according to gain, offset values from TMU_GAIN, TMU_OFFSET registers respectively.

Temperature Value Register

The `TMU_TEMP` register provides the temperature value from the A/D converter and the status.

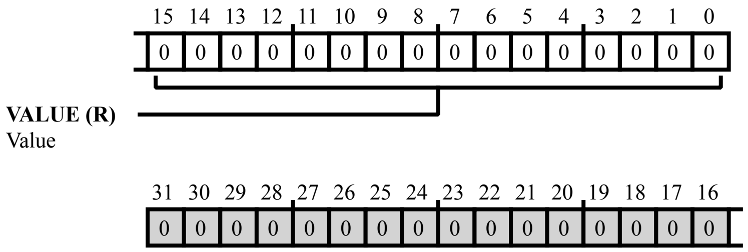

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 48-12: TMU_TEMP Register Diagram

Table 48-16: TMU_TEMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Value. The `TMU_TEMP.VALUE` bit field is the temperature value from A/D converter. This value is stored in two's complement fixed point Q7.8 format where the 8 MSB's represent the integer part in 2's complement, and the remaining 8 LSB's represent the decimal part.

49 Harmonic Analysis Engine (HAE)

The harmonic analysis engine (HAE) analyzes harmonic frequencies present on the voltage and current input samples. The HAE receives input samples from two source channels whose frequencies are 45–65 Hz. The HAE then processes the input samples and produces output results. The output results consist of power quality measurements of the fundamental and up to 12 more harmonics.

HAE Features

The HAE features include:

- Processing of two 24-bit signed input channels, consisting of one voltage and one current. The input full scale is limited to ~ +/-6,000,000
- Processing of fundamental frequencies between 45-65 Hz
- Processing of input samples at a nominal 8 kHz rate
- Processing of the fundamental plus 12 harmonic frequencies
- Active, reactive, apparent, I_{RMS}, V_{RMS}, and power factor on the fundamental frequency
- Active, reactive, apparent, I_{RMS}, V_{RMS}, power factor, I_{HD+n}, and V_{HD+n} on the 12 harmonic frequencies
- The accuracy of the measurements, relative to a full scale of +/-6,000,000:
    - Fundamental active or reactive powers 0.1% down to 1/1000 of full scale
    - Fundamental apparent power 0.2% down to 1/1000 of full scale
    - Fundamental I_{RMS}/V_{RMS} 0.1% down to 1/1000 of full scale
    - Fundamental power factor 0.3% based on active and apparent accuracy
    - Harmonic active or reactive powers 1% down to 1/1000 of full scale
    - Harmonic apparent power 2% down to 1/1000 of full scale
    - Harmonic I_{RMS}/V_{RMS} 1% down to 1/1000 of full scale
    - Harmonic power factor 3% based on active and apparent accuracy

  - Harmonic I_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic I_{RMS}
  - Harmonic V_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic V_{RMS}
- Twelve 6-bit fields for selecting harmonics to analyze, limited by bandwidth of the input signal. The fundamental component always is provided.

HAE Functional Description

The following sections provide a functional description of the HAE.

Harmonic engine

The hardware block of the harmonic engine works with other HAE blocks to co-process full and partial results

Harmonic Analyzer

The harmonic analyzer block works with the harmonic engine to co-process full and partial results.

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

ADSP-SC58x HAE Register List

The Harmonic Analysis Engine (HAE) analyzes harmonics present on voltage and current input samples. The HAE receives input samples from two source channels, processes the samples, and produces output results. The output results consist of power quality measurements of the fundamental and up to twelve additional harmonic components. A set of registers governs HAE operations. For more information on HAE functionality, see the HAE register descriptions.

Table 49-1: ADSP-SC58x HAE Register List

Name	Description
HAE_CFG0	Configuration 0 Register
HAE_CFG1	Configuration 1 Register
HAE_CFG2	Configuration 2 Register
HAE_CFG3	Configuration 3 Register
HAE_CFG4	Configuration 4 Register
HAE_DIDT_COEF	DIDT Coefficient Register
HAE_DIDT_GAIN	DIDT Gain Register
HAE_H[nn]_INDX	Harmonic n Index Register
HAE_ISAMPLE	I (Current) Sample Register

Table 49-1: ADSP-SC58x HAE Register List (Continued)

Name	Description
`HAE_IWAVEFORM`	I (Current) Waveform Register
`HAE_RUN`	Run Register
`HAE_STAT`	Status Register
`HAE_VLEVEL`	Voltage Level Register
`HAE_VSAMPLE`	V (Voltage) Sample Register
`HAE_VWAVEFORM`	V (Voltage) Waveform Register

ADSP-SC58x HAE Interrupt List

Table 49-2: ADSP-SC58x HAE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
158	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0	Level	32
159	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1	Level	33
160	HAE0_TXDMA	HAE0 TX DMA Channel 0	Level	31
161	HAE0_STAT	HAE0 Status	Level	
212	HAE0_RXDMA_CH0_ERR	HAE0 RX DMA Channel 0 Error	Level	
213	HAE0_RXDMA_CH1_ERR	HAE0 RX DMA Channel 1 Error	Level	
214	HAE0_TXDMA_ERR	HAE0 TX DMA Channel Error	Level	

ADSP-SC58x HAE Trigger List

Table 49-3: ADSP-SC58x HAE Trigger List Masters

Trigger ID	Name	Description	Sensitivity
46	`HAE0_RXDMA_CH0`	HAE0 RX DMA Channel 0	Level
47	`HAE0_RXDMA_CH1`	HAE0 RX DMA Channel 1	Level
48	`HAE0_TXDMA`	HAE0 TX DMA Channel 0	Level

Table 49-4: ADSP-SC58x HAE Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
39	`HAE0_RXDMA_CH0`	HAE0 RX DMA Channel 0	Pulse
40	`HAE0_RXDMA_CH1`	HAE0 RX DMA Channel 1	Pulse
41	`HAE0_TXDMA`	HAE0 TX DMA Channel 0	Pulse

ADSP-SC58x HAE DMA Channel List

Table 49-5: ADSP-SC58x HAE DMA Channel List

DMA ID	DMA Channel Name	Description
DMA31	HAE0_TXDMA	HAE0 TX DMA Channel
DMA32	HAE0_RXDMA_CH0	HAE0 RX DMA Channel 0
DMA33	HAE0_RXDMA_CH1	HAE0 RX DMA Channel 1

HAE Block Diagram

The *HAE Block Diagram* shows the functional blocks within the HAE.

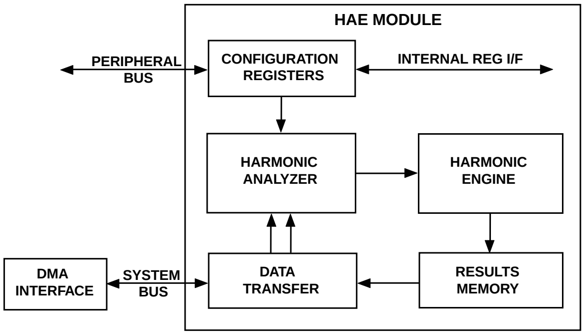

HAE MODULE
PERIPHERAL BUS
CONFIGURATION REGISTERS
INTERNAL REG I/F
HARMONIC ANALYZER
HARMONIC ENGINE
DMA INTERFACE
SYSTEM BUS
DATA TRANSFER
RESULTS MEMORY

Figure 49-1: HAE Block Diagram

HAE Architectural Concepts

Using the HAE features and event control to their greatest potential requires an understanding of these architectural concepts.

- Harmonic Engine
- Harmonic Analyzer
- Data Transfer Module
- Results Memory

Harmonic Engine

The *HAE Engine Block Diagram* presents a synthesized diagram of the harmonic engine, its settings, and its output registers.

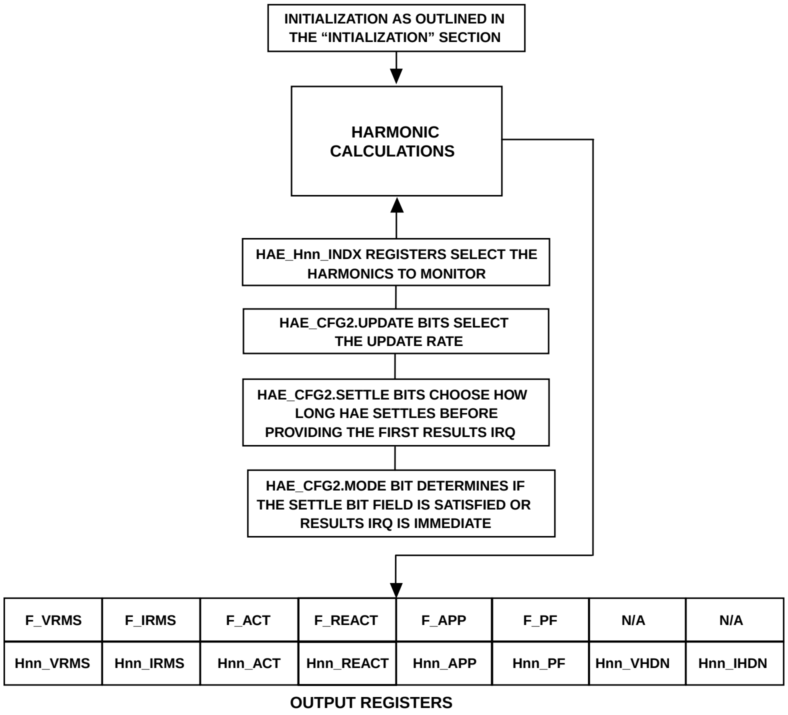

INITIALIZATION AS OUTLINED IN THE "INTIALIZATION" SECTION
HARMONIC CALCULATIONS
HAE_Hnn_INDX REGISTERS SELECT THE HARMONICS TO MONITOR
HAE_CFG2.UPDATE BITS SELECT THE UPDATE RATE
HAE_CFG2.SETTLE BITS CHOOSE HOW LONG HAE SETTLES BEFORE PROVIDING THE FIRST RESULTS IRQ
HAE_CFG2.MODE BIT DETERMINES IF THE SETTLE BIT FIELD IS SATISFIED OR RESULTS IRQ IS IMMEDIATE
F_VRMS
F_IRMS
F_ACT
F_REACT
F_APP
F_PF
N/A
N/A
Hnn_VRMS
Hnn_IRMS
Hnn_ACT
Hnn_REACT
Hnn_APP
Hnn_PF
Hnn_VHDN
Hnn_IHDN
OUTPUT REGISTERS

Figure 49-2: HAE Engine Block Diagram

The harmonic engine hardware block works with other HAE blocks to co-process full and partial results (see Harmonic Calculations). At the start of a new sampling period (described in Initialization), the harmonic engine cycles through predefined locations in data RAM, which contain the analyzer processing results. See Harmonic Analyzer for more information.

As the harmonic engine produces results in their final formats (described in HAE Result Ranges and Formats), the results are stored in the results memory (see Results Memory).

The HAE engine computes harmonic information for line frequencies 45–66 Hz. Neutral current can also be analyzed simultaneously with the sum of phase currents. See Theory of Operation for more information.

Harmonic Analyzer

The harmonic analyzer block works with the Harmonic Engine to co-process full and partial results.

To perform harmonic analysis, the HAE contains a pair of voltage and current inputs and a set of 12 indexes to indicate which harmonic components to extract. See Theory of Operation and Harmonic Calculations for more information.

High-Pass Filters (HPFs)

The voltage and current have the option of being high-pass filtered to remove DC offsets. The ***Frequency Response for High-Pass Filters*** figure shows the frequency response of the high-pass filters (the -3 dB point is around 1.1 Hz). The HPFs are enabled by default.

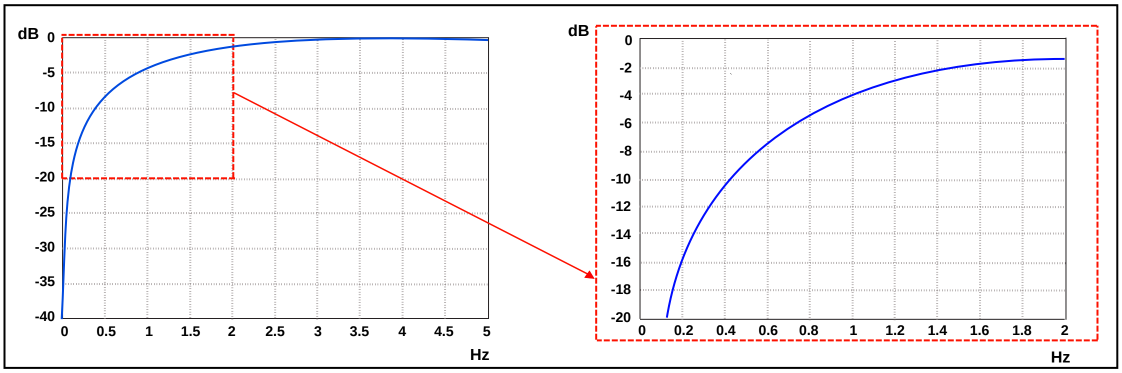


Figure 49-3: Frequency Response for High-Pass Filters

Digital Integrators

For cases when the current is sensed with a di/dt sensor, such as a Rogowski coil, the HAE offers an internal digital integrator for compensation. Ideally, it causes a perfect 90 degrees of phase shift at all the frequencies; at 50 Hz it is close to that value (see the ***Digital Integrator*** figure). The integrator is necessary to restore the signal to its original form before using the signal in HAE calculations. The digital integrator is disabled by default.

With digital integrator enabled for current channel, the amplitude of current will change with the frequency according to a slope of 20 dB per decade. For frequency of 50 Hz, the amplitude of current remains same after passing through the integrator. However, for other frequencies in the range 45-65 Hz, the amplitude of current changes with a slope of 20 dB per decade after passing through integrator.

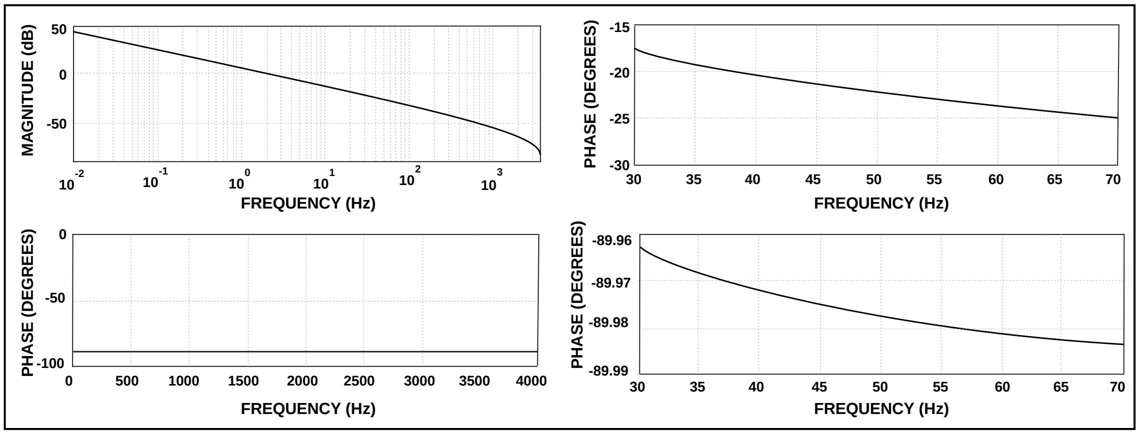


Figure 49-4: Digital Integrator

Phase-Locked Loop and Clock Control (PLL)

The fundamental frequency of the system is extracted from the voltage signal using the phase-locked loop and clock control (PLL) techniques. PLL techniques are optimized for signals with frequencies used in standard power grids around the world (50 Hz or 60 Hz). The techniques consider possible deviations of up to 5 Hz, so the final guaranteed operating range is from 45 Hz to 65 Hz.

The initial detection time of the frequency depends on its value and can take up to several seconds. This activity only happens at the start-up of the HAE block. Once the value of the fundamental frequency is detected, the PLL tracks it continuously.

Settling Times

The block that extracts all of the harmonic RMS and power values is replicated 12 times in parallel for all of the harmonic indexes plus once for the fundamental. Once an index for a particular harmonic changes, there is a settling time of about 700 mSec (~700 / 0.125 = 5600 8K samples). This settling time achieves less than 0.1% error needed for all the internal RMS and powers computations (see the ***Fundamental and Harmonic Values - Settling Times*** plots).

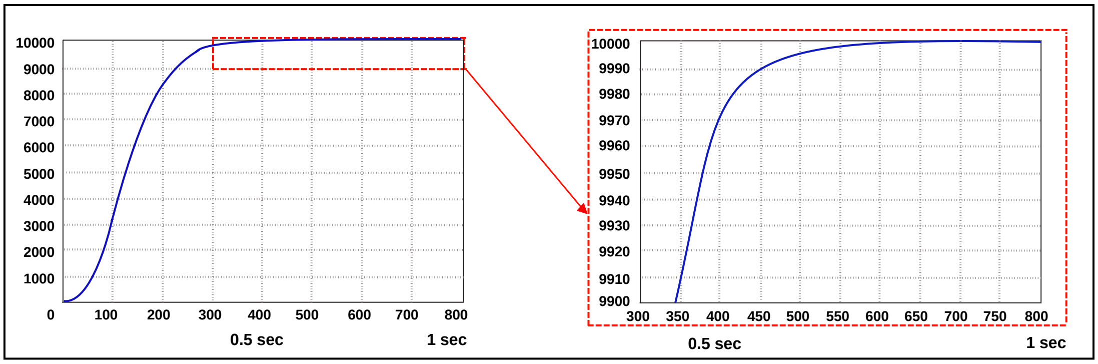


Figure 49-5: Fundamental and Harmonic Values - Settling Times

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

RX I (Receive Current Sample) Channel

The RX I channel transfers HAE current (I) input samples from system memory through direct memory access (DMA). The RX I request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX I samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched. The SCLK0_0 / `HAE_CFG1.STARTDIV` determines the HAE sample loop, which must be programmed to be nominally 8 kHz in frequency. There is one `DWORD` transfer each HAE sample loop.

There is also an RX I memory-mapped location (`HAE_ISAMPLE` register) in peripheral space (MMR), which enables the MCU to provide the I channel samples, timed with the `HAE_STAT.RXIRQ` bit, if desired.

RX V (Receive Voltage Sample) Channel

The RX V channel transfers HAE voltage (V) input samples from system memory through DMA. The RX V request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX V samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched.

The SCLK0_0 / `HAE_CFG1.STARTDIV`, determines the HAE sample loop which must be programmed to be nominally 8 kHz in frequency. There is one `DWORD` transfer each HAE sample loop.

There is also an RX V memory-mapped location (`HAE_VSAMPLE` register) in peripheral space (MMR), which enables the MCU to provide the V channel samples, timed with the `HAE_STAT.RXIRQ` bit, if desired.

Example:

- The `HAE_STAT.RXIRQ` bit toggles prior to RX transfers:

  The RX interrupt is used internally in hardware to request I and V samples. The MCU can also use the interrupt when it supplies the input samples, rather than the DMA interface.

- RX transfers indicate that the RX channel is ready:

  The HAE RX channels request one RX sample at an 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The holding register stores the RX data, replacing the previous data. The holding register contents are moved up to an output register and driven to the Harmonic Analyzer module for processing. This operation amounts to a two-deep FIFO, allowing a full sample time of latency on the input sample arrival, nominally 125 uSec (8 kHz).

  If the MCU supplies the input samples, rather than the DMA RX interfaces, there are two memory-mapped locations (`HAE_ISAMPLE` and `HAE_VSAMPLE`), where the next samples are written. As previously mentioned, the HAE can use the `HAE_STAT.RXIRQ` bit to time the sample delivery.

TX (Transmit Results) Channel

The TX channel transfers HAE results from results memory to system memory through DMA. The results for the fundamental and 12 harmonics are stored in 13 8-location fields in the results memory. Therefore, the maximum number of `DWORD`s to transfer is 13 x 8 = 104. Use the `HAE_CFG3.CHANEN` bits to select the channels to transfer.

The HAE can request a transfer of results for each 8 kHz sample period, meaning that up to 104 `DWORDs` can be transferred each 125 uSec. Use the `HAE_CFG2.UPDATE` bit field to set the request rate to longer intervals. The results also are memory-mapped to peripheral address space, which enables the MCU to read the results, timed with the `HAE_STAT.TXIRQ` bit, if desired.

Example:

- The `HAE_CFG3.CHANEN` bit field is set to 0x009 to transfer the fundamental and harmonic contained in the `HAE_H3_INDX` register (programmed to 5th).

- The `HAE_STAT.TXIRQ` bit toggles prior to TX transfers:

  The HAE uses the TX interrupt internally in hardware to start the transfer. The MCU can also use the interrupt when the MCU reads the HAE results, rather than the DMA interface transferring it.

- TX transfers indicate that the HAE results data is ready:

  The HAE TX channel can transfer one `DWORD` every other clock. The internal hardware parses through the `HAE_CFG3.CHANEN` bit field, eventually traversing all 13 bits. If consecutive channels are enabled, two more

`IDLE` clocks are inserted between the adjacent channels for a total of three `IDLE` clocks. When channels are skipped, one more `IDLE` clock is inserted for each unselected channel. Therefore, there are 3 (inter-channel `IDLE`) + 2 (skipped channels) = 5 `IDLE` clocks between the fundamental and `HAE_H3_INDX` transfers.

Results Memory

The results memory is organized by the fundamental and 12 harmonic indexes. Each of the 13 potentially analyzed frequencies has eight locations dedicated to store various power quality measurements. Each frequency is offset from the next by eight locations.

The ***HAE Results Memory*** figure shows the results memory contents.

	INDEX	0	1	2	3	4	5	6	7
GROUP	OFFSET								
FUNDAMENTAL	0x00	F_IRMS	F_VRMS	F_ACT	F_REACT	F_APP	F_PF	N/A	N/A
H1_INDEX	0x08	H1_IRMS	H1_VRMS	H1_ACT	H1_REACT	H1_APP	H1_PF	H1_IHDN	H1_VHDN
H2_INDEX	0x10	H2_IRMS	H2_VRMS	H2_ACT	H2_REACT	H2_APP	H2_PF	H2_IHDN	H2_VHDN
H3_INDEX	0x18	H3_IRMS	H3_VRMS	H3_ACT	H3_REACT	H3_APP	H3_PF	H3_IHDN	H3_VHDN
H4_INDEX	0x20	H4_IRMS	H4_VRMS	H4_ACT	H4_REACT	H4_APP	H4_PF	H4_IHDN	H4_VHDN
H5_INDEX	0x28	H5_IRMS	H5_VRMS	H5_ACT	H5_REACT	H5_APP	H5_PF	H5_IHDN	H5_VHDN
H6_INDEX	0x30	H6_IRMS	H6_VRMS	H6_ACT	H6_REACT	H6_APP	H6_PF	H6_IHDN	H6_VHDN
H7_INDEX	0x38	H7_IRMS	H7_VRMS	H7_ACT	H7_REACT	H7_APP	H7_PF	H7_IHDN	H7_VHDN
H8_INDEX	0x40	H8_IRMS	H8_VRMS	H8_ACT	H8_REACT	H8_APP	H8_PF	H8_IHDN	H8_VHDN
H9_INDEX	0x48	H9_IRMS	H9_VRMS	H9_ACT	H9_REACT	H9_APP	H9_PF	H9_IHDN	H9_VHDN
H10_INDEX	0x50	H10_IRMS	H10_VRMS	H10_ACT	H10_REACT	H10_APP	H10_PF	H10_IHDN	H10_VHDN
H11_INDEX	0x58	H11_IRMS	H11_VRMS	H11_ACT	H11_REACT	H11_APP	H11_PF	H11_IHDN	H11_VHDN
H12_INDEX	0x60	H12_IRMS	H12_VRMS	H12_ACT	H12_REACT	H12_APP	H12_PF	H12_IHDN	H12_VHDN

Figure 49-6: HAE Results Memory

RAM Parity Protection

HAE Results Upper Byte ID

The HAE results are stored in a 24-bit wide memory as shown in Results Memory.

When the system bus moves the results into memory, the HAE hardware appends a unique `CHANNEL:INDEX` identifier to the upper byte of each results location. The intent of the ID byte is to help in data parsing of HAE results.

The 32-bit appended result is as follows:

`CHANNEL[3:0]` ||	`INDEX[3:0]` ||	`RESULTS[23:0]`

The `CHANNEL` in the upper byte corresponds to `nn` in `Hnn_INDX` of that particular results location. The `INDX` in the upper byte corresponds to the `INDX` within the `Hnn_INDX` grouping.

For example, location `H7_PF` has `0x75` in the upper byte of the system bus transfer of that HAE results memory location.

HAE Result Ranges and Formats

The HAE results, stored in the results memory, have formats appropriate for the given measurement. Measurements accuracy is relative to the full scale value of the input samples, and the assumed full scale is +/~6,000,000. If the full scale is above this value, overflow can occur and the results are undefined. A lower full scale limits the dynamic range accuracy of the measurements. For example, a full scale, which is 50% lower than the assumed 6,000,000 full scale, has the dynamic range reduced by 50%. Potentially, this result can be better or worse, as determined by the ADC noise floor supplying the input samples.

The HAE results are as linear as the input samples, within the accuracy ranges specified earlier. The results assume a full-scale input sample of ~=/-6,000,000 and sufficiently low ADC noise floor. Evaluate the system using your specific ADC and full scale specifications to predict the HAE range of accuracy and expected results.

The RMS values of voltage and current calculated from HAE are exactly not same as the theoretical RMS values. The ratio between HAE calculated RMS value and the theoretical value is around 1.11. A gain factor should be added in the software so as to match the theoretical and HAE calculated RMS values. Calibration can be done by calculating the ratio between HAE calculated RMS value and the theoretical one for a particular amplitude of V and I signal and line frequency.

The following list gives the HAE result formats.

- I_{RMS} and V_{RMS}: both fundamental and harmonic I_{RMS} and V_{RMS} are unsigned magnitudes with full scale of ~4,200,000
- Active and reactive power: both active and reactive powers are signed numbers with full scale of ~+/-4,200,000
- Apparent power: apparent power is an unsigned magnitude with full scale of ~4,200,000
- Power factor: each LSB of the fundamental and harmonic power factors equates to a weight of 2^{-23}. Hence, the maximum register value of 0x7FFFFF equates to a power factor value of 1. The minimum register value of 0x800000 corresponds to a power factor of -1. If the power factor is outside of the -1 to +1 range because of offset and gain calibrations, the result is set at -1 or +1. The result depends on the sign of the fundamental reactive power.
- I_{HD+n}/V_{HD+n}: the harmonic distortion plus noise ratios are computed using the RMS of the fundamental and the RMS of the harmonic under analysis. In other words, the ratio only covers the particular harmonic under analysis versus the fundamental. The ratios are stored as 24-bit values in 3.21 signed format. The ratios are limited to +3.9999, and all greater results are clamped to it. The `HD+n` ratios cannot be negative.

HAE Operating Modes

The HAE uses the DMA interface to transfer data to and from system memory. The HAE configuration registers enable the module and calibrate its frequency (clock) divide and other parameters, as described in HAE Programming Model. The HAE triggers and status signals indicate system events and errors.

HAE Data Transfer Modes

The HAE uses the RX DMA interface to transfer data from system memory. Samples for the current (I) and voltage (V) channels of the AFE or SINC filter compose the data: two `WORDS` are transferred each 8 kHz sample period into the HAE (I and V).

The HAE uses the TX DMA interface to transfer data to system memory. The fundamental and selected harmonic results compose the data: up to 13 (fundamental + 12 harmonics) x 8 = 104 `DWORD`s are transferred each 8K sample period.

See Data Transfer Module for more information.

HAE Event Control

The HAE module uses DMA to transfer samples to and data from system memory. The HAE also can use TX and RX events to time the arrival of input samples and extract the results by the MCU:

- The MCU uses the RX event (`HAE_STAT.RXIRQ`) as an IRQ to time when to write the I and V samples into the HAE.
- The MCU uses the TX event (`HAE_STAT.TXIRQ`) as an IRQ to time when to read the HAE results.

HAE Interrupt Signals

The interrupt signals to the HAE module include:

- The RX interrupt to time the delivery of waveform samples as inputs to the HAE.
- The TX interrupt to time when the HAE results memory is ready with new values for the current sample.

The HAE generates the RX interrupts at a rate specified by the `HAE_CFG1.STARTDIV` bit field and the TX interrupts at a rate specified by the `HAE_CFG2.UPDATE` bit field. Refer to Data Transfer Module for more information.

HAE Status and Error Signals

The trigger and status signals to the HAE module include:

- `HAE_STAT.RDY` (HAE ready status). When the bit is set, the HAE is fully accessible.
- `HAE_STAT.RXIRQ` (RX IRQ status). The bit mirrors the RX interrupt.
- `HAE_STAT.TXIRQ` (TX IRQ status). The bit mirrors the TX interrupt.

The status bit requires a 1 to clear and reenable the corresponding interrupt for the next sample period. Refer to Data Transfer Module for more information.

HAE Programming Model

The following sections provide basic procedures for configuring various HAE operations.

Current and voltage data can be transferred from system memory to HAE. HAE results can be transferred to system memory using core and DMA accesses. DMA transfers can be set up to transfer a configurable number of I and V samples and HAE results between HAE and system memory automatically. Core-driven transfers use HAE interrupts to signal the processor core to provide I and V data to the sample registers and read the HAE results from the system memory.

The following sections provide recommended programming guidelines to be followed for core and DMA mode.

Configuring the HAE for DMA Transfers

- Enable HAE operations by configuring the `HAE_CFG2.EN` bit.
- Poll for HAE ready status from the `HAE_STAT.RDY` bit configuration.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Choose the harmonic channels to be monitored by configuring the `HAE_H[nn]_INDX` registers and enable channels by configuring the `HAE_CFG3.CHANEN` bits.
- Configure the RX DMA channel 0 and the RX DMA channel 1 to receive I and V samples. Configure the TX DMA channel 0 to transmit harmonic calculation results.
- Set the HAE clock divider using the `HAE_CFG1.STARTDIV` field to 8 kHz.
- Set the `HAE_RUN` register.

Configuring the HAE for Core Transfers

- Enable HAE operations by configuring the `HAE_CFG2.EN` bit.
- Poll for HAE ready status using the `HAE_STAT.RDY` bit.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Enable receive and transmit IRQ using the `HAE_CFG0` register.
- Configure the `HAE_VLEVEL` register depending on input voltage magnitude requirements.
- Configure the `HAE_H[nn]_INDX` registers to choose the harmonic channels to monitor. Enable the channels using the `HAE_CFG3.CHANEN` bits.
- Set the HAE clock divider using the `HAE_CFG1.STARTDIV` field to 8 kHz.

- Set the `HAE_RUN` register.
- At each receive IRQ (`HAE_STAT.RXIRQ` = 1), write I and V sample values to the `HAE_ISAMPLE` and the `HAE_VSAMPLE` registers respectively.
- At each transmit IRQ (`HAE_STAT.TXIRQ` = 1), read the HAE results from the HAE result RAM memory region based on the index.

HAE Programming Concepts

Using the HAE features to their greatest potential requires an understanding of some HAE-related concepts.

- Theory of Operation
- Initialization
- Harmonic Calculations
- Configuring Harmonic Calculations Update Rate

Theory of Operation

The HAE theory of operation can be described using the following scenario:

Consider a nonsinusoidal AC system supplied by a voltage, v(t), that consumes the current, i(t):

$$v(t) = \sum_{k=1}^{\infty} V_k \sqrt{2} \sin(k\omega t + \varphi_k)$$

$$i(t) = \sum_{k=1}^{\infty} I_k \sqrt{2} \sin(k\omega t + \gamma_k)$$

where:

- V_k, I_k are the RMS voltage and current, respectively, of each harmonic.
- Φ_k, γ_k are the phase delays of each harmonic.
- ω is the angular velocity at the fundamental (line) frequency f.

The HAE harmonic calculations are specified for line frequencies 45–65 Hz.

The maximum number of harmonics which the HAE can accept for a particular line frequency depends upon the bandwidth of the input waveforms supplied.

When the HAE analyzes I and V samples, the following metering quantities are computed:

- Fundamental current RMS: I_1
- Fundamental voltage RMS: V_1
- RMS of up to 12 harmonics of the current channel: I_n, n = 2, 3,…, 12
- RMS of up to 12 harmonics of the voltage channel: V_n, n = 2, 3,…, 12

- Fundamental active power: $P_1 = V_1 I_1 \cos(\varphi_1 - \gamma_1)$
- Fundamental reactive power: $Q_1 = V_1 I_1 \sin(\varphi_1 - \gamma_1)$
- Fundamental apparent power: $S_1 = V_1 I_1$
- Power factor of the fundamental:

  $$pf_1 = \operatorname{sgn}(Q_1)\cdot \frac{P_1}{S_1}$$

  Active power of up to 12 harmonics:

  $P_n = V_n I_n \cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Reactive power of up to 12 harmonics:

  $Q_n = V_n I_n \sin(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Apparent power of up to 12 harmonics:

  $S_n = V_n I_n$, n = 2, 3, …, 12
- Power factor of up to 12 harmonics:

  $$pf_n = \operatorname{sgn}(Q_n)\cdot \frac{P_n}{S_n}, \quad n = 2, 3, \ldots, 12$$
- Total harmonic distortion of the current channel:

  $$(THD)_I = \frac{\sqrt{I^2 - I_1^2}}{I_1}$$
- Total harmonic distortion of the voltage channel:

  $$(THD)_V = \frac{\sqrt{V^2 - V_1^2}}{V_1}$$
- Harmonic distortion of up to 12 harmonics on the current channel:

  $$HD_{I_n} = \frac{I_n}{I_1}, \quad n = 2, 3, \ldots, 12$$
- Harmonic distortion of up to 12 harmonics on the voltage channel:

  $$HD_{V_n} = \frac{V_n}{V_1}, \quad n = 2, 3, \ldots, 12$$

Initialization

The HAE typically is initialized with parameters and settings for a given usage scenario. It is interrupt-driven when new results are available.

The HAE programming depends on whether the line frequency is 50 Hz or 60 Hz. If the external sensor is a di/dt type, there are also some differences. The ***HAE Initialization*** flowchart shows the initialization sequence.

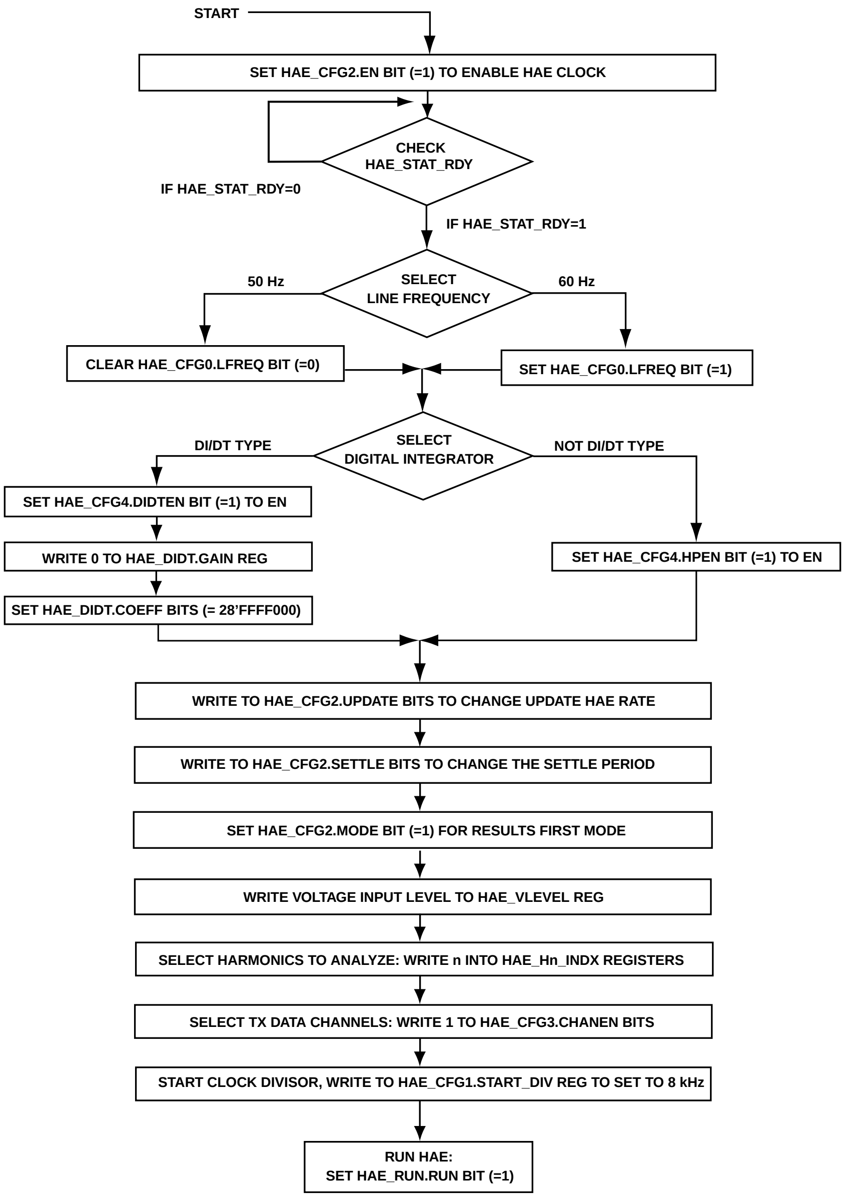

START
SET HAE_CFG2.EN BIT (=1) TO ENABLE HAE CLOCK
CHECK HAE_STAT_RDY
IF HAE_STAT_RDY=0
IF HAE_STAT_RDY=1
SELECT LINE FREQUENCY
50 Hz
60 Hz
CLEAR HAE_CFG0.LFREQ BIT (=0)
SET HAE_CFG0.LFREQ BIT (=1)
SELECT DIGITAL INTEGRATOR
DI/DT TYPE
NOT DI/DT TYPE
SET HAE_CFG4.DIDTEN BIT (=1) TO EN
WRITE 0 TO HAE_DIDT.GAIN REG
SET HAE_DIDT.COEFF BITS (= 28'FFFF000)
SET HAE_CFG4.HPEN BIT (=1) TO EN
WRITE TO HAE_CFG2.UPDATE BITS TO CHANGE UPDATE HAE RATE
WRITE TO HAE_CFG2.SETTLE BITS TO CHANGE THE SETTLE PERIOD
SET HAE_CFG2.MODE BIT (=1) FOR RESULTS FIRST MODE
WRITE VOLTAGE INPUT LEVEL TO HAE_VLEVEL REG
SELECT HARMONICS TO ANALYZE: WRITE n INTO HAE_Hn_INDX REGISTERS
SELECT TX DATA CHANNELS: WRITE 1 TO HAE_CFG3.CHANEN BITS
START CLOCK DIVISOR, WRITE TO HAE_CFG1.START_DIV REG TO SET TO 8 kHz
RUN HAE:
SET HAE_RUN.RUN BIT (=1)

Figure 49-7: HAE Initialization

Harmonic Calculations

When the harmonic engine runs, it computes information about the fundamental and up to 12 harmonics. The HAE simultaneously monitors the indexes of the additional 12 harmonics, provided by the 8-bit registers `HAE_H[nn]_INDX`. Write the index of the harmonic into the register for the harmonic to be monitored. If the second harmonic is monitored, write `2`. If harmonic 51 is desired, write `51`. The HAE always monitors the fundamental component, independent of the values written into `HAE_H[nn]_INDX`. Therefore, if one of these registers is equal to `1`, the HAE monitors the fundamental components multiple times. The maximum index allowed in the `HAE_H[nn]_INDX` registers is 63.

As a reference, the ***Harmonic Engine Outputs and Registers where Values are Stored*** table presents the harmonic engine outputs and registers that store the outputs.

Table 49-6: Harmonic Engine Outputs and Registers where Values are Stored

Quantity	Definition	HAE Registers
RMS of the Fundamental Component	V_1, I_1	`F_VRMS`, `F_IRMS`
RMS of a Harmonic Component	V_n, I_n, n = 2, 3,…, 12	`Hnn_VRMS`, `Hnn_XIRMS`
Active Power of the Fundamental Component	$P_1 = V_1 I_1 \cos(\varphi_1 - \gamma_1)$	`F_ACT`
Active Power of a Harmonic Component	$P_n = V_n I_n \cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12	`Fnn_ACT`
Reactive Power of the Fundamental Component	$Q_1 = V_1 I_1 \sin(\varphi_1 - \gamma_1)$	`F_REACT`
Reactive Power of a Harmonic Component	$Q_n = V_n I_n \sin(\varphi_1 - \gamma_1)$, n = 2, 3,…, 12	`Hnn_REACT`
Apparent Power of the Fundamental Component	$S_1 = V_1 I_1$	`F_APP`
Apparent Power of a Harmonic Component	$S_n = V_n I_n$, n = 2, 3, …, 12	`Hnn_APP`
Power Factor of the Fundamental Component	$pf_1 = \mathrm{sgn}(Q_1) \cdot \frac{P_1}{S_1}$	`F_PF`
Power Factor of a Harmonic Component	$pf_n = \mathrm{sgn}(Q_n) \cdot \frac{P_n}{S_n}$, n = 2, 3,…, 12	`Hnn_PF`
Harmonic Distortion of a Harmonic Component	$HD_{V_n} = \frac{V_n}{V_1}$, $HD_{I_n} = \frac{I_n}{I_1}$, n = 2, 3,…, 12	`Hnn_VHDN`, `Hnn_IHDN`

Configuring Harmonic Calculations Update Rate

The harmonic engine functions at an 8 kHz rate. From the moment the `HAE_CFG2` register is initialized, and the harmonic indexes are set in the `HAE_H[nn]_INDX` index registers, the HAE calculations take typically 750 mSec to settle within the specification parameters.

The HAE module uses the `HAE_CFG2.UPDATE` bits to manage the update rate of the output registers for the harmonic engine. It manages the update rate independent of the calculations rate of 8 kHz for the engine. The default value of `000` means that the registers are updated every 125 uSec (8 kHz rate). Other update periods are: 250 uSec (`001`), 1 mSec (`010`), 16 mSec (`011`), 128 mSec (`100`), 512 mSec (`101`), 1.024 mSec (`110`). If the `HAE_CFG2.UPDATE` bits are `111`, the harmonic calculations are disabled.

The HAE module provides two ways to manage the harmonic computations. It enables the first approach when bit `HAE_CFG2.MODE` is cleared to its default value of 0. The state sets status bit `HAE_STAT.TXIRQ` to 1 after a certain period and then every time the harmonic calculations are updated at `HAE_CFG2.UPDATE` frequency. This functionality allows an external microcontroller to access the harmonic calculations only after they have settled. The HAE uses the state of bits `HAE_CFG2.SETTLE` to determine the time period. The possible values of settling time of the harmonic calculations are 512 mSec (00), 768 mSec(01), 1024 mSec (10), and 1280 mSec (11).

The HAE module enables the second approach when bit `HAE_CFG2.MODE` is set to 1. The state sets the status bit `HAE_STAT.TXIRQ` to 1 every time the harmonic calculations are updated at the `HAE_CFG2.UPDATE` frequency, without waiting for the harmonic calculations to settle. This functionality allows an external microcontroller to access the harmonic calculations immediately after starting. A write to the `HAE_STAT` register clears the status bit. The corresponding bit (`HAE_STAT.TXIRQ`) is set to 1.

ADSP-SC58x HAE Register Descriptions

Harmonic Analysis Engine (HAE) contains the following registers.

Table 49-7: ADSP-SC58x HAE Register List

Name	Description
`HAE_CFG0`	Configuration 0 Register
`HAE_CFG1`	Configuration 1 Register
`HAE_CFG2`	Configuration 2 Register
`HAE_CFG3`	Configuration 3 Register
`HAE_CFG4`	Configuration 4 Register
`HAE_DIDT_COEF`	DIDT Coefficient Register
`HAE_DIDT_GAIN`	DIDT Gain Register
`HAE_H[nn]_INDX`	Harmonic n Index Register
`HAE_ISAMPLE`	I (Current) Sample Register
`HAE_IWAVEFORM`	I (Current) Waveform Register
`HAE_RUN`	Run Register
`HAE_STAT`	Status Register
`HAE_VLEVEL`	Voltage Level Register
`HAE_VSAMPLE`	V (Voltage) Sample Register
`HAE_VWAVEFORM`	V (Voltage) Waveform Register

Configuration 0 Register

The `HAE_CFG0` register configures high-level interrupts and specifies the line frequency for HAE operations.

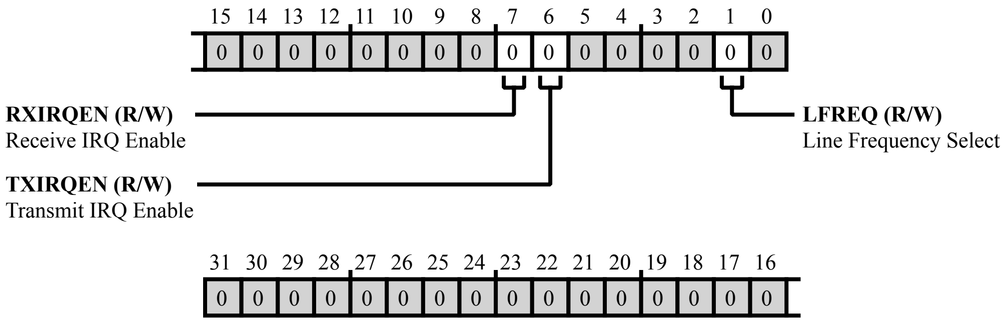

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RXIRQEN (R/W)
Receive IRQ Enable
TXIRQEN (R/W)
Transmit IRQ Enable
LFREQ (R/W)
Line Frequency Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-8: HAE_CFG0 Register Diagram

Table 49-8: HAE_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	RXIRQEN	Receive IRQ Enable. The `HAE_CFG0.RXIRQEN` bit enables an interrupt, which the HAE triggers on each request for a new input sample on both the I and V channels.	
		0	Disable
		1	Enable
6 (R/W)	TXIRQEN	Transmit IRQ Enable. The `HAE_CFG0.TXIRQEN` bit enables an interrupt, which the HAE triggers on each result as the result is calculated and ready to transmit.	
		0	Disable
		1	Enable
1 (R/W)	LFREQ	Line Frequency Select. The `HAE_CFG0.LFREQ` bit specifies the line frequency for the HAE. Set the bit to match the line frequency being analyzed.	
		0	50 Hz
		1	60 Hz

Configuration 1 Register

The `HAE_CFG1` register configures the HAE frequency (clock) divider.

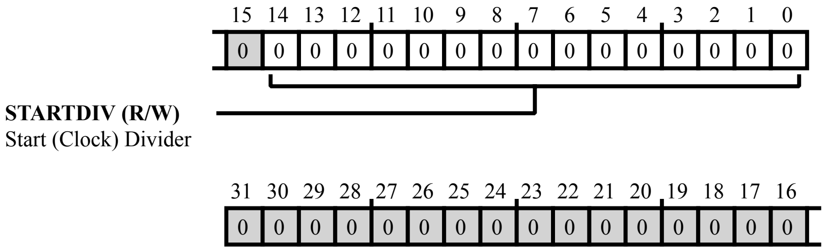

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STARTDIV (R/W)
Start (Clock) Divider
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-9: HAE_CFG1 Register Diagram

Table 49-9: HAE_CFG1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	STARTDIV	Start (Clock) Divider. The `HAE_CFG1.STARTDIV` bits provide the sample clock divider. Write the value to divide the main clock down to 8 kHz. The HAE sample loop is determined by SCLK0_0 / (`HAE_CFG1.STARTDIV`).

Configuration 2 Register

The `HAE_CFG2` register enables and configures HAE operations as related to output results.

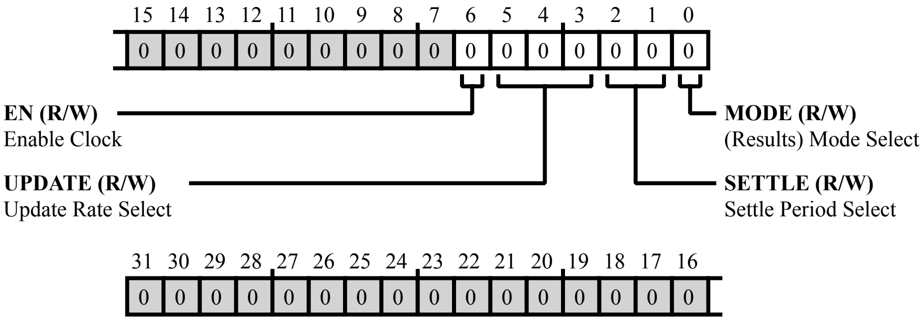

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EN (R/W)
Enable Clock
UPDATE (R/W)
Update Rate Select
MODE (R/W)
(Results) Mode Select
SETTLE (R/W)
Settle Period Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-10: HAE_CFG2 Register Diagram

Table 49-10: HAE_CFG2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	EN	Enable Clock. The `HAE_CFG2.EN` bit enables the HAE the main clock, enabling HAE operation.	
		0	Disable
		1	Enable
5:3 (R/W)	UPDATE	Update Rate Select. The `HAE_CFG2.UPDATE` bits determine the rate (in microseconds or milliseconds) at which the HAE updates the output results. The `HAE_CFG2.UPDATE` bits can also disable harmonic calculations.	
		0	125 uSec
		1	250 uSec
		2	1 mSec
		3	16 mSec
		4	128 mSec
		5	512 mSec
		6	1024 mSec
		7	Disable

Table 49-10: HAE_CFG2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:1 (R/W)	SETTLE	Settle Period Select. The `HAE_CFG2.SETTLE` bits determine the time (in milliseconds) that the HAE waits before producing results. The time is based on an 8K sample count.	
		0	512 mSec
		1	768 mSec
		2	1024 mSec
		3	1280 mSec
0 (R/W)	MODE	(Results) Mode Select. The `HAE_CFG2.MODE` bit determines whether the HAE produces results immediately or waits per the `HAE_CFG2.SETTLE` bit field.	
		0	Results after Settle
		1	Results First

Configuration 3 Register

The HAE_CFG3 register configures HAE data transfer operations by selecting the fundamental and potential of twelve additional harmonic channels. The selected harmonics have their data (results) transferred to memory using DMA. First, the fundamental; followed by the selected channel n, in the order from the lowest to the highest numbered channel. Each selected channel has its eight result words transferred.

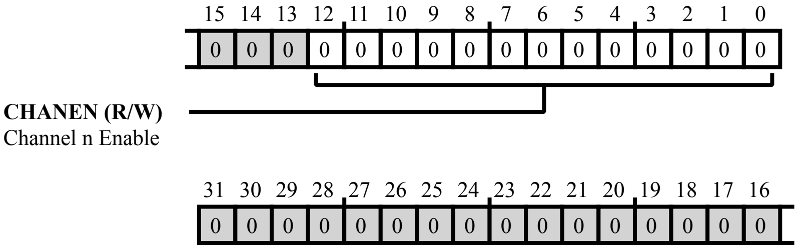

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CHANEN (R/W)
Channel n Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-11: HAE_CFG3 Register Diagram

Table 49-11: HAE_CFG3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12:0 (R/W)	CHANEN	Channel n Enable. Each `HAE_CFG3.CHANEN` bit enables the fundamental and potential of twelve additional harmonic data channels. The following enumerations apply to each bit. Bit 0 denotes the fundamental channel. Bits 1-12 denote the harmonic channels 1-12, accordingly.	
		0	Disable channel n
		8191	Enable channel n

Configuration 4 Register

The `HAE_CFG4` register configures the internal digital integrator and high-pass filters (HPS) for HAE operations. The digital integrator is disabled and the filters are enabled by default.

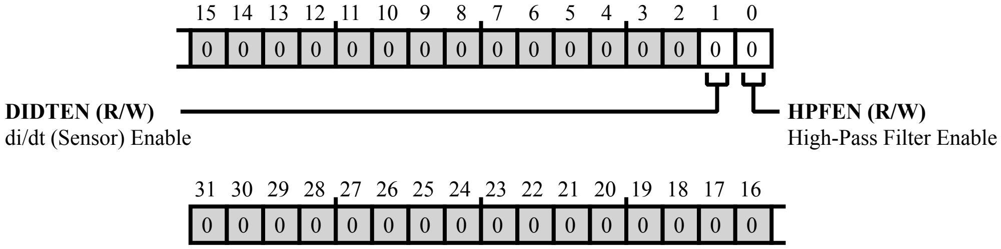

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DIDTEN (R/W)
di/dt (Sensor) Enable
HPFEN (R/W)
High-Pass Filter Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-12: HAE_CFG4 Register Diagram

Table 49-12: HAE_CFG4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DIDTEN	di/dt (Sensor) Enable. The `HAE_CFG4.DIDTEN` bit enables the internal digital integrator for a di/dt sensor. Set the bit (=1) if the sensor is a di/dt type sensor.	
		0	Disable
		1	Enable
0 (R/W)	HPFEN	High-Pass Filter Enable. The `HAE_CFG4.HPFEN` bit enables the high-pass filters. Set the bit (=1) unless measuring DC levels.	
		0	Disable
		1	Enable

DIDT Coefficient Register

The `HAE_DIDT_COEF` register sets the di/dt sensor’s coefficient.

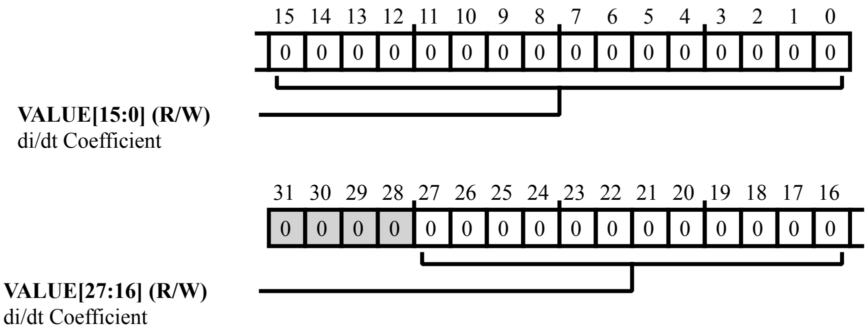

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
di/dt Coefficient
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[27:16] (R/W)
di/dt Coefficient

Figure 49-13: HAE_DIDT_COEF Register Diagram

Table 49-13: HAE_DIDT_COEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Coefficient. The `HAE_DIDT_COEF.VALUE` bits provide the coefficient for a di/dt type sensor. If the `HAE_CFG4.DIDTEN` bit is set (=1), set this bit field to 28'FFFF000.

DIDT Gain Register

The HAE_DIDT_GAIN register provides the di/dt sensor's gain.

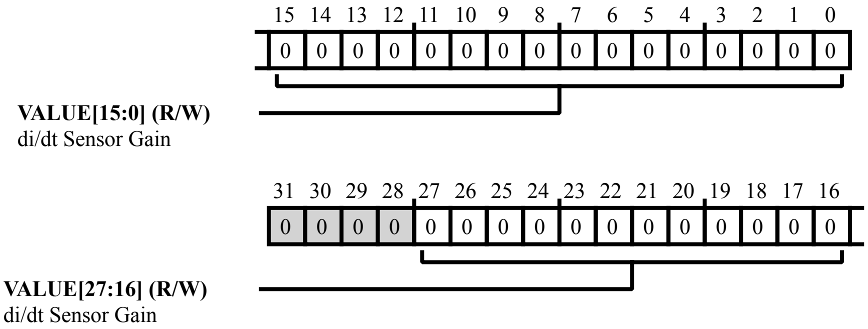

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
di/dt Sensor Gain
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[27:16] (R/W)
di/dt Sensor Gain

Figure 49-14: HAE_DIDT_GAIN Register Diagram

Table 49-14: HAE_DIDT_GAIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Sensor Gain. The `HAE_DIDT_GAIN.VALUE` bits provide the gain for a di/dt type sensor. If the `HAE_CFG4.DIDTEN` bit is set (=1), set this bit field to 0.

Harmonic n Index Register

The `HAE_H[nn]_INDX` registers select harmonics for HAE operations. The fundamental always is provided. The harmonic results appear in the results RAM memory region based on the index.

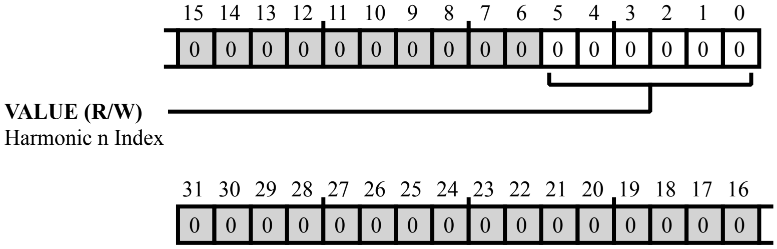

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Harmonic n Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-15: HAE_H[nn]_INDX Register Diagram

Table 49-15: HAE_H[nn]_INDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	Harmonic n Index. The `HAE_H[nn]_INDX.VALUE` bits select the harmonic channel n to analyze. Write the index of the harmonic into the `HAE_H[nn]_INDX` register for that harmonic to be selected.

I (Current) Sample Register

The `HAE_ISAMPLE` register provides current (I) input samples for HAE calculations.

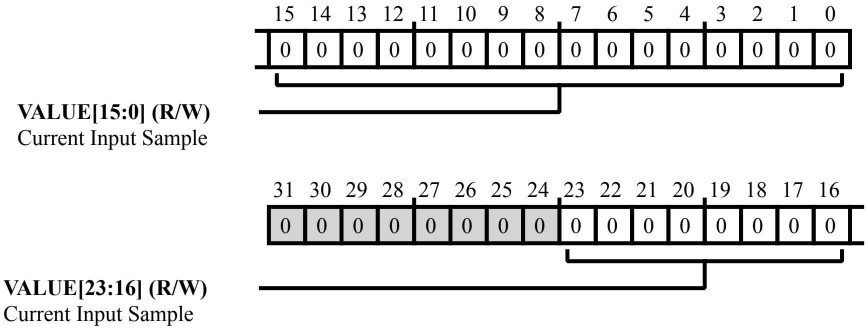

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Current Input Sample
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[23:16] (R/W)
Current Input Sample

Figure 49-16: HAE_ISAMPLE Register Diagram

Table 49-16: HAE_ISAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Current Input Sample. The `HAE_ISAMPLE.VALUE` bits hold the current sample if provided by the MCU. The sample can be timed with the `HAE_STAT.RXIRQ` bit.

I (Current) Waveform Register

The HAE_IWAVEFORM register holds processed current waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.

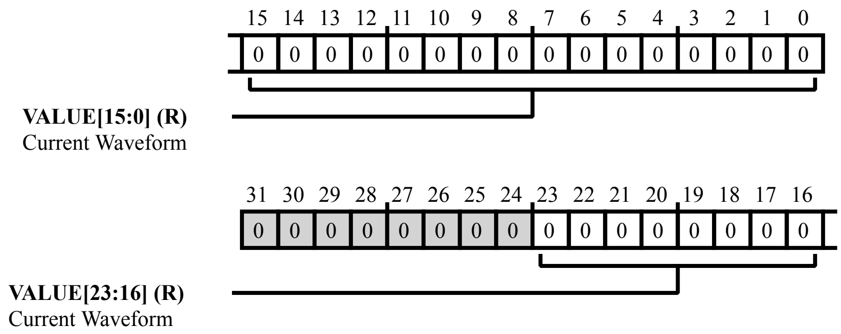

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Current Waveform
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[23:16] (R)
Current Waveform

Figure 49-17: HAE_IWAVEFORM Register Diagram

Table 49-17: HAE_IWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Current Waveform. The `HAE_IWAVEFORM.VALUE` bits hold the processed current input sample.

Run Register

The `HAE_RUN` register starts/idles HAE harmonic calculations.

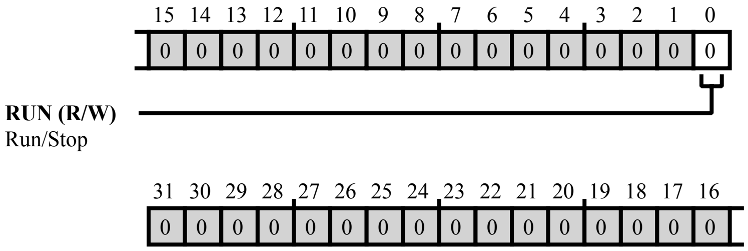

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RUN (R/W)
Run/Stop
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-18: HAE_RUN Register Diagram

Table 49-18: HAE_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	RUN	Run/Stop. The `HAE_RUN.RUN` bit starts the HAE harmonic calculations.	
		0	Stop
		1	Run

Status Register

The HAE_STAT register indicates status for the HAE module and its interrupts.

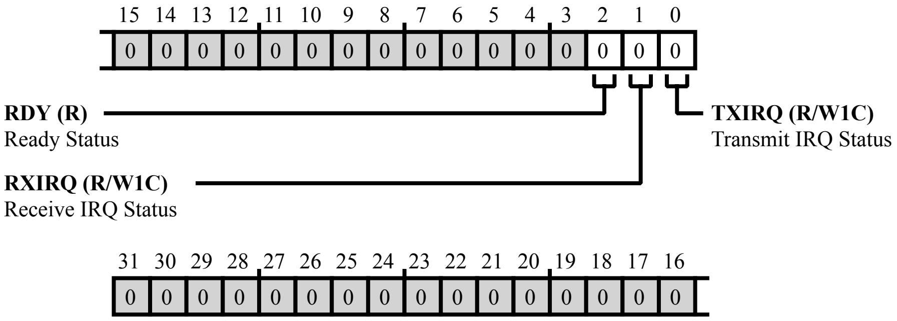

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RDY (R)
Ready Status
RXIRQ (R/W1C)
Receive IRQ Status
TXIRQ (R/W1C)
Transmit IRQ Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 49-19: HAE_STAT Register Diagram

Table 49-19: HAE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	RDY	Ready Status. The HAE_STAT.RDY bit indicates status for the HAE. When the bit is set (=1), the HAE is fully accessible, following the setting of the HAE_CFG2.EN bit.	
		0	No Status
		1	Ready
1 (R/W1C)	RXIRQ	Receive IRQ Status. The HAE_STAT.RXIRQ bit indicates status for an HAE RX interrupt. The bit mirrors the HAE_CFG0.RXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected
0 (R/W1C)	TXIRQ	Transmit IRQ Status. The HAE_STAT.TXIRQ bit indicates status for an HAE TX interrupt. The bit mirrors the HAE_CFG0.TXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected

Voltage Level Register

The `HAE_VLEVEL` register is used to scale the fixed fundamental voltage level internally. This assists the HAE in locking onto the fundamental voltage channel.

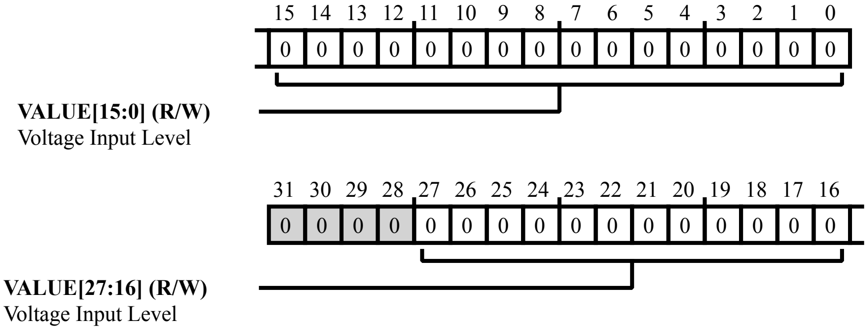

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Voltage Input Level
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[27:16] (R/W)
Voltage Input Level

Figure 49-20: HAE_VLEVEL Register Diagram

Table 49-20: HAE_VLEVEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	Voltage Input Level. The `HAE_VLEVEL.VALUE` bits hold a value to scale the fixed fundamental voltage level. Use this formula: VLEVEL = FS/V_{IN} * 5033168, where V_{IN} is the fundamental voltage input level.

V (Voltage) Sample Register

The HAE_VSAMPLE register provides voltage (V) input samples for HAE calculations.

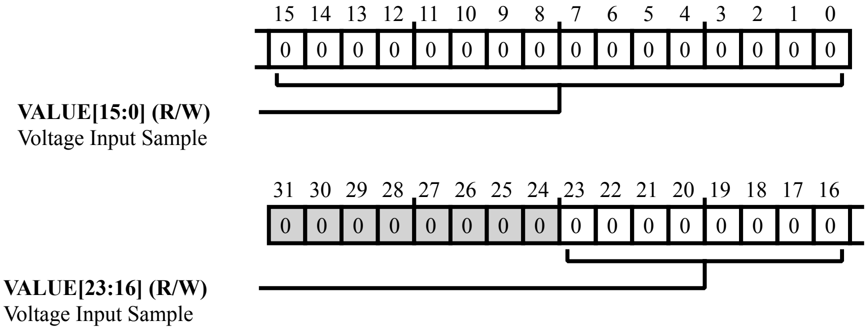

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Voltage Input Sample
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[23:16] (R/W)
Voltage Input Sample

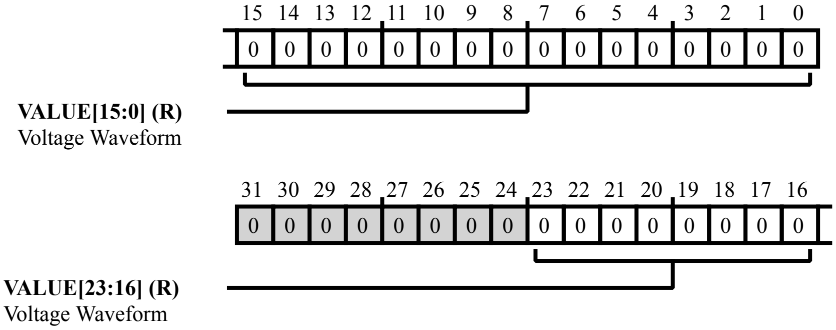
Figure 49-21: HAE_VSAMPLE Register Diagram

Table 49-21: HAE_VSAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Voltage Input Sample. The `HAE_VSAMPLE.VALUE` bits hold the voltage sample if provided by the MCU. The sample can be timed with the `HAE_STAT.RXIRQ` bit.

V (Voltage) Waveform Register

The `HAE_VWAVEFORM` register contains processed voltage waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Voltage Waveform
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[23:16] (R)
Voltage Waveform

Figure 49-22: HAE_VWAVEFORM Register Diagram

Table 49-22: HAE_VWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Voltage Waveform. The `HAE_VWAVEFORM.VALUE` bits hold the processed voltage input sample.

50 FFT Accelerator (FFTA)

The Fast Fourier Transform Accelerator (FFTA) performs memory to memory FFT/IFFT operations without core software intervention. Additionally, the FFTA architecture allows execution of complex, pipelined, memory to memory algorithms including ping-ponged, windowed frequency domain filtering and very large FFTs. The FFTA may also be used in conjunction with minimal computation support from a core in applications such as the overlap-add operations required for large frequency-domain-based convolutions.

FFTA Features

The following list describes the FFTA features.

- Supports both complex and real FFT and IFFT operations.
- Supports 64, 128, 256, 512, 1024, 2048 points in small FFT mode and 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304 points in large FFT mode.
- Supports the IEEE-754/854 single-precision floating-point data format, round to even.
- Radix-4 butterfly efficiency at a radix-2 (integer power of two) point granularity.
- Automatic insertion of zeros for real FFTs.
- Supports automatic conjugating of the twiddle factors for IFFT.
- Supports automatic scaling of FFT and IFFT inputs.
- Hardware support for windowing and frequency domain filtering.
- Hardware support for magnitude squared FFT output.
- Hardware support for pipelined data flow.
- Dedicated high speed DMA engines for data load and dump with a data width 64-bit clocked by SYSCLK.
- Supports data and coefficient access from both on-chip (L1/L2) and off-chip memories (L3).
- Optional support for bypassing the compute engine to perform high speed memory-to-memory MDMA transfers.
- Clock division options for power reductions. Supports 1:1, 1:2, 1:4 and 1:8 clock ratio modes.

FFTA Functional Description

The FFTA module provides the following functionality.

Complex and real FFT and IFFT operations

The inputs and outputs for the complex FFT routines are packed arrays of floating point numbers. In a packed array, the real and imaginary parts of each complex number are placed in alternate neighboring elements.

Hardware support for windowing, frequency domain filtering and pipelined data flow

Windowing is a technique used to shape the time portion of measurement data, to minimize edge effects that result in spectral leakage in the FFT spectrum. By using Window functions correctly, the spectral resolution of the frequency-domain result increases. Data pipelining helps to process one set of frames while other is being moved into and out of the accelerator via DMA.

Dedicated high speed DMA engines for data load and dump with a data width 64-bit is clocked by SYSCLK. This enables faster data movement between the memory internal to the accelerator and memories external to the accerator (L1/L2/L3).

Supports data and coefficient access from both on-chip (L1/L2) and off-chip memories (L3)

This functionality helps to process larger data buffers which cannot fit within the limited size of the L1/L2 memories.

ADSP-SC58x FFTA Register List

High-Speed FFT Compute Unit

Table 50-1: ADSP-SC58x FFTA Register List

Name	Description
FFTA_CTL	Control Register
FFTA_INST[nn]	Instruction Memory Register
FFTA_LC[nn]	Loop Counter Value Register
FFTA_PC	Program Counter Register
FFTA_SCALE	FFT/IFFT Scale Factor Register
FFTA_STAT	Status Register
FFTA_THREADOFFSET	Thread Count Offset Register
FFTA_WCTL	Wrapper Control Register
FFTA_XFRLEFT[nn]	Load/Dump Transfer Left Register

FFTA Definitions

To make the best use of the FFTA, it is useful to understand the following terms.

Small FFT

Small FFT corresponds to the FFT/IFFT operation where number of points are less than or equal to 2048. These operations can be accomplished directly with the local memory supported by the FFTA.

Large FFT

Large FFT corresponds to the FFT/IFFT operation where the number of points is greater than 2048. These operations are not directly supported by the FFTA because of the limited local memory. These are carried out in multiple stages using a divide and conquer approach by performing a number of small FFT operations.

CCES

CrossCore Embedded Studio

RTL

CrossCore Embedded Studio C/C++ Runtime Library

CCES RTL Manual

CrossCore Embedded Studio C/C++ Compiler and Library Manual for SHARC® Processors

ADSP-SC58x FFTA Interrupt List

Table 50-2: ADSP-SC58x FFTA Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
150	FFTA0_TXDMA	FFTA0 Transmit DMA		41
151	FFTA0_RXDMA	FFTA0 Receive DMA		42
152	FFTA0_STAT	FFTA0 Status		
210	FFTA0_TXDMA_ERR	FFTA0 Transmit DMA Error		
211	FFTA0_RXDMA_ERR	FFTA0 Receive DMA Error		

ADSP-SC58x FFTA Trigger List

Table 50-3: ADSP-SC58x FFTA Trigger List Masters

Trigger ID	Name	Description	Sensitivity
57	`FFTA0_TXDMA`	FFTA0 Transmit DMA	
58	`FFTA0_RXDMA`	FFTA0 Receive DMA	
59	`FFTA0_TRIGOUT`	FFTA0 Trigger Out	

Table 50-4: ADSP-SC58x FFTA Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
44	`FFTA0_TXDMA`	FFTA0 Transmit DMA	Pulse
45	`FFTA0_RXDMA`	FFTA0 Receive DMA	Pulse

FFTA Block Diagram

The FFTA contains a compute engine that operates in conjunction with a High Speed Distributed DMA Engine (HS-DDE). The compute engine appears as a data sink and data source to the two dedicated DDEs. The DDEs stream data into and out of the engine's FIFOs. To support faster data movement, the HS-DDE engine has a bus width of 64 bits and runs at SYSCLK speed. The compute engine data bus is 256 bits wide. The data transfer between the DMA engines and compute engine happens via data pack and unpack logic.

The FFTA compute engine runs at FFTCLK. The value of FFTCLK can go up to a maximum of SYSCLK. However, to achieve lower power consumption, it is also possible to operate the compute engine at frequencies lower than SYSCLK. This can be done by programming the FFTCLK:SYSCLK frequency ratio in the `FFTA_WCTL.FFTCLKRATIO` bit bield.

The two channel high speed DMA engines can also (optionally) be used for high speed memory to memory DMA (MDMA) transfer if the FFT compute engine is not used in the system.

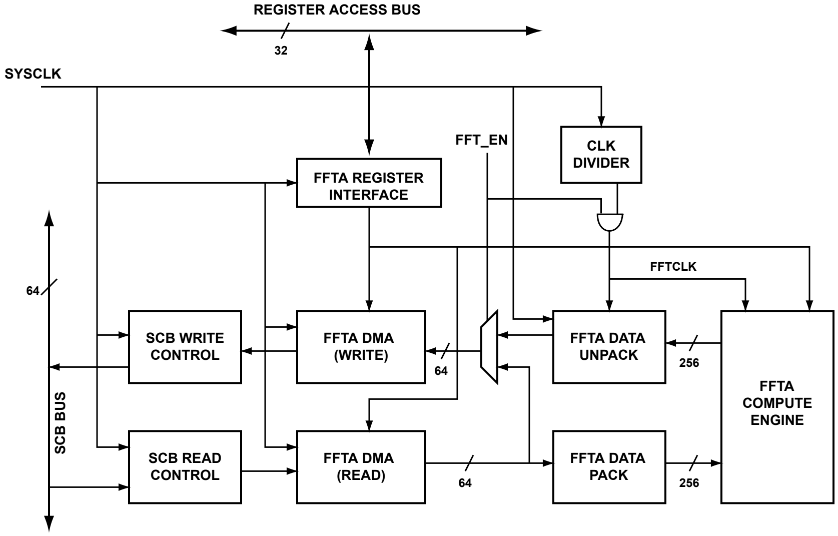

REGISTER ACCESS BUS
32
SYSCLK
FFT_EN
CLK DIVIDER
FFTA REGISTER INTERFACE
64
FFTCLK
SCB WRITE CONTROL
FFTA DMA (WRITE)
64
FFTA DATA UNPACK
256
FFTA COMPUTE ENGINE
SCB BUS
SCB READ CONTROL
FFTA DMA (READ)
64
FFTA DATA PACK
256

Figure 50-1: FFTA Block Diagram

FFTA Programming Model

A software library containing support for using the FFTA is provided with the CrossCore Embedded Studio. Using this library, C or C++ programs running on an ARM Cortex A5 or SHARC+ ADSP-215xx and ADSP-SC5xx core can access the FFTA to implement various specific use scenarios as discussed in the following sections. Analog Devices Inc. does not support programming the FFTA at register level (except for the optional programming of the `FFTA_WCTL` register). The detailed description of the FFTA registers is provided in this chapter for help with debugging.

FFTA Use Cases

The following section provides the conceptual description of the FFTA use cases supported by the CCES RTL APIs. For exact description on the programming model for using the APIs, refer to the *CrossCore Embedded Studio C/C++ Library Manual for SHARC Processors* manual.

1. Single-Shot FFT
2. Pipelined small FFT
3. Pipelined small interleaved FFT and IFFT operations

Single-Shot FFT

This use case is equivalent to the case where the core calls the CCES RTL's FFT/IFFT core library function. An API is called to perform an FFT/IFFT operation on a set of data and wait for the FFT/IFFT processing to finish. In this case all three units of the FFTA (input, compute, and output) operate in a sequential manner as illustrated by the ***Single-Shot*** figure. Notice how each frame passes through the stages of the FFTA block (input > compute > output) sequentially. The total time required to complete the FFT operation is equal to the time taken to load the input data plus time taken to compute the FFT/IFFT plus the time taken to dump the output data. CCES provides an FFTA version of these functions/APIs and the APIs supporting this use case always operate in synchronous mode.

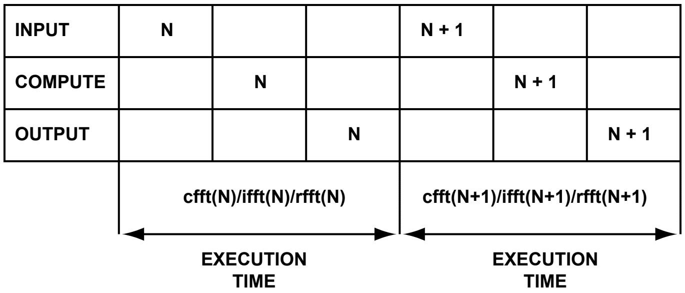

INPUT
COMPUTE
OUTPUT
N
N + 1
N
N + 1
N
N + 1
cfft(N)/ifft(N)/rfft(N)
cfft(N+1)/ifft(N+1)/rfft(N+1)
EXECUTION TIME
EXECUTION TIME

Figure 50-2: Single-Shot

Pipelined Small FFT

The piplined small FFT is used where an FFT/IFFT operation with a fixed number of points is performed continuously on more than one set of input data. The FFTA architecture supports pipelining of the input and output data which allows the FFTA to load one set of input data and dump another set of processed output data in parallel while another set of data (fetched in the previous pipeline cycle) is being processed. This helps to suppress the input and output DMA overheads while the FFTA compute engine is busy processing the data. This way, programs can take maximum advantage of the FFT compute engine's performance.

To support such cases, the FFTA APIs provide a mechanism where the FFTA is configured one time to perform the FFT/IFFT operation with a fixed number of points in a continuous pipelined manner. Next, the APIs related to the data transfers (asynchronous) are called to send the input data to and collect the output data from the FFTA. After all the data is processed, the FFTA can be closed. For more details on these APIs, refer to the CCES RTL manual.

The ***Pipelined Small FFT*** figure illustrates this use case. There is large startup latency (Input DMA time + Compute time + Output DMA time) involved to fill the pipeline with three frames. After that, each frame can be processed in a lesser steady state time (maximum of Input DMA time, Compute time, and Output DMA time).

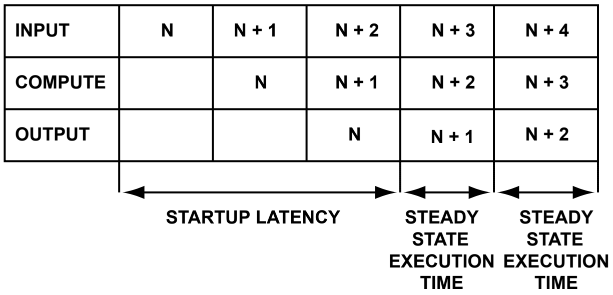

INPUT
COMPUTE
OUTPUT
N
N + 1
N + 2
N + 3
N + 4
N
N + 1
N + 2
N + 3
N
N + 1
N + 2
STARTUP LATENCY
STEADY STATE EXECUTION TIME
STEADY STATE EXECUTION TIME

Figure 50-3: Pipelined Small FFT

Pipelined Small Interleaved FFT and IFFT Operations

Pipelined small interleaved FFT and IFFT operations are useful when data is first converted to the frequency domain, processed in the frequency domain and converted back to the time domain. The ***Typical Data Flow***

Frequency Domain Processing figure illustrates this where a particular input data frame N is first converted to the frequency domain (N’) with an FFT operation. This FFT output is then processed in the frequency domain to produce the output frame N”. The processed output frame (N”) is then converted in to time domain output frame (N’”).

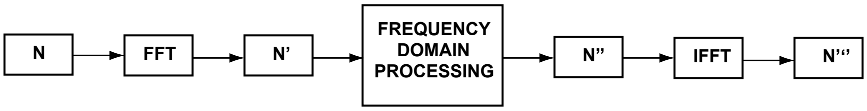
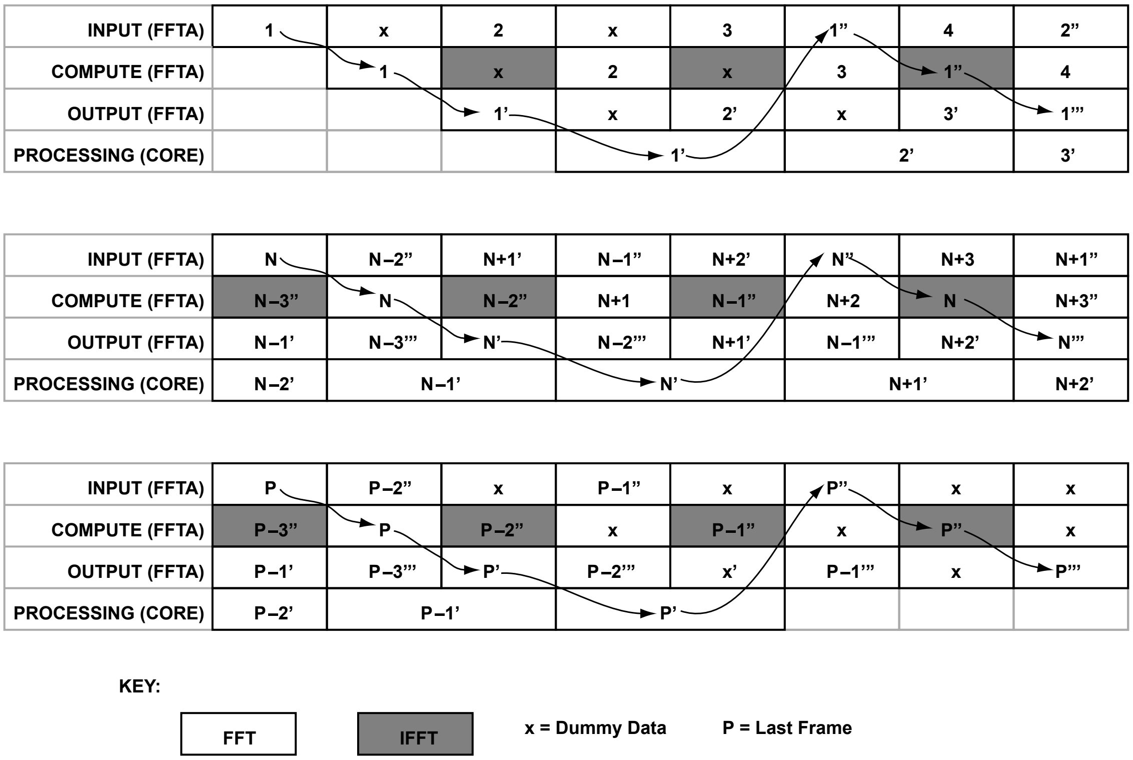


Figure 50-4: Typical Data Flow Frequency Domain Processing

The FFTA can be used to perform the above operation on multiple sets of data in a pipelined manner as shown in the ***Pipelined Interleaved FFT/IFFT*** figure. Unlike in the Pipelined Small FFT section, the compute engine performs both FFT and IFFT operations together in an interleaved manner. Note that the input data for the first valid IFFT operation (1”) is available only when the FFT output of the first frame (1’) is processed by the core. It is necessary to send dummy input data at the start of the pipeline for the initial few IFFT operations as shown in the ***Pipelined Interleaved FFT/IFFT*** figure. The output of these IFFT operations can be ignored. Similarly, dummy data must be sent at the end of the pipeline for the last few FFT operations. The output of these FFT operations can be ignored.



INPUT (FFTA)	1	x	2	x	3	1”	4	2”
COMPUTE (FFTA)		1	x	2	x	3	1”	4
OUTPUT (FFTA)			1’	x	2’	x	3’	1’”
PROCESSING (CORE)				1’		2’		3’

INPUT (FFTA)	N	N–2”	N+1’	N–1”	N+2’	N”	N+3	N+1”
COMPUTE (FFTA)	N–3”	N	N–2”	N+1	N–1”	N+2	N	N+3”
OUTPUT (FFTA)	N–1’	N–3’”	N’	N–2’”	N+1’	N–1’”	N+2’	N’”
PROCESSING (CORE)	N–2’	N–1’		N’		N+1’		N+2’

INPUT (FFTA)	P	P–2”	x	P–1”	x	P”	x	x
COMPUTE (FFTA)	P–3”	P	P–2”	x	P–1”	x	P”	x
OUTPUT (FFTA)	P–1’	P–3’”	P’	P–2’”	x’	P–1’”	x	P’”
PROCESSING (CORE)	P–2’	P–1’		P’				

Figure 50-5: Pipelined Interleaved FFT/IFFT

Similar to the Pipelined Small FFT section, the FFTA APIs provide a mechanism where the FFTA can be configured once to perform interleaved FFT and IFFT operations with a fixed number of points in a continuous pipelined manner. After that, programs just need to call the APIs related only to the data transfers (asynchronous) to send the input data to and collect the output data from the FFTA. After all the data is processed, the FFTA can be closed. For more details on these APIs, refer to the CCES RTL manual.

Fast Convolution

The FFTA can also be used to perform FIR filtering with the help of fast convolution. In this method, FFTA converts the input data in frequency domain, multiplies it with the FIR coefficients pre-transformed in the frequency domain, and then performs IFFT to return output in the time domain. The core is responsible for maintaining the output delay line and performing overlap-add operation.

For example, the ***FFTA Block Diagram*** shows how a block of 512 input samples is processed with an FIR filter of 8192 taps with the help of fast convolution method.

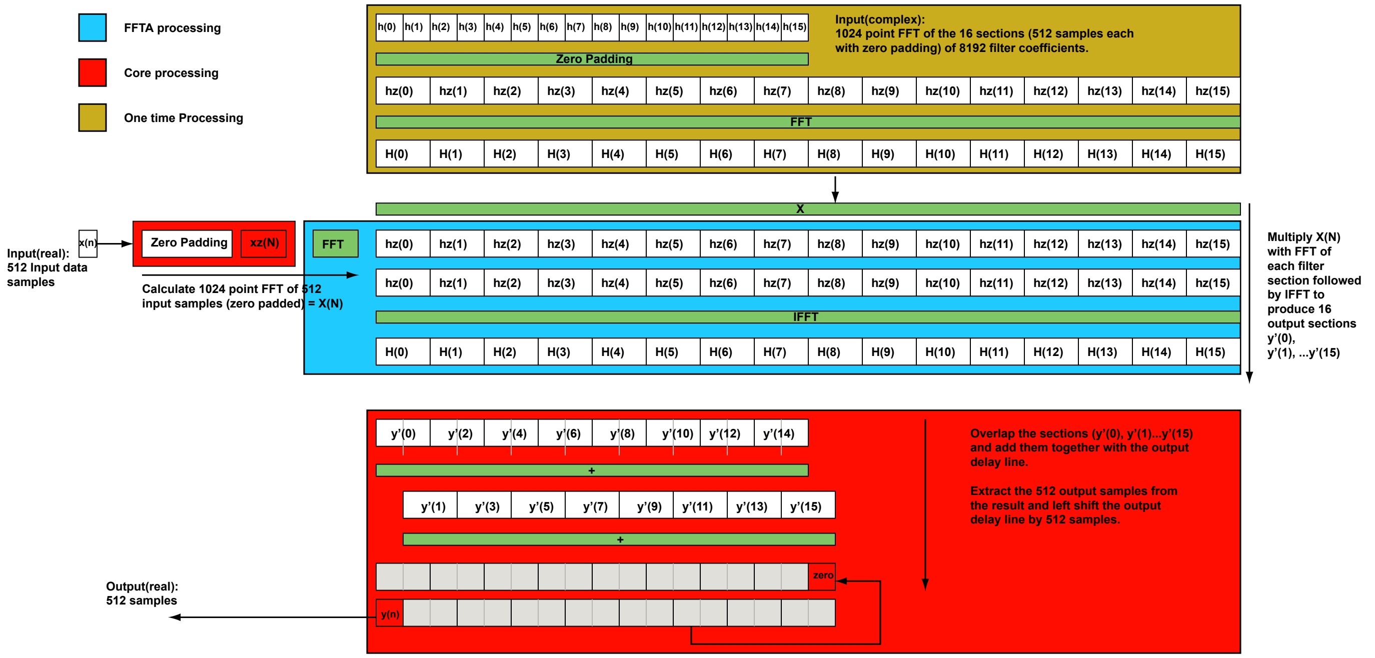

FFTA processing
Core processing
One time Processing
h(0) h(1) h(2) h(3) h(4) h(5) h(6) h(7) h(8) h(9) h(10) h(11) h(12) h(13) h(14) h(15)
Input(complex):
1024 point FFT of the 16 sections (512 samples each with zero padding) of 8192 filter coefficients.
Zero Padding
hz(0) hz(1) hz(2) hz(3) hz(4) hz(5) hz(6) hz(7) hz(8) hz(9) hz(10) hz(11) hz(12) hz(13) hz(14) hz(15)
FFT
H(0) H(1) H(2) H(3) H(4) H(5) H(6) H(7) H(8) H(9) H(10) H(11) H(12) H(13) H(14) H(15)
X
Input(real):
512 Input data samples
x(n)
Zero Padding
xz(N)
FFT
Calculate 1024 point FFT of 512 input samples (zero padded) = X(N)
hz(0) hz(1) hz(2) hz(3) hz(4) hz(5) hz(6) hz(7) hz(8) hz(9) hz(10) hz(11) hz(12) hz(13) hz(14) hz(15)
hz(0) hz(1) hz(2) hz(3) hz(4) hz(5) hz(6) hz(7) hz(8) hz(9) hz(10) hz(11) hz(12) hz(13) hz(14) hz(15)
IFFT
H(0) H(1) H(2) H(3) H(4) H(5) H(6) H(7) H(8) H(9) H(10) H(11) H(12) H(13) H(14) H(15)
Multiply X(N) with FFT of each filter section followed by IFFT to produce 16 output sections y'(0), y'(1), ...y'(15)
y'(0) y'(2) y'(4) y'(6) y'(8) y'(10) y'(12) y'(14)
+
y'(1) y'(3) y'(5) y'(7) y'(9) y'(11) y'(13) y'(15)
+
zero
y(n)
Output(real):
512 samples
Overlap the sections (y'(0), y'(1)...y'(15) and add them together with the output delay line.
Extract the 512 output samples from the result and left shift the output delay line by 512 samples.

Figure 50-6: FFTA Block Diagram

The steps involved are as follows:

1. The FIR coefficients are divided in to 16 (=8192/512) sections h(0), h(1), h(2),…..h(15). Each section is padded with 512 zeros to generate hz(0), hz(1), hz(2),….hz(15) and then converted into the frequency domain (1024 point FFT) to generate H(0), H(1), H(2),….H(15).
2. Each time a new block of 512 samples x(n) is received, it is first padded with 512 zeros and the converted to frequency domain (1024 point FFT) to generate X(N).
3. X(N) is then multiplied with each of the 16 sections H(0), H(1), H(2),….H(15) to generate X(N)H(0), X(N)H(1), X(N)H(2),…. X(N)H(15).
4. All the 16 blocks of the result in 3. are then converted in to time domain (1024 point IFFT) to generate the output sections y'(0), y'(1),y'(2)……y'(15) each containing 1024 output samples.

5. The output sections generated in 4. are overlapped and added with the existing output delay line.
6. The left most 512 samples from the output delay line generated in 6. are extracted to generate 512 output samples corresponding to the original 512 input samples.
7. The delay line is then shifted left by 512 samples which is used as an input for the next processing iteration.

ADSP-SC58x FFTA Register Descriptions

High-Speed FFT Compute Unit (FFTA) contains the following registers.

Table 50-5: ADSP-SC58x FFTA Register List

Name	Description
`FFTA_CTL`	Control Register
`FFTA_INST[nn]`	Instruction Memory Register
`FFTA_LC[nn]`	Loop Counter Value Register
`FFTA_PC`	Program Counter Register
`FFTA_SCALE`	FFT/IFFT Scale Factor Register
`FFTA_STAT`	Status Register
`FFTA_THREADOFFSET`	Thread Count Offset Register
`FFTA_WCTL`	Wrapper Control Register
`FFTA_XFRLEFT[nn]`	Load/Dump Transfer Left Register

Control Register

The FFTA_CTL register is used to reset, start, pause and single step the HPFP-FFT engine.

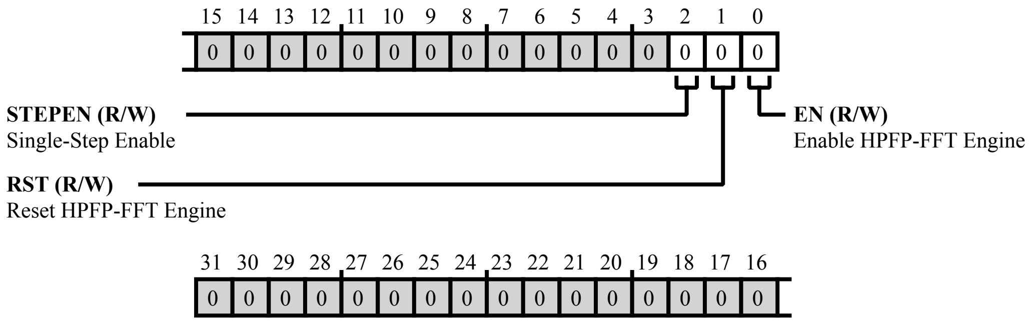

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STEPEN (R/W)
Single-Step Enable
RST (R/W)
Reset HPFP-FFT Engine
EN (R/W)
Enable HPFP-FFT Engine
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 50-7: FFTA_CTL Register Diagram

Table 50-6: FFTA_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	STEPEN	Single-Step Enable. When the FFTA_CTL.STEPEN and FFTA_CTL.EN bits are set together the engine to release the instruction currently pointed to by the PC to execution, increment the PC then stop execution. The FFTA_CTL.STEPEN and FFTA_CTL.EN bits are automatically cleared when the released instruction completes execution. The FFTA_CTL.EN bit or FFTA_CTL.STEPEN bit can then be read to determine if the instruction step has completed. Note that a single step advances the PC one increment and at least one thread completes execution but one or two other threads may still not have completed execution. The FFTA_CTL.STEPEN control bit is self-clearing.	
		0	Single-Step Disabled
		1	Single-Step Enabled
1 (R/W)	RST	Reset HPFP-FFT Engine. When the FFTA_CTL.RST bit is set all MMRs are cleared. The FFTA_CTL.RST bit is self-clearing.	
		0	Not reset HPFP-FFT engine
		1	Reset HPFP-FFT engine
0 (R/W)	EN	Enable HPFP-FFT Engine. The FFTA_CTL.EN bit must be set for the engine to process instructions. When the FFTA_CTL.EN bit is cleared the engine is in an idle state.	
		0	Disable HPFP-FFT engine
		1	Enable HPFP-FFT engine

Instruction Memory Register

The `FFTA_INST[nn]` MMR registers hold (up to 64) FFT instructions.

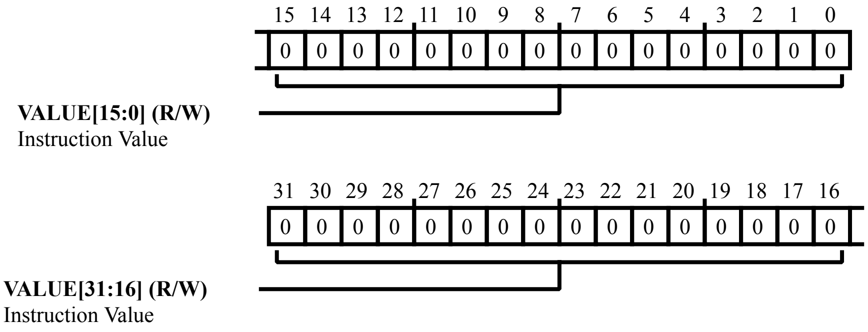

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Instruction Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Instruction Value

Figure 50-8: FFTA_INST[nn] Register Diagram

Table 50-7: FFTA_INST[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Instruction Value. The `FFTA_INST[nn].VALUE` bit field contains 64 instruction MMR to hold FFT instructions.

Loop Counter Value Register

The `FFTA_LC[nn]` register holds the current value of the corresponding loop counter.

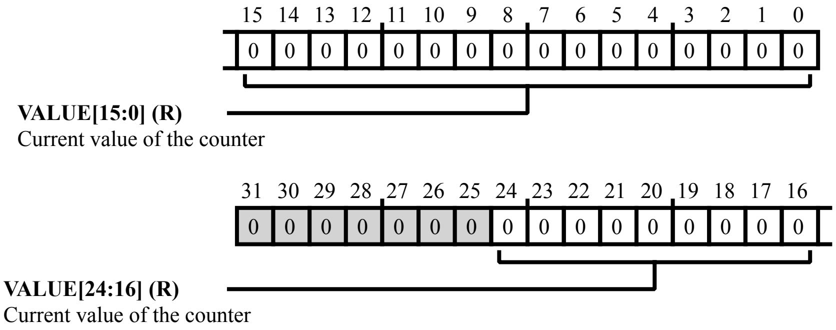

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Current value of the counter
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[24:16] (R)
Current value of the counter

Figure 50-9: FFTA_LC[nn] Register Diagram

Table 50-8: FFTA_LC[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:0 (R/NW)	VALUE	Current value of the counter.

Program Counter Register

The FFTA_PC register contains the MMR address offset of the current instruction in the instruction queue. The PC advances unless the instruction it points to is not acknowledged. When the instruction pointed to is acknowledged it starts the task and immediately advances. See the rules for instruction acknowledge in the Sequencer section of the Programming reference.

Note that the nop(), jumpCNZ(), load_loop_cntr(), incr_thread_offset() and load_scale() instructions are immediately acknowledged. Also note that only 6 bits are used, which implies that this counter is incremented in the fashion of modulo 64. For example, If the current PC=63, after it is incremented by 1, PC=0.

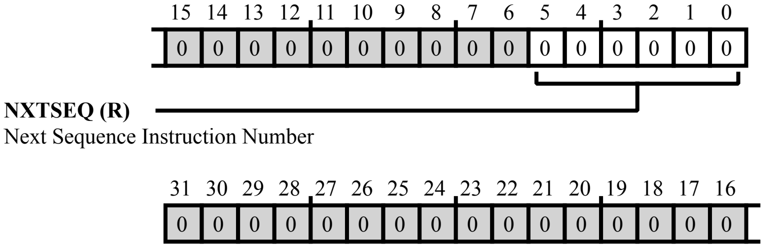


Figure 50-10: FFTA_PC Register Diagram

Table 50-9: FFTA_PC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/NW)	NXTSEQ	Next Sequence Instruction Number. The FFTA_PC.NXTSEQ bits provide the value by which the PC advances unless the instruction it points to is not acknowledged. When the instruction pointed to is acknowledged it starts the task and immediately advances. See the rules for instruction acknowledgement in the Sequencer section. The nop(), jumpCNZ(), load_loop_cntr(), incr_thread_offset() and load_scale() instructions are immediately acknowledged. Note that only 6 bits are used, which implies that this counter is incremented in the fashion of modulo 64. For example, If the current PC=63, after it is incremented by 1, PC=0.

FFT/IFFT Scale Factor Register

The FFTA_SCALE register is loaded using the load_scale() instruction. The scale factor is in a single-precision IEEE format. On reset the SCALE is set to 1.0 (floating point).

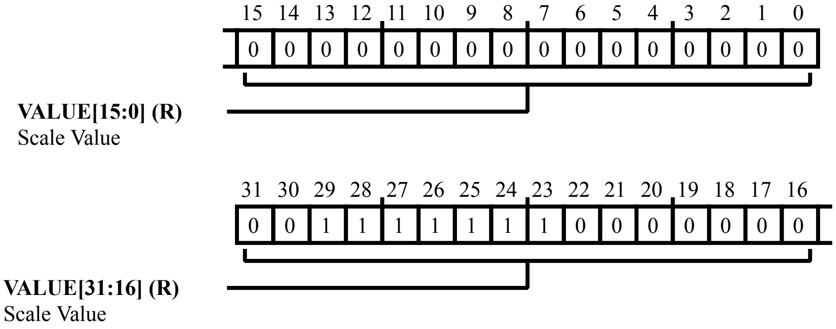

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Scale Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0
VALUE[31:16] (R)
Scale Value

Figure 50-11: FFTA_SCALE Register Diagram

Table 50-10: FFTA_SCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Scale Value. Reset to 1.0

Status Register

The FFTA_STAT register indicates the status of FFT operations. Sticky status flags remain set until cleared by reset. In the status description the term math operation or math result refers to all multiply, add and subtract results including intermediate operations.

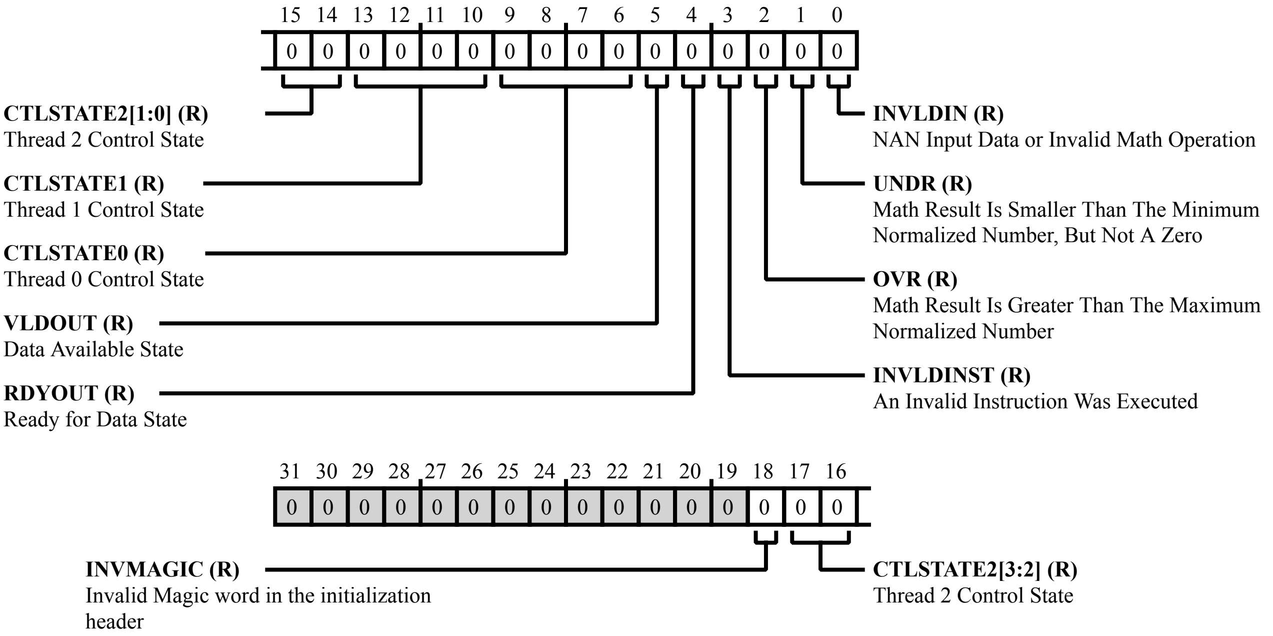

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CTLSTATE2[1:0] (R)
Thread 2 Control State
CTLSTATE1 (R)
Thread 1 Control State
CTLSTATE0 (R)
Thread 0 Control State
VLDOUT (R)
Data Available State
RDYOUT (R)
Ready for Data State
INVLDIN (R)
NAN Input Data or Invalid Math Operation
UNDR (R)
Math Result Is Smaller Than The Minimum Normalized Number, But Not A Zero
OVR (R)
Math Result Is Greater Than The Maximum Normalized Number
INVLDINST (R)
An Invalid Instruction Was Executed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INVMAGIC (R)
Invalid Magic word in the initialization header
CTLSTATE2[3:2] (R)
Thread 2 Control State

Figure 50-12: FFTA_STAT Register Diagram

Table 50-11: FFTA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/NW)	INVMAGIC	Invalid Magic word in the initialization header. Invalid magic word in the initialization header.	
		0	Magic word is ok
		1	Magic word is not ok
17:14 (R/NW)	CTLSTATE2	Thread 2 Control State. The FFTA_STAT.CTLSTATE2 bit indicates the data available state (reflects state of fft.valid_o).	
		0	idle state
		1	load coef state
		2	load data state
		3	dump data state
		4	fft state

Table 50-11: FFTA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	multiply state
		6	maganitude squared state
		7	generate twiddles state
		8	bit reversal state
13:10 (R/NW)	CTLSTATE1	Thread 1 Control State. The `FFTA_STAT.CTLSTATE1` bit indicates the data available state (reflects state of fft.valid_o).	
		0	idle state
		1	load coef state
		2	load data state
		3	dump data state
		4	fft state
		5	multiply state
		6	maganitude squared state
		7	generate twiddles state
		8	bit reversal state
9:6 (R/NW)	CTLSTATE0	Thread 0 Control State. The `FFTA_STAT.CTLSTATE0` bit indicates the data available state (reflects state of fft.valid_o).	
		0	idle state
		1	load coef state
		2	load data state
		3	dump data state
		4	fft state
		5	multiply state
		6	maganitude squared state
		7	generate twiddles state
		8	bit reversal state

Table 50-11: FFTA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	VLDOUT	Data Available State. The `FFTA_STAT.VLDOUT` bit indicates that data is available in the OFIFO to be read out (reflects state of fft.valid_o).	
		0	Data Not Available
		1	Data Available
4 (R/NW)	RDYOUT	Ready for Data State. The `FFTA_STAT.RDYOUT` bit indicates that the IFIFO is ready to receive data (reflects state of fft.ready_o).	
		0	Data Out Not Ready
		1	Data Out Ready
3 (R/NW)	INVLDINST	An Invalid Instruction Was Executed. The `FFTA_STAT.INVLDINST` bit indicates an invalid instruction was executed. Asserts a FFT_I interrupt output when the INVALIDINST flag is first set. This INVAILIDINST flag is sticky.	
		0	No Invalid Instruction
		1	An Invalid Instruction Was Executed
2 (R/NW)	OVR	Math Result Is Greater Than The Maximum Normalized Number. The `FFTA_STAT.OVR` bit indicates a math result after rounding has a magnitude greater than the maximum normalized number. Asserts a FFT_I interrupt output when the OFLOW flag is first set. This OFLOW flag is sticky.	
		0	No Overflow
		1	Overflow
1 (R/NW)	UNDR	Math Result Is Smaller Than The Minimum Normalized Number, But Not A Zero. The `FFTA_STAT.UNDR` bit indicates a math result after rounding has a magnitude less than the minimum normalized number, and it is not an exact zero. Asserts a FFT_I interrupt output when the UFLOW flag is first set. This UFLOW flag is sticky.	
		0	No Underflow
		1	Underflow
0 (R/NW)	INVLDIN	NAN Input Data or Invalid Math Operation. The `FFTA_STAT.INVLDIN` bit indicates a NAN input data or invalid math operation: (0 x infinity, infinity - infinity). Asserts FFT_I interrupt output when the INVALIDIN flag is first set. This INVALIDIN flag is sticky.	
		0	Input And Math Opererations Are Valid
		1	Input And/Or Math Opererations Are Invalid

Thread Count Offset Register

The FFTA_THREADOFFSET register is a modulo 3 added to an instruction's specified thread number to create the actual thread assignment. The incr_thread_offset() instruction will modulo 3 increment the thread offset register.

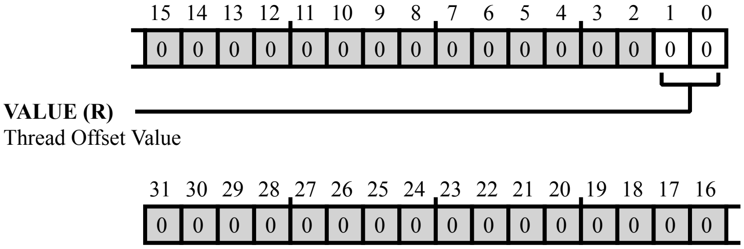

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Thread Offset Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 50-13: FFTA_THREADOFFSET Register Diagram

Table 50-12: FFTA_THREADOFFSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/NW)	VALUE	Thread Offset Value.

Wrapper Control Register

The `FFTA_WCTL` register contains bits that configure the FFTCLK divider value and enable the FFT engine.

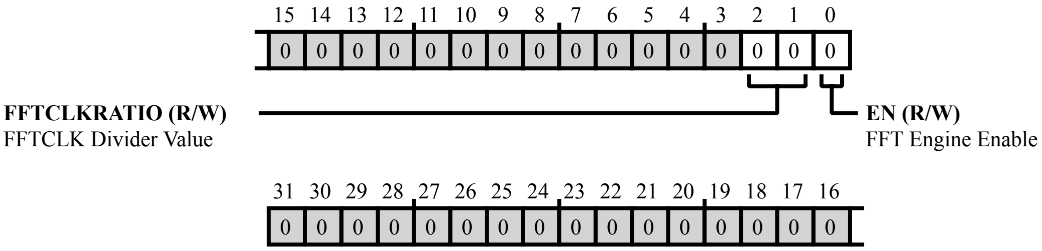

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FFTCLKRATIO (R/W)
FFTCLK Divider Value
EN (R/W)
FFT Engine Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 50-14: FFTA_WCTL Register Diagram

Table 50-13: FFTA_WCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:1 (R/W)	FFTCLKRATIO	FFTCLK Divider Value. The `FFTA_WCTL.FFTCLKRATIO` bits provide the FFTCLK divider value which selects the FFTCLK:SYSCLK_0 frequency ratio.	
		0	FFTCLK:SYSCLK_0 = 1:1
		1	FFTCLK:SYSCLK_0 = 1:2
		2	FFTCLK:SYSCLK_0= 1:4
		3	FFTCLK:SYSCLK_0 = 1:8
0 (R/W)	EN	FFT Engine Enable. The `FFTA_WCTL.EN` bit, when set (=1) enables the FFT engine.	
		0	Disable FFT engine and enable MDMA
		1	Enable FFT engine and Disable MDMA

Load/Dump Transfer Left Register

The three `FFTA_XFRLEFT[nn]` registers, one for each thread, contain the number of 64-bit data or coefficient loads or data dumps remaining if a load or dump instruction is executing in that thread. The value of each register ranges from 0 to 1023, which means (Load/dump transfers remaining for thread)/8. So load/dump remaining can cover the range of [0, 1023*8]=[0, 8184]. If no load or dump is executing in that thread then the value is 0. The values reflect the count at the start of the pipeline (stage N0).

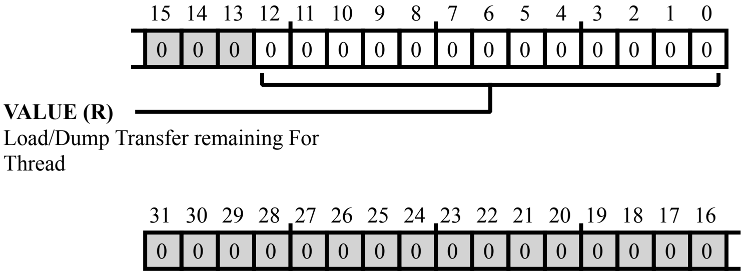

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Load/Dump Transfer remaining For Thread
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 50-15: FFTA_XFRLEFT[nn] Register Diagram

Table 50-14: FFTA_XFRLEFT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/NW)	VALUE	Load/Dump Transfer remaining For Thread.

51 FIR Accelerator (FIR)

Finite Impulse Response (FIR) filters are frequently used in DSP applications. The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core. FIR filters are used in a wide array of applications including multi-rate processing with an interpolator or decimator.

Features

This hardware module can perform FIR filters without core intervention. This feature gives programs freedom to use the core to implement complex algorithms, effectively adding more bandwidth to the processor. The FIR supports the following features:

- Fixed-point and 32-bit IEEE floating-point format
- Four MAC units that operate in parallel
- Various rounding modes
- Single rate or multi-rate window processing
- Programmable rates with decimation or interpolation mode
- Up to 32 filter channels available in TDM

NOTE: The FIR accelerator module has local memory that the core cannot access during regular operation. Unlike previous SHARC processors, the FIR accelerator modules each have access to the system memory (on-chip or off-chip).

Also, unlike in previous SHARC processors, where only one of the FIR or IIR accelerators can be used at a time, the SHARC+ processor can use both of these accelerators simultaneously.

Clocking

The FIR accelerator runs at the maximum speed of the system clock frequency (SCLK0).

Functional Description

The ***FIR Block Diagram*** shows the block diagram of the 1024-TAP FIR hardware accelerator. The accelerator consists of a 1024 word coefficient memory, a 1024 deep delay line for data, and four MAC units. The accelerator runs at the *SCLK0* frequency.

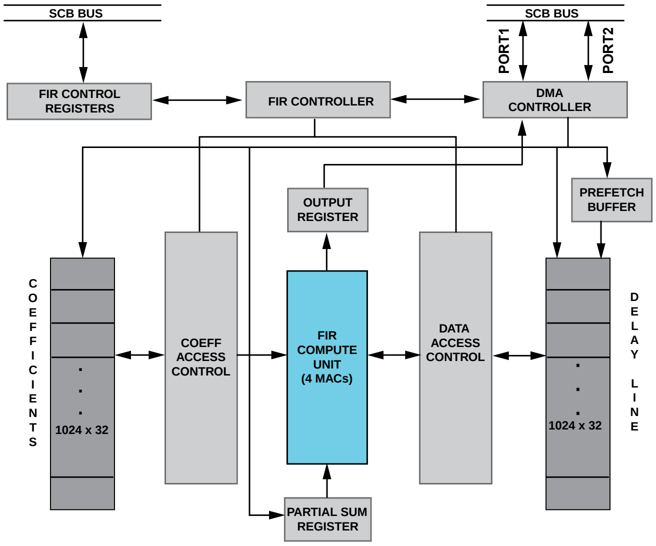

SCB BUS
SCB BUS
PORT1
PORT2
FIR CONTROL REGISTERS
FIR CONTROLLER
DMA CONTROLLER
OUTPUT REGISTER
PREFETCH BUFFER
COEFFICIENTS
1024 x 32
COEFF ACCESS CONTROL
FIR COMPUTE UNIT (4 MACs)
DATA ACCESS CONTROL
1024 x 32
DELAY LINE
PARTIAL SUM REGISTER

Figure 51-1: FIR Block Diagram

The FIR accelerator has the following logical sub blocks:

1. A datapath unit that consists of:
    - A 1024 deep coefficient memory
    - A 1024 deep delay line for the data
    - Four 32-bit floating-point and fixed-point multiplier and adder units
    - One 32-bit prefetch buffer to operate in a pipelined fashion
    - One 32-bit buffer to hold previous partial sum
    - One 32-bit buffer to hold the output
2. Configuration registers for the number of TAPs, number of channels, filter enable, interrupt control, DMA enable, up sample or down sample control, and ratios.
3. Core access interface for writing to the DMA and filter configuration registers and reading the status register.
4. DMA bus interface for transferring data and coefficients to and from the accelerator.
5. DMA configuration registers including chain pointer, input, output, and coefficient registers.

ADSP-SC58x FIR Register List

The FIR accelerator is a dedicated hardware interface used to perform filter processing to reduce the instruction processing load on the core.

Table 51-1: ADSP-SC58x FIR Register List

Name	Description
`FIR_CHNPTR`	FIR Chain Pointer Register
`FIR_COEFCNT`	FIR Coefficient Count Register
`FIR_COEFIDX`	FIR Coefficient Index Register
`FIR_COEFMOD`	FIR Coefficient Modifier Register
`FIR_CTL1`	FIR Global Control Register
`FIR_CTL2`	FIR Channel Control Register
`FIR_DBG_ADDR`	Debug Address Register
`FIR_DBG_CTL`	FIR Debug Control Register
`FIR_DBG_RDDAT`	FIR Debug Data Read Register
`FIR_DBG_WRDAT`	FIR Debug Data Write Register
`FIR_DMASTAT`	FIR DMA Status Register
`FIR_INBASE`	FIR Input Data Base Register
`FIR_INCNT`	FIR Input Data Count Register
`FIR_INIDX`	FIR Input Data Index Register
`FIR_INMOD`	FIR Input Data Modifier Register
`FIR_MACSTAT`	FIR MAC Status Register
`FIR_OUTBASE`	FIR Output Data Base Register
`FIR_OUTCNT`	FIR Output Data Count Register
`FIR_OUTIDX`	FIR Output Data Index Register
`FIR_OUTMOD`	FIR Output Data Modifier Register

ADSP-SC58x FIR Interrupt List

Table 51-2: ADSP-SC58x FIR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
153	FIR0_DMA	FIR0 DMA	Edge	
154	FIR0_STAT	FIR0 Status	Edge	

ADSP-SC58x FIR Trigger List

Table 51-3: ADSP-SC58x FIR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
60	`FIR0_DMA`	FIR0 DMA	Edge

Table 51-4: ADSP-SC58x FIR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

Compute Block

The MAC unit, shown in the ***FIR MAC Unit*** figure, has four multiply accumulators. The accumulators operate simultaneously on a single filter as described as follows:

- The MAC unit operates on the data and coefficient fetched from the data and coefficient RAMs
- Each MAC can perform 32-bit floating-point or 32-bit fixed-point MAC operations
- Floating-point format is IEEE-compliant
- Multiply and accumulation operation (addition) are pipelined
- A 32-bit floating-point MAC operation generates 32-bit multiply results
- A 32-bit fixed-point operation generates 80-bit results (64-bit result + 16 guard bits)

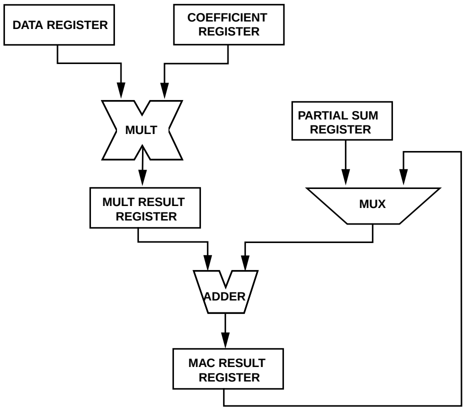


Figure 51-2: FIR MAC Unit

Partial Sum Register

The partial sum register is useful for Floating-point Multi-Iteration mode. For a particular channel, the intermediate MAC result is written to the output buffer of the system (on-chip or off-chip) memory. If the same channel is requested again, the partial result register is updated with the intermediate MAC result through DMA from output buffer of the system memory. The result is added to the current MAC result after each iteration. This process repeats until all iterations complete (the entire soft filter length is processed).

Delay Line Memory

The accelerator has a 1024 TAP delay line to hold the data locally. The DMA controller fetches the data from system memory and loads it into the delay line. Four read accesses can be made to the delay line simultaneously.

Coefficient Memory

The accelerator has a 1024 deep coefficient memory to store the coefficients. The DMA controller loads the coefficients from system memory into coefficient memory. Four coefficients can be fetched from the coefficient memory simultaneously. If the soft filter length is more than 1024, processing happens in multi-iteration mode.

Prefetch Data Buffer

The prefetch data buffer enables pipeline operation. One data sample is prefetched when the compute unit is operating on the delay line corresponding to the current sample. The data prefetched in this buffer is later used to update the delay line for the next sample. This operation happens in parallel again when the compute unit is not accessing the delay line. In other words, it happens when the compute unit is adding the output from the four MACs and the partial sum register.

Table 51-5: Pipeline Operation for Window Size = 1

Cycles	1	2	3	4	5	6
Output DMA			N	N1	N2	N3
Compute		N	N1	N2	N3	
Input DMA	N	prefetch N1	prefetch N2	prefetch N3		

Processing Output

The accelerator uses all four MACs simultaneously to calculate one output sample as shown in the ***Multi-Iteration Filtering Flow*** figure and the following procedure.

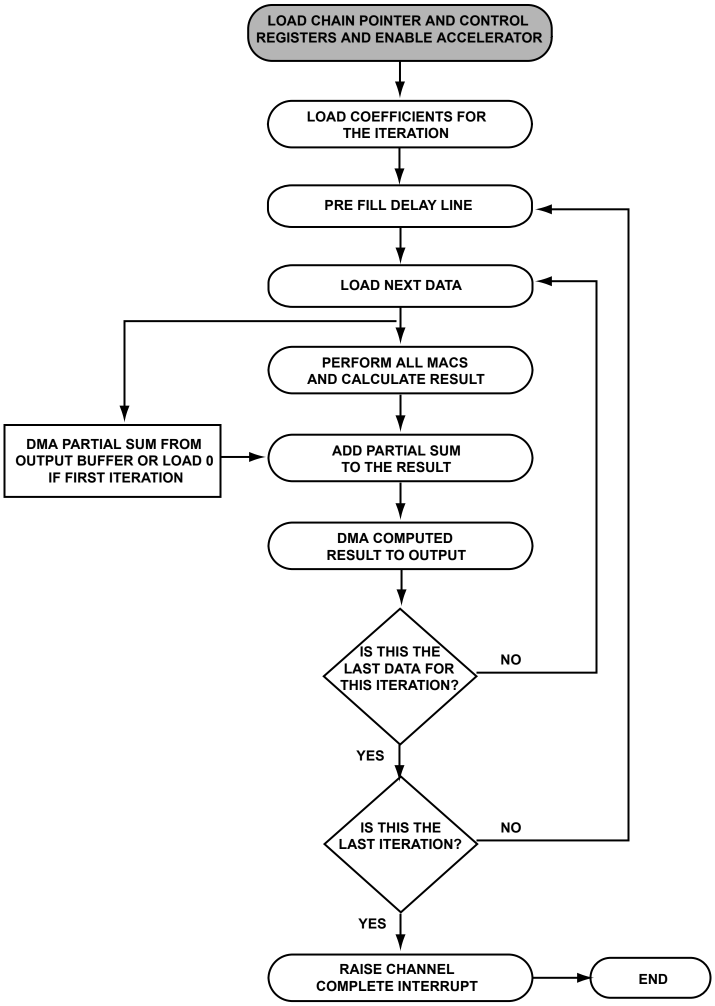

LOAD CHAIN POINTER AND CONTROL REGISTERS AND ENABLE ACCELERATOR
LOAD COEFFICIENTS FOR THE ITERATION
PRE FILL DELAY LINE
LOAD NEXT DATA
PERFORM ALL MACS AND CALCULATE RESULT
DMA PARTIAL SUM FROM OUTPUT BUFFER OR LOAD 0 IF FIRST ITERATION
ADD PARTIAL SUM TO THE RESULT
DMA COMPUTED RESULT TO OUTPUT
IS THIS THE LAST DATA FOR THIS ITERATION?
NO
YES
IS THIS THE LAST ITERATION?
NO
YES
RAISE CHANNEL COMPLETE INTERRUPT
END

Figure 51-3: Multi-Iteration Filtering Flow

1. The accelerator fetches four input data from the delay line and four corresponding coefficients from the coefficient memory and feeds them to the MAC units for multiply and accumulation operations.
2. The accelerator repeats the procedure with the next four input data and coefficients until all the TAPs complete. For example, this procedure happens N/4 times for an N TAP filter.
3. When all the TAPs are complete, the accelerator adds the four MAC outputs together to the previous partial sum (if any) to calculate the final result.
4. Finally, that output sample is stored back in system memory.

System Memory Storage

The following sections describe the storage format for the accelerator.

CAUTION: Store all data in memory aligned to the word address boundaries. Any other programmed addresses do not flag an error.

Coefficients and Input Buffer Storage

For any N TAP filter with coefficients:

```
C[i] i = 0,1,
...
N - 1
```

store the coefficients in system memory buffer in the order:

```
C[N - 1], C[N - 2]
...
C[1],C[0]
```

and CI should point to `C(N - 1)`.

Single Rate Input Filtering

The total size of the input buffer must at least be equal to `N - 1 + W`. If the input buffer that needs to be processed is:

```
 x[n],x[n+1],x[n+2]
 ...
 x[n+W-1]
```

store it in the memory as

```
x[n-(N-1)], x[n-(N-2)]
...
x[n-1], x[n], x[n+1]
...
x[n+W-1]
```

and the `FIR_INIDX` register value should point to `x[n - (N - 1)]`

Decimation

Assuming M = decimation ratio, the total size of the input buffer should at least be equal to `N - 1 + W x M`. If the input buffer that needs to be processed is:

```
x[n],x[n+1],x[n+2]....x[n+WxM-1],
```

store it in the memory as:

```
x[n-(N-1)], x[n-(N-2)]....x[n-1],
x[n], x[n+1]....x[n+WxM-1]
```

and the `FIR_INIDX` register value should point to `x[n-(N-1)]`.

Interpolation

Assuming L= interpolation ratio, the total size of the input buffer should be at least equal to Ceil `((N-1)/L)` `+W/L`.

If the input buffer that needs to be processed is:

```
x[n], x[n+1], x[n+2]....x[n+W/L-1],
and K= Ceil((N-1)/L)
```

store it in the memory as:

```
x[n-k], x[n-(K-1)], x[n-(K-2)]....x[n-1],
x[n], x[n+1].... x[n+W/L-1]
```

and the `FIR_INIDX` register value should point to `x[n-K]`.

Operating Modes

The FIR core performs a sum-of-products operation to compute the convolution sum. It supports single-rate, decimation, and interpolation functions.

Single Rate Processing

In a single-rate filter, the output result rate is equal to the input sample rate. The filter output Y(n) is computed according to following equation where N is the number of filter coefficients: c(k) k = 0 to N – 1 are the filter coefficients and x(n) represents the input time-series.

$$Y(n)= \sum_{k=0}^{N-1} c(k) \cdot x(n-k)$$

Figure 51-4: Filter Output Calculation

Single Iteration

Results are computed in a single iteration when the soft filter length is less than or equal to 1024.

Floating-point Multi-Iteration

Results are computed in multiple iterations when the soft filter length is greater than 1024 (for example, 2048 TAPs on a 1024 hard filter length). In this mode, the controller implements two iterations of 1024 TAPs.

NOTE: If the soft filter length is not a multiple of the hard filter length, the controller iterates until the soft filter length is satisfied.

Example: 550 taps on a 256 tap filter. In this example, the FIR controller implements two iterations of 256 taps and one iteration of 38 taps.

NOTE: Multi-iteration mode is not supported in fixed-point format.

Window Processing

Configure the window size to 1 to select sample-based processing mode. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window-based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

Multi-Rate Processing

Multi-rate filters change the sampling rate of a signal—they convert the input samples of a signal to a different set of data that represents the same signal sampled at a different rate.

Decimation

A decimation filter provides a single output result for every M input samples, where M is the decimation ratio. The output rate is 1/M'th of the input rate. The filter implementation exploits the low output sample rate by not starting a computation until a new set of M input samples is available.

In this mode, after low-pass filtering (for anti-aliasing), FIR logic discards the ratio – 1 samples of output data. For performance optimization, FIR logic skips the computation of output samples, which are discarded.

The input buffer size for decimation filters is N – 1 + (W × M) where:

- N is the number of taps
- W is the window size
- M is the decimation ratio

The window size (`FIR_CTL2.WINDOW` bits) must be programmed with the number of output samples.

To start this mode, programs set the `FIR_CTL2.RATIO` and `FIR_CTL2.UPSAMP` bits (along with normal filter setting). Also, the `FIR_CTL2.TAPLEN` bit field value should be greater than or equal to the `FIR_CTL2.RATIO` bit field value for the decimation filter.

Interpolation

An interpolation filter provides L output results for each new input sample, where L is the interpolation ratio. The output rate is L times the input rate.

In this mode, according to the ratio specified in configuration register, FIR logic inserts L – 1 zeros between any two input samples (upsampling). It then performs the interpolation (through the FIR filter).

Both upsampling and downsampling do not support multi-iteration mode. Therefore, the filtering operation only happens on up to 1024 TAPs and the ratio of up and downsampling can only be an integer value.

In an interpolation filter, FIR logic inserts L – 1 zeros between each sample. The program has to make sure that these zeros fully shift out of the delay line before moving on to the next channel. This operation puts a restriction on window size in terms of L – *the sample ratio* as showing in the expression:

WINDOWSIZE = *n* × *SAMPLERATIO*, where *n* is the number of input samples.

The input buffer size is smallest integer greater than or equal to (N – 1 + W)/L for interpolation filters where:

- N is the number of taps
- W is the window size
- L is the interpolation ratio

To start the mode, programs configure the `FIR_CTL2.RATIO` and `FIR_CTL2.UPSAMP` bits (along with filter settings).

Channel Processing

The ***Single Channel Filtering Flow*** figure shows the flow diagram for processing a single channel. Channels are processed in TDM format by setting the `FIR_CTL1.CH` bits to a value greater than one. In the time slot corresponding to a particular channel, the corresponding TCB is loaded from system memory.

1. The `FIR_CTL2` register value is fetched from system memory and is used to configure the filter parameters for that channel.
2. The accelerator fetches the coefficients using the `FIR_COEFIDX` register as the pointer and loads them into coefficient memory.
3. The delay line is pre-filled using the `FIR_INIDX` register as the pointer.
4. The accelerator calculates the first output and stores the result back into the output buffer using the `FIR_OUTIDX` register as the pointer.
5. While calculating the output, the accelerator fetches the next data in parallel. After one window of data is processed, the index registers in the system memory TCB are updated. In the next time slot of the same channel, processing can continue from where it stopped.
6. Processing moves to the next channel and repeats the procedure. If the soft filter length is more than the hard filter length, multiple iterations occur to process the window.

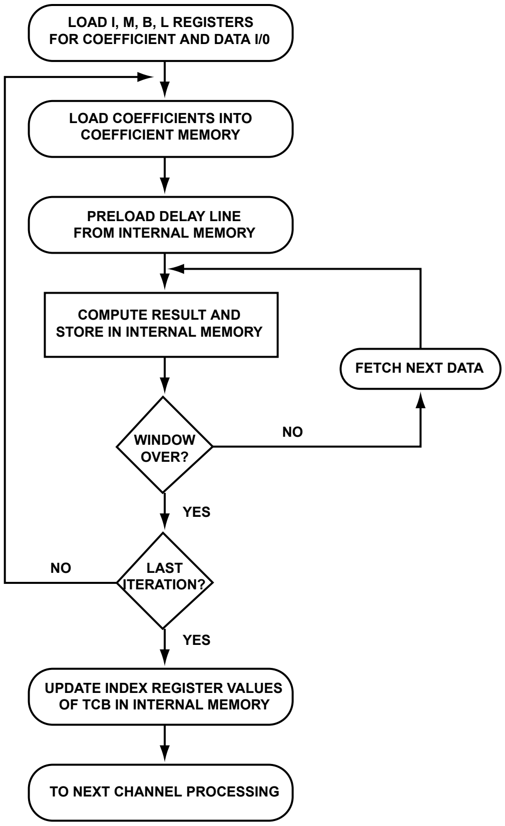

LOAD I, M, B, L REGISTERS FOR COEFFICIENT AND DATA I/0
LOAD COEFFICIENTS INTO COEFFICIENT MEMORY
PRELOAD DELAY LINE FROM INTERNAL MEMORY
COMPUTE RESULT AND STORE IN INTERNAL MEMORY
FETCH NEXT DATA
WINDOW OVER?
NO
YES
NO
LAST ITERATION?
YES
UPDATE INDEX REGISTER VALUES OF TCB IN INTERNAL MEMORY
TO NEXT CHANNEL PROCESSING

Figure 51-5: Single Channel Filtering Flow

Floating-Point Data Format

The FIR accelerator treats data and coefficients in 32-bit floating-point format as the default functional mode.

Fixed-Point Data Format

In fixed-point mode, the 32-bit input data or coefficient is treated as fixed point. A 32-bit fixed-point MAC operation generates an 80-bit result. Fixed-point data or coefficients can be unsigned integer, unsigned fractional and signed integer.

NOTE: In fixed-point mode, the entire 80-bit result register is always written back in bursts of 3 × 32 bits. The first word is the LSW, the second word is the MSW, and the third word is a 16-bit overflow. The remaining 16 bits are padded with zeros. Therefore, for fixed-point mode: WINDOWSIZE = WINDOWSIZE × 3.

If the signed fractional format is used, the output must be scaled by 2. The MAC does not right shift to remove the redundant sign bit. A final routine must decimate the output buffer to the desired samples.

Multi-iteration mode is not supported in this format. Therefore, the maximum TAP length is 1024.

Data Transfer

The FIR filter works exclusively through DMA.

Chain Assignment

The structure of a TCB is conceptually the same as a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB can point to itself to continuously re-run the same DMA. The FIR accelerator reads each word of the TCB and loads it into the corresponding register. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

The FIR accelerator DMA supports circular buffer chained DMA. The FIR accelerator does not support circular buffering for the coefficient buffer. The ***TCBs for Chained DMA*** table shows the required TCBs for chained DMA.

Table 51-6: TCBs for Chained DMA

Address	Register
TCB	`FIR_CHNPTR`
TCB + 0x1	`FIR_COEFCNT`
TCB + 0x2	`FIR_COEFMOD`
TCB + 0x3	`FIR_COEFIDX`
TCB + 0x4	`FIR_OUTBASE`
TCB + 0x5	`FIR_OUTCNT`
TCB + 0x6	`FIR_OUTMOD`
TCB + 0x7	`FIR_OUTIDX`
TCB + 0x8	`FIR_INBASE`
TCB + 0x9	`FIR_INCNT`
TCB + 0xA	`FIR_INMOD`
TCB + 0xB	`FIR_INIDX`
TCB + 0xC	`FIR_CTL2`

The `FIR_COEFCNT` register is loaded with the values in the `FIR_COEFCNT` TCB field and is decremented from that value onwards. However, coefficient loading continues until the number of coefficients, equal to the tap length, are read. This condition is true even if the `FIR_COEFCNT` register reaches zero as in the case of a tap length = 10, and the `FIR_COEFCNT` field in the TCB is initialized to 0. The value in the `FIR_COEFCNT` register is –10 after all coefficients are loaded.

NOTE: Initialize `FIR_CHNPTR` to TCB + 12.

DMA Access

The FIR accelerator has two DMA channels (accelerator input and output) to connect to the system memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Accelerator TCB

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the internal memory of the processor. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) must be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters (such as filter tap length, window size, sample rate conversion settings) for each channel
- DMA parameter register values for the input data (delay line)
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

Intermediate results in multi-iteration mode are saved in the output buffer.

As shown in the ***Circular Buffer Addressing*** figure, the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the appropriate values to the `FIR_INIDX` and `FIR_OUTIDX` bit fields of the TCB in memory. Then, data processing can begin from where it left off during the next time slot of that channel.

The write-back value for input buffer is:

- `FIR_INIDX` + W for single rate filtering
- `FIR_INIDX` + W × M for decimation (M = decimation ratio)
- `FIR_INIDX` + W/L for interpolation (L = interpolation ratio)
- The write-back value for output buffer in floating point mode is: `FIR_OUTIDX` + W
- The write-back value for output buffer in fixed-point mode is: `FIR_OUTIDX` + 3 × W

NOTE: The `FIR_CTL2` register is part of the FIR TCB. This configuration allows programming individual FIR channels with different control attributes.

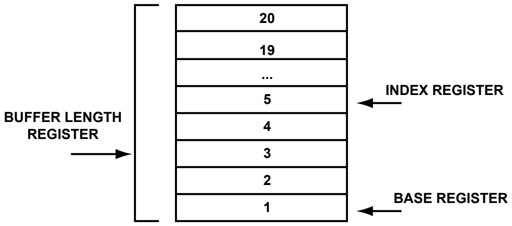


Figure 51-6: Circular Buffer Addressing

Programming Model

The following sections provide general programming information for the FIR accelerator.

Single Channel Processing

1. Create input, coefficient, and output buffers in system memory.

   For input and coefficient buffer storage format, see Coefficients and Input Buffer Storage.

2. Create the TCBs in system memory. Each TCB corresponds to a particular channel.

   TCBs hold the `FIR_CTL2` register which allows programming the window size and tap size along with up or down sample enable, sample rate conversion enable, and the conversion ratio for decimation and interpolation filters.

3. Configure the index, modifier and length entries in the TCBs to point to the corresponding channels' data buffer, coefficient buffer, and output data buffer.

   The output index register must point to the start of the output buffer. However, initialize the value of the input index register based on the explanation provided in Coefficients and Input Buffer Storage.

4. The core configures the `FIR_CTL1` register with the number of channels (one channel), fixed- or floating-point format.

5. Set the enable bit to start the accelerator operation in the modes configured (in `FIR_CTL1` and `FIR_CTL2` registers) by loading the TCB of the first channel. Once the first channel window is calculated, the input and output index registers are written back to internal memory corresponding to the first channel. Once the write-back is complete the accelerator moves into idle.

NOTE: All the addresses programmed in the TCB correspond to 32-bit address boundaries and should not contain the lower 2 bits (assumed as zeros).

Multichannel Processing

The *Wait for Core Intervention ≥ Idle (if auto channel iterate bit = 0)* figure shows the diagram for multichannel filtering. Multiple channels are processed in a time division multiplexed (TDM) format. After completing all the channels, the accelerator can either repeat the slots or wait for core intervention.

For multichannel filtering, use the following steps.

1. Program the number of channels using the `FIR_CTL1.CH` bits.
2. Configure the TCBs in system memory with one channel's TCB pointing to the next channel's TCB.
3. Write the first TCB value into the `FIR_CHNPTR` register and enable the accelerator.

   The accelerator fetches the first channel's TCB and, using it as pointer, prefills the delay line and coefficient memory and loads the `FIR_CTL2` register to configure the filter parameters corresponding to that channel.

   The accelerator then calculates output samples corresponding to one window and stores the data back in internal memory.

   At the end of the window the accelerator updates the `FIR_INIDX` and `FIR_OUTIDX` registers in the TCB of system memory and moves to the next channel.

   When all the channels are finished and the auto channel iterate bit (`FIR_CTL1.CAI`) =1, the accelerator processes the first channel again and iterates through the channels. If the `FIR_CTL1.CAI` bit =0, the accelerator waits for core intervention.

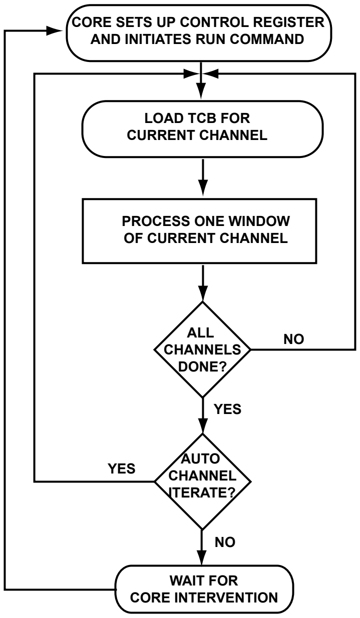

CORE SETS UP CONTROL REGISTER AND INITIATES RUN COMMAND
LOAD TCB FOR CURRENT CHANNEL
PROCESS ONE WINDOW OF CURRENT CHANNEL
ALL CHANNELS DONE?
NO
YES
AUTO CHANNEL ITERATE?
YES
NO
WAIT FOR CORE INTERVENTION

Figure 51-7: Wait for Core Intervention ≥ Idle (if auto channel iterate bit = 0)

Dynamic Coefficient Processing Notes

1. The dynamic update of the coefficients can be useful for the FIR accelerator. The FIR accelerator reloads the coefficients for each iteration (if the `FIR_CTL1.CAI` bit is set) before the start of processing of each channel.
2. The dynamic coefficient update is possible for single iteration mode (tap length ≤1024). Ensure that the new coefficients are updated after the accelerator loads the coefficients for current processing and before the next processing starts. The expression for the maximum time available for the coefficient memory update should be equal to 49 +N × 11 + W × (N/4 + 2) SCLK cycles.

3. For multi-iteration mode, dynamic updates are not supported. Programs must finish current processing, disable the accelerator, update the coefficients, and reenable the accelerator.

Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Clear the `FIR_CTL1.DMAEN` bit.
2. Set the `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.MEM`, and `FIR_DBG_CTL.HLD` bits.
3. Set the `FIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write the start address to the `FIR_DBG_ADDR` register.

   NOTE: If the bit 11 in the `FIR_DBG_ADDR` register is set, coefficient memory is selected.

5. Write data to the `FIR_DBG_WRDAT` register.

Read from Local Memory

1. Clear the `FIR_CTL1.DMAEN` bit.
2. Set the `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.MEM`, and `FIR_DBG_CTL.HLD` bits.
3. Set the `FIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write the start address to the `FIR_DBG_ADDR` register.

   NOTE: If bit 11 in the `FIR_DBG_ADDR` register is set, coefficient memory is selected.

5. Read data from the `FIR_DBG_RDDAT` register.

Single-Step Mode

Single-step mode can be used for debug purposes. An extra debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the `FIR_DBG_CTL.HLD` bit and enable `FIR_DBG_CTL.EN` and `FIR_DBG_CTL.RUN` bits.
3. Program the FIR module according to the application.
4. In single-step mode, each iteration is updated in the emulator session.

FIR Programming Example

In this example the application requires the FIR to filter six channels of data. The first four channels require a 256 TAP filter and the last two channels require a 1024 TAP filter. The window size for all the channels is 128.

1. Create a circular data buffer in system memory for each channel.

The buffer should be large enough to avoid overwriting data before the accelerator processes it. Ideally, the input buffer size for a channel is *tap length* + *window size* – 1 for that channel. The 256 coefficients of each of the first four channels and the 1024 coefficients each of the last two channels are also configured in system memory buffers. The output buffer size is equal to the window size.

2. Create six TCBs in system memory with each channel's chain pointer (CP) entry pointing to the next channel's and the sixth channel's CP entry pointing back to the first channel's in a circular manner.
3. Configure the `FIR_CTL2` register for the first four channels' TCBs to 256 TAPs and a window size of 128, and the next two channels for 1024 TAPs and a window size of 128, respectively.
4. Configure the index, modifier, length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer, and output data buffer. The location of the first channel's TCB is written to the `FIR_CHNPTR` register. The `FIR_CTL1.CH` bit field is then programmed with a value that corresponds to six channels.
    a. The accelerator iterates through six channels once and then waits for core intervention (the `FIR_CTL1.CAI` bit is not set, the DMA is enabled, and the `FIR_CTL1.EN` bit is set).
    b. The accelerator loads the TCB of the first channel, then loads the coefficient and data, and processes one window.
    c. After saving the index values to memory, the accelerator moves to the next channel.
    d. After all six channels are complete, the accelerator halts and waits for core intervention.

Computing FIR Output, Tap Length Greater than 4096

With little core intervention, the FIR accelerator can also be used to calculate output for a tap length greater than 4096 taps. The section demonstrates the calculation with an example of 8192 taps.

1. Divide the transfer function of an 8192 FIR filter into two 4096 FIR filters:

$$H(Z) = b_0 + b_1\mathrm{Z}^{-1} + b_2Z^{-2} + \ldots\ldots\ldots\ldots b_{4095}Z^{-4095} + b_{4096}Z^{-4096}\mathrm{b}_{4097}Z^{-4097} + \ldots\ldots\ldots b_{8191}Z^{-8191}$$

$$= b_0 + b_1\mathrm{Z}^{-1} + b_2Z^{-2} + \ldots\ldots\ldots\ldots b_{4095}Z^{-4095} + Z^{-4096}(b_{4096} + b_{4097} + \ldots\ldots\ldots b_{8191}Z^{-4096})$$

2. Divide the filter coefficients of an 8192 tap filter among two 4096 tap FIR filters:

    Filter 1

    Coefficients = b0, b1, b2,……, b4095

    Input data = x[n], x [n – 1],……….x[n – 4095]

    Filter 2

    Coefficients = b4096, b4097,…………..,b8191

    Input data = x[n – 4096], x[n – 4097],……x[n – 8191]

The accelerator can be used in two-channel mode where channel 1 operates on x[n]...x[n – 4095] input data with the filter coefficients of filter 1 and channel 2 operates on x[n – 4096]...x[n – 8191] with the filter coefficients of filter 2.

Once both the channels are processed, add the partial sum output of both the channels can to get the final output. Implement this approach (tap length = TAPS = 8192, window size = WINDOW) using the following programming steps.

1. Create a circular input data buffer in system memory (IBUF). The buffer must be large enough to avoid overwriting data before the accelerator processes it. Ideally, the input buffer size for a channel is TAPS + WINDOW – 1.
2. Create a coefficient buffer of size TAPS (8192) (CBUF).
3. Create one output buffer of size WINDOW (OBUF) and another temporary output buffer (OBUF1) to store the partial sum.
4. Create two TCBs in system memory with first TCB chained to the second and second one chained to the first in circular manner.
    a. The `FIR_COEFIDX` bit field of the first TCB should point to the start address of the coefficient buffer (CBUF) and that of the second TCB should point to 4096 offset from the start of the coefficient buffer (CBUF + 4096).
    b. The `FIR_OUTBASE` and `FIR_OUTIDX` bit field of the first TCB should point to the start address of OBUF and that of the second TCB should point to the start address of OBUF1.
    c. The `FIR_INIDX` bit field of the first TCB should point to the start address of IBUF and that of the second TCB should point to 4096 offset from the start address of IBUF.
    d. Configure the `FIR_CTL2` bit field of both the TCB for tap length = TAP/2 = 4096 and window size = WINDOW.
5. Initialize the `FIR_CHNPTR` register pointing to the first TCB.
6. Program the `FIR_CTL1` register to initiate the accelerator processing now by setting the `FIR_CTL1.EN` and `FIR_CTL1.DMAEN` bits and the number of channels configured as 2.
7. Wait for the FIR all channel done interrupt (`FIR_DMASTAT.ACDONE`) to occur. Inside the ISR, add the partial sum results using the core from both the output buffers (OBUF and OBUF1) to get the final output. To save memory, replace the contents of the buffer OBUF with the final output result.

Debug Features

The following sections provide information for debugging the FIR accelerator.

Local Memory Access

The contents of FIR delay line and coefficient memories are made observable for debug by setting the `FIR_DBG_CTL.EN`/`FIR_DBG_CTL.MEM` and `FIR_DBG_CTL.HLD` bits. The debug address register (`FIR_DBG_ADDR`) and two data registers are provided for debug operations. Bit 11 of the `FIR_DBG_ADDR` register selects coefficient memory when set (=1) and selects delay line memory when cleared (=0).

In the debug mode, the read data register (`FIR_DBG_RDDAT`) returns the contents of the memory location pointed to by the address register. Data can be written into any memory location using `FIR_DBG_WRDAT` register writes. If the address auto-increment bit (`FIR_DBG_CTL.ADRINC`) is set, the address register auto-increments on `FIR_DBG_WRDAT` writes and `FIR_DBG_RDDAT` reads. During auto-increment the `FIR_DBG_ADDR` register cannot cross the data memory or coefficient memory boundary.

Single-Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The `FIR_DBG_CTL.EN`, `FIR_DBG_CTL.HLD`, and `FIR_DBG_CTL.MEM` bits control the FIR MAC units.

Emulation Considerations

In FIR debug mode, the DMA operations are not observable.

Interrupts

The ***FIR Interrupt Overview*** table provides the source of interrupt and service instructions for the FIR interrupts.

Table 51-7: FIR Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
FIR_DMA FIR_STAT	Input DMA complete Output DMA complete Window complete All channels complete	N/A	ROC from `FIR_DMASTAT` + RTI instruction
	MAC IEEE floating-point exceptions MAC fixed-point overflow		ROC from `FIR_MACSTAT` + RTI instruction

Sources

The FIR module drives two interrupt signals: `FIR_DMA` for the DMA status and `FIR_STAT` for the MAC status. The FIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. Note that the interrupt follows the access completion rule, where the interrupt is generated when all data are written back to system memory.

MAC Status

A MAC status interrupt is generated under the following conditions and is reflected in the `FIR_MACSTAT` register.

- Multiplier result zero – Set if multiplier result is zero
- Multiplier result infinity – Set if multiplier result is infinity
- Multiply invalid – Set if multiply operation is invalid
- Adder result zero – Set if adder result is zero
- Adder result infinity – Set if adder result is infinity
- Adder invalid – Set if addition is invalid
- Adder overflow – for fixed-point operation

Service

When a DMA interrupt occurs, programs can find whether the input or output DMA interrupt occurred by reading the DMA status register (`FIR_DMASTAT`). The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this state by reading the MAC status register (`FIR_MACSTAT`). The read of the register clears the (sticky) MAC interrupt status bits.

The status interrupt sources are derived from the `FIR_MACSTAT` register. A status interrupt can occur due to the last set of MAC operations of a processing iteration that correspond to a particular channel. The interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped when another processing iteration results in a non-zero or valid multiply or add result. However, in this situation, it is difficult to isolate whether the interrupt corresponds to the previous processing iteration or the current one. This functionality makes using status interrupts impractical.

Another option is to poll the status bits of the `FIR_MACSTAT` register inside the DMA interrupt service routine. However, consider the behavior of the status bits. The status bits in the `FIR_MACSTAT` registers are sticky. Once a status bit is set, it gets cleared only when the `FIR_MACSTAT` register is read and the previous set of MAC operations resulted in a non-zero, valid output. Therefore, if the last set of MAC operations of a processing iteration results in a zero, non-valid output, the corresponding status bit are not cleared, even after reading the `FIR_MACSTAT` register. To avoid a false indication in the next processing iteration, ensure that all the status bits are cleared after the current iteration finishes.

The solution is to read the `FIR_MACSTAT` register twice inside the DMA interrupt service routine. The first read identifies which status bits are set. The second read is used to discover if the status bit was set because of the last set of MAC operations. If the status bit was not set because of the last set of MAC operations, it provides a zero result.

If the bit was set because of the last set of MAC operations, clear the status bit by performing a simple dummy FIR processing iteration (tap length = 4 and window size = 1). Choose the appropriate `FIR_MACSTAT` register after the processing is complete.

For more information, see the "FIR MAC Status Register" section.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral-specific).

Write Effect Latency

For details on write effect latency, see the *SHARC Processor Programming Reference*.

FIR Throughput

Accelerator input and output channels are used to connect to internal memory. Data throughput is one 32-bit data word per SCLK cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two SCLK clock cycles.

The following information describes the performance of the FIR accelerator in SCLK cycles. The total number of SCLK cycles for a single channel single rate filtering N <= 1024 is the sum of the cycles taken for the following operations.

- TCB load (SCLK cycles) = $13 \times 11 + 3$
- Coefficient Load (SCLK cycles)= $N \times 11 + 3$
- Data Load (SCLK cycles)= $N \times 11 + 3$
- Calculation (SCLK cycles)= $(N/4 + 2) \times W + 16$
- Write Back (SCLK Cycles) = 27

Total number of SCLK cycles for a single rate filtering $N <= 1024$ is:

$(195 + 2 \times N \times 11 + W \times (N/4+2)) \times C$ SCLK cycles

where:

- N is the Number of taps
- W is the Window size
- C is the Number of channels

ADSP-SC58x FIR Register Descriptions

The FIR accelerator (FIR) contains the following registers.

Table 51-8: ADSP-SC58x FIR Register List

Name	Description
FIR_CHNPTR	FIR Chain Pointer Register
FIR_COEFCNT	FIR Coefficient Count Register
FIR_COEFIDX	FIR Coefficient Index Register
FIR_COEFMOD	FIR Coefficient Modifier Register
FIR_CTL1	FIR Global Control Register
FIR_CTL2	FIR Channel Control Register
FIR_DBG_ADDR	Debug Address Register
FIR_DBG_CTL	FIR Debug Control Register
FIR_DBG_RDDAT	FIR Debug Data Read Register
FIR_DBG_WRDAT	FIR Debug Data Write Register
FIR_DMASTAT	FIR DMA Status Register
FIR_INBASE	FIR Input Data Base Register
FIR_INCNT	FIR Input Data Count Register
FIR_INIDX	FIR Input Data Index Register
FIR_INMOD	FIR Input Data Modifier Register
FIR_MACSTAT	FIR MAC Status Register
FIR_OUTBASE	FIR Output Data Base Register
FIR_OUTCNT	FIR Output Data Count Register
FIR_OUTIDX	FIR Output Data Index Register
FIR_OUTMOD	FIR Output Data Modifier Register

FIR Chain Pointer Register

The `FIR_CHNPTR` register contains the chain pointer address.

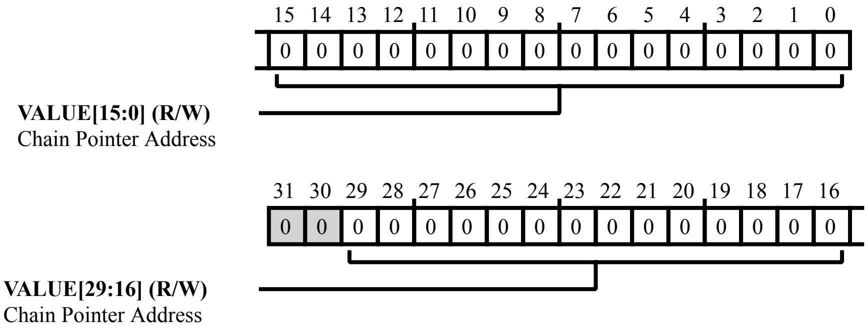

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Chain Pointer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Chain Pointer Address

Figure 51-8: FIR_CHNPTR Register Diagram

Table 51-9: FIR_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Chain Pointer Address. The `FIR_CHNPTR.VALUE` bit field contains the chain pointer address.

FIR Coefficient Count Register

The `FIR_COEFCNT` register contains the 16-bit coefficient buffer count.

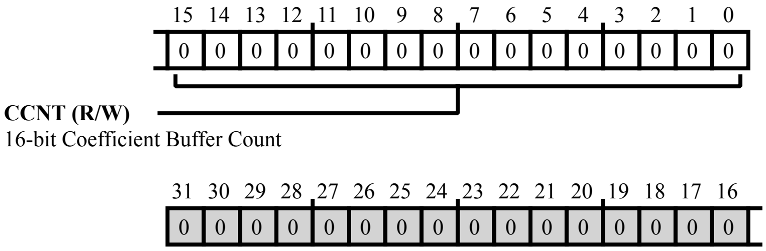

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CCNT (R/W)
16-bit Coefficient Buffer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-9: FIR_COEFCNT Register Diagram

Table 51-10: FIR_COEFCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CCNT	16-bit Coefficient Buffer Count. The `FIR_COEFCNT.CCNT` bit field contains the 16-bit coefficient buffer count.

FIR Coefficient Index Register

The `FIR_COEFIDX` register contains the coefficient index word address with the lower two bits removed.

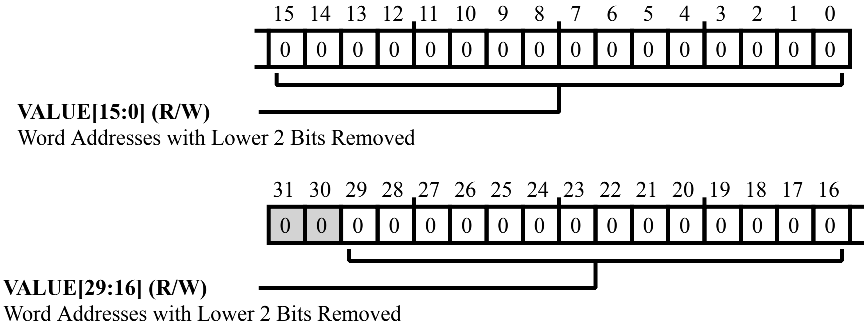

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Word Addresses with Lower 2 Bits Removed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Word Addresses with Lower 2 Bits Removed

Figure 51-10: FIR_COEFIDX Register Diagram

Table 51-11: FIR_COEFIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Addresses with Lower 2 Bits Removed. The `FIR_COEFIDX.VALUE` bit field contains the word addresses with the lower 2 bits removed.

FIR Coefficient Modifier Register

The FIR_COEFMOD register contains the 16-bit coefficient index modifier.

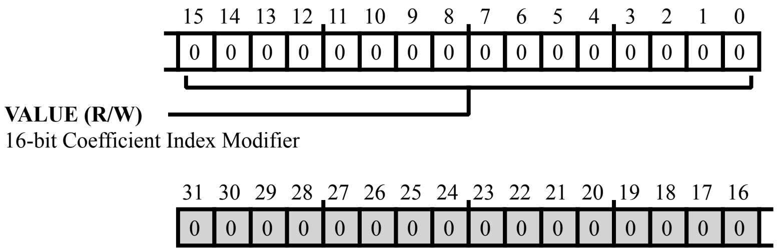

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Coefficient Index Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-11: FIR_COEFMOD Register Diagram

Table 51-12: FIR_COEFMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Coefficient Index Modifier. The FIR_COEFMOD.VALUE bit field contains the 16-bit coefficient index modifier.

FIR Global Control Register

The `FIR_CTL1` register is used to configure the global parameters for the accelerator. These parameters include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

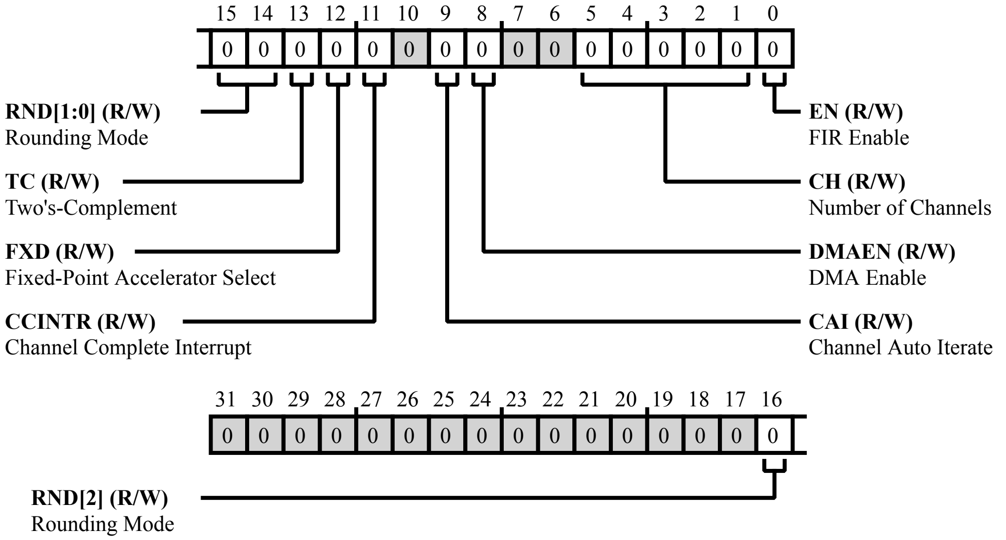

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RND[1:0] (R/W)
Rounding Mode
TC (R/W)
Two's-Complement
FXD (R/W)
Fixed-Point Accelerator Select
CCINTR (R/W)
Channel Complete Interrupt
EN (R/W)
FIR Enable
CH (R/W)
Number of Channels
DMAEN (R/W)
DMA Enable
CAI (R/W)
Channel Auto Iterate
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RND[2] (R/W)
Rounding Mode

Figure 51-12: FIR_CTL1 Register Diagram

Table 51-13: FIR_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	RND	Rounding Mode. The `FIR_CTL1.RND` bit configures the accelerator to use one of the following rounding modes.	
		0	Round to Nearest
		1	Truncate (Round away from zero)
13 (R/W)	TC	Two's-Complement. The `FIR_CTL1.TC` bit configures the accelerator to use either unsigned integer or signed integer	
		0	Unsigned integer
		1	Signed integer
12 (R/W)	FXD	Fixed-Point Accelerator Select. The `FIR_CTL1.FXD` bit configures the accelerator to use either 32-bit IEEE floating-point or 32-bit fixed-point.	
		0	32-bit IEEE floating-point
		1	32-bit fixed point

Table 51-13: FIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	CCINTR	Channel Complete Interrupt. The `FIR_CTL1.CCINTR` bit configures the accelerator to generate an interrupt when each or all channels are done.	
		0	Interrupt is generated only when all channels are done
		1	Interrupt is generated after each channel is done
9 (R/W)	CAI	Channel Auto Iterate. The `FIR_CTL1.CAI` bit, if cleared, causes the TDM processing to stop (idle) once all channels are done. If set, processing moves to the first channel and continues TDM processing in a loop when all channels are done.	
		0	TDM processing stops (idle) once all channels are done
		1	Moves to first channel and continues TDM processing in a loop when all channels are done
8 (R/W)	DMAEN	DMA Enable. The `FIR_CTL1.DMAEN` bit enables and disables DMA on the FIR accelerator.	
		0	DMA disabled
		1	DMA enabled
5:1 (R/W)	CH	Number of Channels. The `FIR_CTL1.CH` bit field configures the number of channels and is programmable from 0 to 31.	
		0	Channel 1
		1-30	Channel 2-31
		31	Channel 32
0 (R/W)	EN	FIR Enable. The `FIR_CTL1.EN` bit enables and disables the FIR accelerator.	
		0	Disable FIR
		1	Enable FIR

FIR Channel Control Register

The `FIR_CTL2` register is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio.

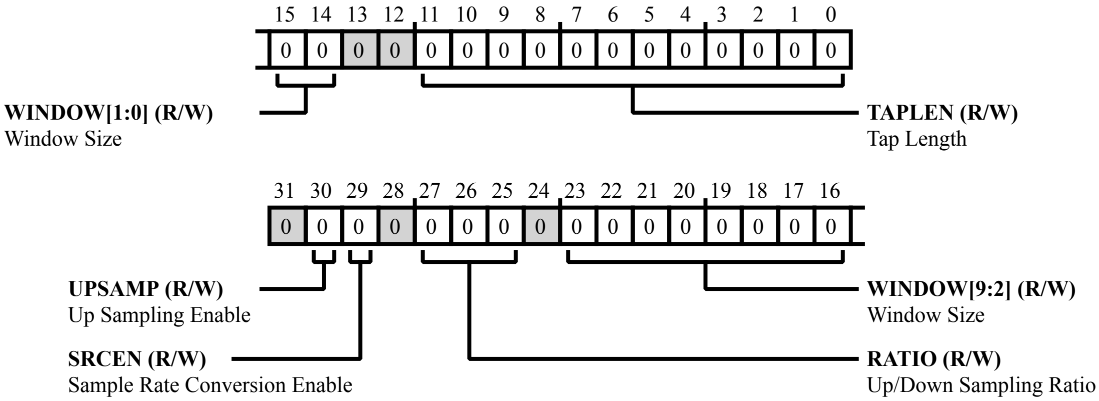

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WINDOW[1:0] (R/W)
Window Size
TAPLEN (R/W)
Tap Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
UPSAMP (R/W)
Up Sampling Enable
SRCEN (R/W)
Sample Rate Conversion Enable
WINDOW[9:2] (R/W)
Window Size
RATIO (R/W)
Up/Down Sampling Ratio

Figure 51-13: FIR_CTL2 Register Diagram

Table 51-14: FIR_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	UPSAMP	Up Sampling Enable. The `FIR_CTL2.UPSAMP` bit enables up sampling.	
		0	Down sampling
		1	Up sampling
29 (R/W)	SRCEN	Sample Rate Conversion Enable. The `FIR_CTL2.SRCEN` bit field enables sample rate conversion.	
		0	Disabled
		1	Enabled
27:25 (R/W)	RATIO	Up/Down Sampling Ratio. The `FIR_CTL2.RATIO` bit field sets the sampling ratio (`FIR_CTL2.RATIO` + 1).	
23:14 (R/W)	WINDOW	Window Size. The `FIR_CTL2.WINDOW` bit field sets the window size which specifies the number of sample/block to process (sample based processing = window size of 0).	
11:0 (R/W)	TAPLEN	Tap Length. The `FIR_CTL2.TAPLEN` bit field sets the tap length which is programmable between 0-4095 (Tap Length = `FIR_CTL2.TAPLEN` + 1).	

Debug Address Register

The FIR_DBG_ADDR register holds the debug address.

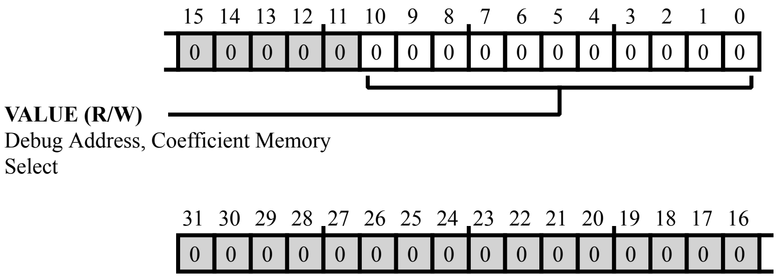

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Debug Address, Coefficient Memory Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-14: FIR_DBG_ADDR Register Diagram

Table 51-15: FIR_DBG_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Debug Address, Coefficient Memory Select. The `FIR_DBG_ADDR.VALUE` bit field holds the debug address (bits 0-10). Bit 11 configures whether the memory access is to coefficient memory (=0) or to delay line memory (=1).

FIR Debug Control Register

The `FIR_DBG_CTL` register controls debugging operations such as enabling debug mode running, hold or single stepping.

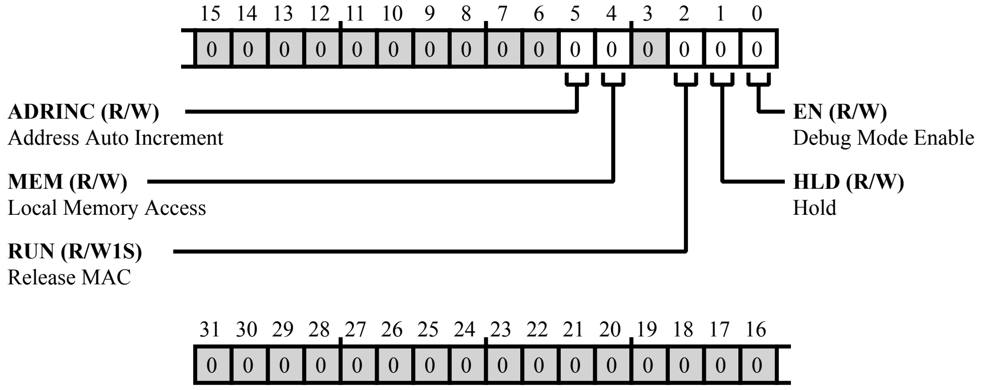

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADRINC (R/W)
Address Auto Increment
MEM (R/W)
Local Memory Access
RUN (R/W1S)
Release MAC
EN (R/W)
Debug Mode Enable
HLD (R/W)
Hold
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-15: FIR_DBG_CTL Register Diagram

Table 51-16: FIR_DBG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	ADRINC	Address Auto Increment. The `FIR_DBG_CTL.ADRINC` bit bit allows the address register to auto-increment on `FIR_DBG_WRDAT` writes and `FIR_DBG_RDDAT` reads.
4 (R/W)	MEM	Local Memory Access. When the `FIR_DBG_CTL.MEM` bit is set, the data and coefficients memory can be indirectly accessed.
2 (R/W1S)	RUN	Release MAC. The `FIR_DBG_CTL.RUN` bit releases the MAC. This bit is self-clearing after one FIR clock cycle.
1 (R/W)	HLD	Hold. The `FIR_DBG_CTL.HLD` bit holds or single-steps through the FIR. 0 Hold 1 Single-step
0 (R/W)	EN	Debug Mode Enable. The `FIR_DBG_CTL.EN` bit enables debug mode. For local memory access, the `FIR_CTL1` register can be cleared. 0 Disable debug mode 1 Enable debug mode

FIR Debug Data Read Register

The `FIR_DBG_RDDAT` register hold the debug read data.

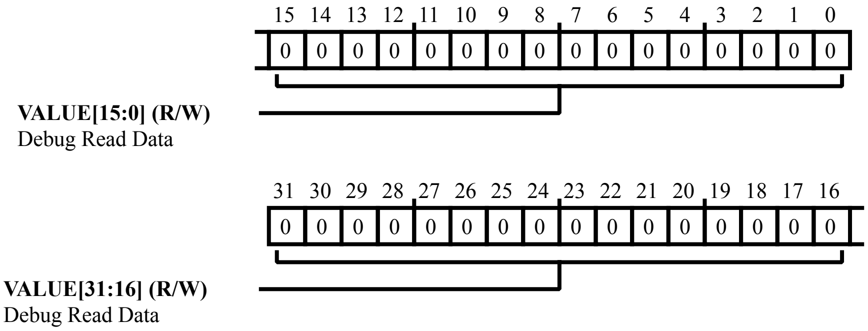

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Debug Read Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Debug Read Data

Figure 51-16: FIR_DBG_RDDAT Register Diagram

Table 51-17: FIR_DBG_RDDAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Read Data. The `FIR_DBG_RDDAT.VALUE` bit field holds the debug read data.

FIR Debug Data Write Register

The `FIR_DBG_WRDAT` register holds the debug write data.

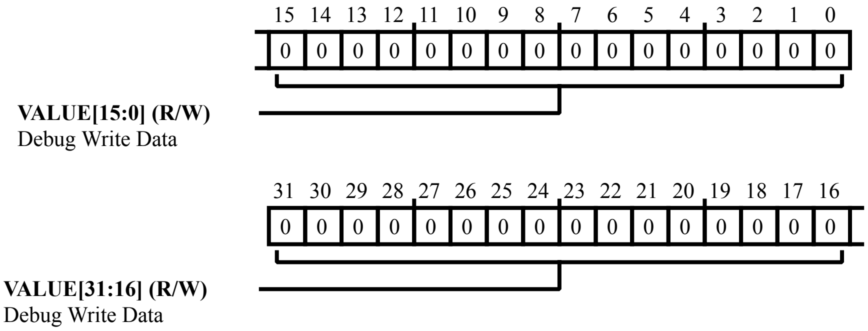

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Debug Write Data
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Debug Write Data

Figure 51-17: FIR_DBG_WRDAT Register Diagram

Table 51-18: FIR_DBG_WRDAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Write Data. The `FIR_DBG_WRDAT.VALUE` bit field holds the debug write data.

FIR DMA Status Register

The `FIR_DMASTAT` register provides information about chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window complete, all channels complete.

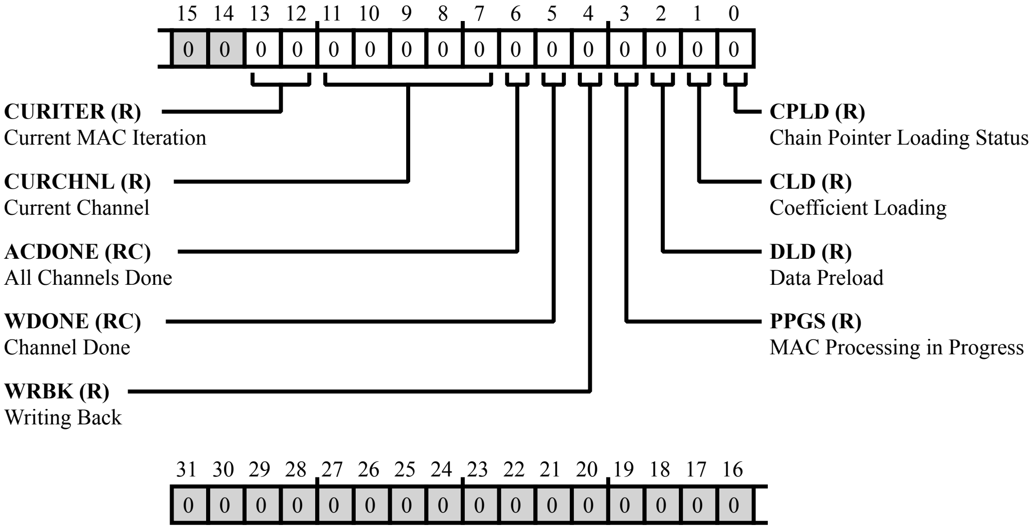

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CURITER (R)
Current MAC Iteration
CURCHNL (R)
Current Channel
ACDONE (RC)
All Channels Done
WDONE (RC)
Channel Done
WRBK (R)
Writing Back
CPLD (R)
Chain Pointer Loading Status
CLD (R)
Coefficient Loading
DLD (R)
Data Preload
PPGS (R)
MAC Processing in Progress
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-18: FIR_DMASTAT Register Diagram

Table 51-19: FIR_DMASTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:12 (R/NW)	CURITER	Current MAC Iteration. The `FIR_DMASTAT.CURITER` bit indicates the current MAC iteration in multi-iteration mode. Zero indicates the final iteration.
11:7 (R/NW)	CURCHNL	Current Channel. The `FIR_DMASTAT.CURCHNL` bit indicates the channel that is being processed in the TDM slot. Zero indicates the last slot.
6 (RC/NW)	ACDONE	All Channels Done. The `FIR_DMASTAT.ACDONE` bit indicates the accelerator that processing all channels is complete. This is a sticky bit and is cleared on a register read. The `FIR_CTL1.CCINTR` bit does not affect the `FIR_DMASTAT.ACDONE` bit.
5 (RC/NW)	WDONE	Channel Done. The `FIR_DMASTAT.WDONE` bit indicates the accelerator that processing the current channel is complete. This is a sticky bit and is cleared on a register read. The `FIR_CTL1.CCINTR` bit does not affect the `FIR_DMASTAT.WDONE` bit.
4 (R/NW)	WRBK	Writing Back. The `FIR_DMASTAT.WRBK` bit indicates the accelerator is writing back the updated index registers.

Table 51-19: FIR_DMASTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	PPGS	MAC Processing in Progress. The `FIR_DMASTAT.PPGS` bit indicates MAC processing in progress.
2 (R/NW)	DLD	Data Preload. The `FIR_DMASTAT.DLD` bit indicates data preloading.
1 (R/NW)	CLD	Coefficient Loading. The `FIR_DMASTAT.CLD` bit indicates coefficient loading.
0 (R/NW)	CPLD	Chain Pointer Loading Status. The `FIR_DMASTAT.CPLD` bit indicates the state machine is in chain pointer load state.

FIR Input Data Base Register

The `FIR_INBASE` register contains the input word base address with the lower two bits removed.

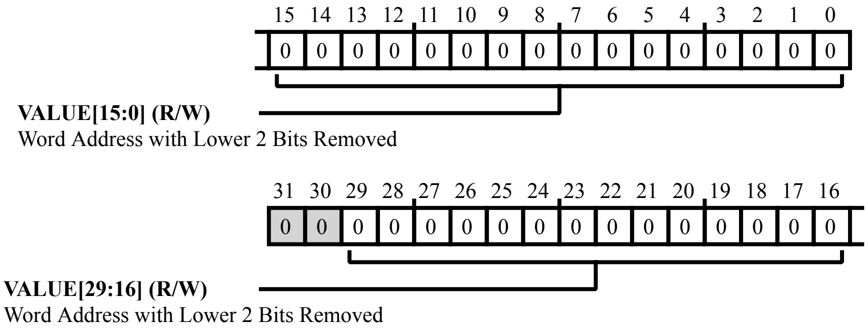

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Word Address with Lower 2 Bits Removed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Word Address with Lower 2 Bits Removed

Figure 51-19: FIR_INBASE Register Diagram

Table 51-20: FIR_INBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The `FIR_INBASE.VALUE` bit field contains the the word address with the lower 2 bits removed.

FIR Input Data Count Register

The `FIR_INCNT` register contains the 16-bit input data count.

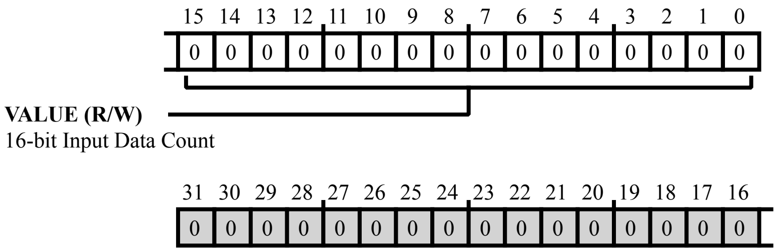

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Input Data Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-20: FIR_INCNT Register Diagram

Table 51-21: FIR_INCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Count. The `FIR_INCNT.VALUE` bit field contains the 16-bit input data count.

FIR Input Data Index Register

The `FIR_INIDX` register contains the input word address with the lower two bits removed.

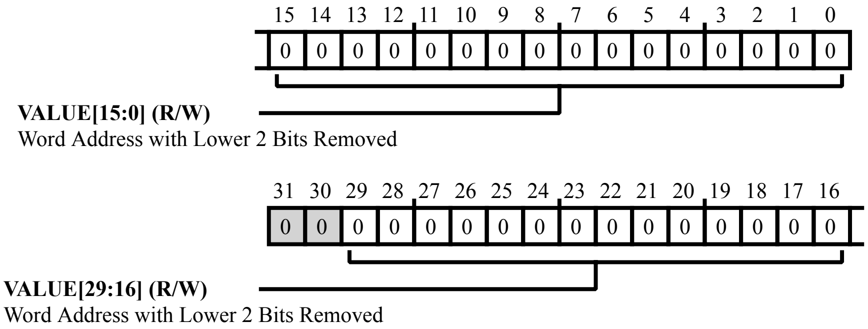

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Word Address with Lower 2 Bits Removed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Word Address with Lower 2 Bits Removed

Figure 51-21: FIR_INIDX Register Diagram

Table 51-22: FIR_INIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The `FIR_INIDX.VALUE` bit field contains the input word address with the lower two bits removed.

FIR Input Data Modifier Register

The `FIR_INMOD` register contains the 16-bit input data buffer modifier.

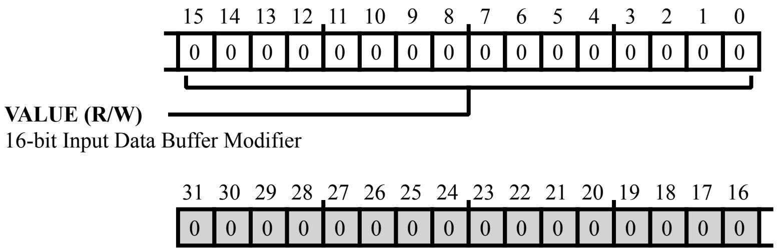

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Input Data Buffer Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-22: FIR_INMOD Register Diagram

Table 51-23: FIR_INMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Modifier. The `FIR_INMOD.VALUE` bit field contains the 16-bit input data buffer modifier.

FIR MAC Status Register

The `FIR_MACSTAT` register provides the status of MAC operations. The status of all four multipliers/adders are available separately for programs to poll. In fixed-point mode only, the ARIx bits are used (all other bits are reserved).

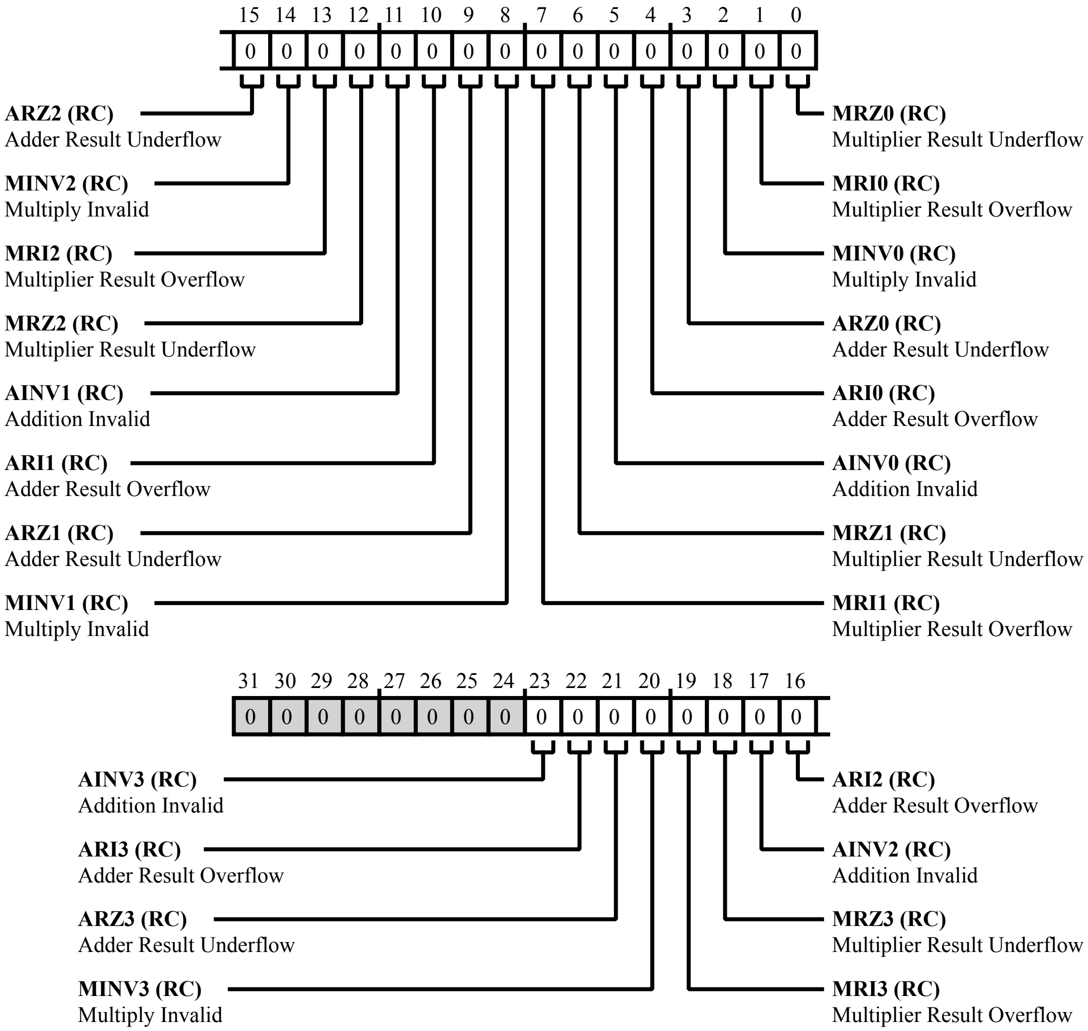

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARZ2 (RC)
Adder Result Underflow
MINV2 (RC)
Multiply Invalid
MRI2 (RC)
Multiplier Result Overflow
MRZ2 (RC)
Multiplier Result Underflow
AINV1 (RC)
Addition Invalid
ARI1 (RC)
Adder Result Overflow
ARZ1 (RC)
Adder Result Underflow
MINV1 (RC)
Multiply Invalid
MRZ0 (RC)
Multiplier Result Underflow
MRI0 (RC)
Multiplier Result Overflow
MINV0 (RC)
Multiply Invalid
ARZ0 (RC)
Adder Result Underflow
ARI0 (RC)
Adder Result Overflow
AINV0 (RC)
Addition Invalid
MRZ1 (RC)
Multiplier Result Underflow
MRI1 (RC)
Multiplier Result Overflow
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AINV3 (RC)
Addition Invalid
ARI3 (RC)
Adder Result Overflow
ARZ3 (RC)
Adder Result Underflow
MINV3 (RC)
Multiply Invalid
ARI2 (RC)
Adder Result Overflow
AINV2 (RC)
Addition Invalid
MRZ3 (RC)
Multiplier Result Underflow
MRI3 (RC)
Multiplier Result Overflow

Figure 51-23: FIR_MACSTAT Register Diagram

Table 51-24: FIR_MACSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (RC/NW)	AINV3	Addition Invalid. The `FIR_MACSTAT.AINV3` bit is set if the adder 3 addition is invalid.

Table 51-24: FIR_MACSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (RC/NW)	ARI3	Adder Result Overflow. The `FIR_MACSTAT.ARI3` bit is set if the adder 3 result is infinity. Indicates overflow in fixed-point mode.
21 (RC/NW)	ARZ3	Adder Result Underflow. The `FIR_MACSTAT.ARZ3` bit is set if the adder 3 result is zero.
20 (RC/NW)	MINV3	Multiply Invalid. The `FIR_MACSTAT.MINV3` bit is set if the multiplier 3 multiply operation is invalid.
19 (RC/NW)	MRI3	Multiplier Result Overflow. The `FIR_MACSTAT.MRI3` bit is set if the multiplier 3 result is infinity.
18 (RC/NW)	MRZ3	Multiplier Result Underflow. The `FIR_MACSTAT.MRZ3` bit is set if the multiplier 3 result is zero.
17 (RC/NW)	AINV2	Addition Invalid. The `FIR_MACSTAT.AINV2` bit is set if the adder 2 addition is invalid.
16 (RC/NW)	ARI2	Adder Result Overflow. The `FIR_MACSTAT.ARI2` bit is set if the adder 2 result is infinity. Indicates overflow in fixed-point mode.
15 (RC/NW)	ARZ2	Adder Result Underflow. The `FIR_MACSTAT.ARZ2` bit is set if the adder 2 result is zero.
14 (RC/NW)	MINV2	Multiply Invalid. The `FIR_MACSTAT.MINV2` bit is set if the multiplier 2 multiply operation is invalid.
13 (RC/NW)	MRI2	Multiplier Result Overflow. The `FIR_MACSTAT.MRI2` bit is set if the multiplier 2 result is infinity.
12 (RC/NW)	MRZ2	Multiplier Result Underflow. The `FIR_MACSTAT.MRZ2` bit is set if the multiplier 2 result is zero.
11 (RC/NW)	AINV1	Addition Invalid. The `FIR_MACSTAT.AINV1` bit is set if the adder 1 addition is invalid.
10 (RC/NW)	ARI1	Adder Result Overflow. The `FIR_MACSTAT.ARI1` bit is set if the adder 1 result is infinity. Indicates overflow in fixed-point mode.
9 (RC/NW)	ARZ1	Adder Result Underflow. The `FIR_MACSTAT.ARZ1` bit is set if the adder 1 result is zero.

Table 51-24: FIR_MACSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (RC/NW)	MINV1	Multiply Invalid. The `FIR_MACSTAT.MINV1` bit is set if the multiplier 1 multiply operation is invalid.
7 (RC/NW)	MRI1	Multiplier Result Overflow. The `FIR_MACSTAT.MRI1` bit is set if the multiplier 1 result is infinity.
6 (RC/NW)	MRZ1	Multiplier Result Underflow. The `FIR_MACSTAT.MRZ1` bit is set if the multiplier 1 result is zero.
5 (RC/NW)	AINV0	Addition Invalid. The `FIR_MACSTAT.AINV0` bit is set if the adder 0 addition is invalid.
4 (RC/NW)	ARI0	Adder Result Overflow. The `FIR_MACSTAT.ARI0` bit is set if the adder 0 result is infinity. Indicates overflow in fixed-point mode.
3 (RC/NW)	ARZ0	Adder Result Underflow. The `FIR_MACSTAT.ARZ0` bit is set if the adder 0 result is zero.
2 (RC/NW)	MINV0	Multiply Invalid. The `FIR_MACSTAT.MINV0` bit is set if the multiplier 0 multiply operation is invalid.
1 (RC/NW)	MRI0	Multiplier Result Overflow. The `FIR_MACSTAT.MRI0` bit is set if the multiplier 0 result is infinity.
0 (RC/NW)	MRZ0	Multiplier Result Underflow. The `FIR_MACSTAT.MRZ0` bit is set if multiplier 0 result is zero.

FIR Output Data Base Register

The `FIR_OUTBASE` register contains the output word base address with the lower two bits removed

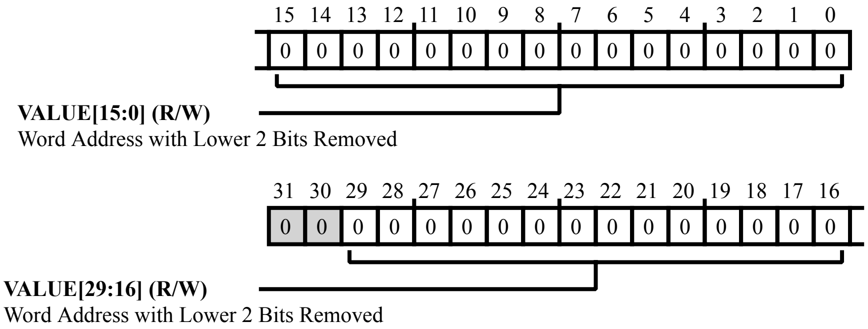

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Word Address with Lower 2 Bits Removed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Word Address with Lower 2 Bits Removed

Figure 51-24: FIR_OUTBASE Register Diagram

Table 51-25: FIR_OUTBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The `FIR_OUTBASE.VALUE` bit field contains the word address with the lower 2 bits removed.

FIR Output Data Count Register

The FIR_OUTCNT register contains the 16-bit output buffer count.

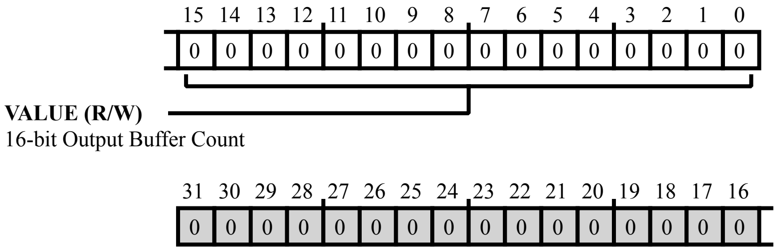

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Output Buffer Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-25: FIR_OUTCNT Register Diagram

Table 51-26: FIR_OUTCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Buffer Count. The `FIR_OUTCNT.VALUE` bit field contains the 16-bit output buffer count.

FIR Output Data Index Register

The `FIR_OUTIDX` register contains the output word address with the lower two bits removed.

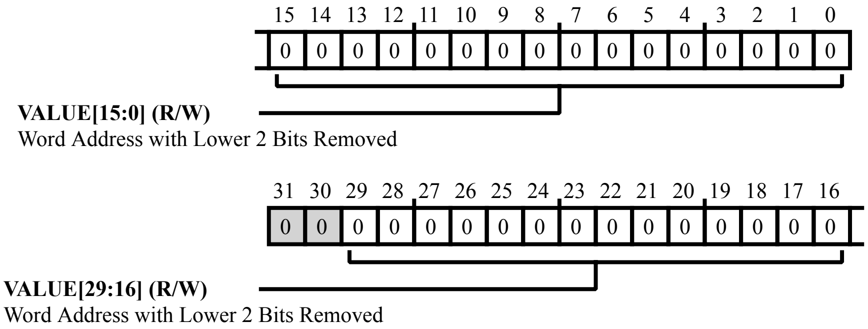

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Word Address with Lower 2 Bits Removed
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Word Address with Lower 2 Bits Removed

Figure 51-26: FIR_OUTIDX Register Diagram

Table 51-27: FIR_OUTIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Word Address with Lower 2 Bits Removed. The `FIR_OUTIDX.VALUE` bit field contains the the word address with the lower 2 bits removed.

FIR Output Data Modifier Register

The `FIR_OUTMOD` register contains the 16-bit output data buffer modifier.

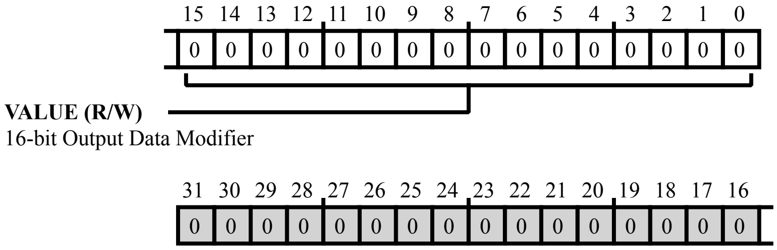

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Output Data Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 51-27: FIR_OUTMOD Register Diagram

Table 51-28: FIR_OUTMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Data Modifier. The `FIR_OUTMOD.VALUE` bit field contains the 16-bit output data modifier.

52 IIR Accelerator (IIR)

The processor includes an Infinite Impulse Response (IIR) filter accelerator implemented in hardware that reduces the processing load on the core, freeing it up for other tasks.

Features

The accelerator supports a maximum of 24 channels. There is support for up to 12 cascaded biquads per channel. The accelerator locally stores all the biquad coefficients of 24 channels. Window size can be configured from 1 (sample based) to 1024. The IIR has the following features.

- Supports IEEE floating point format 32/40-bit
- Supports various rounding modes
- Sample-based or window-based processing
- Up to 12 cascaded biquads per channel
- Up to 24 filter channels available in TDM
- Allows biquad save state storage

NOTE: The IIR accelerator module has local memory which is not accessible by the core during regular operation mode. Unlike previous SHARC processors, the IIR accelerator modules each have access to the system memory (on-chip or off-chip).

Unlike in previous SHARC processors, where only one of the IIR or FIR accelerator can be enabled at a time, the processor can use both the IIR and the FIR accelerators at the same time.

Clocking

The IIR accelerator runs at the maximum speed of SCLK0.

Functional Description

The ***IIR Accelerator Block Diagram*** shows the various blocks of the IIR hardware accelerator. The accelerator has:

- a coefficient memory size of 1440 × 40 bits (12 biquads × 24 channels × 5 coefficients)
- a data memory size of 576 × 40 bits (12 biquads × 24 channels × 2 states)
- one MAC unit with an input data buffer to supply data to the MAC

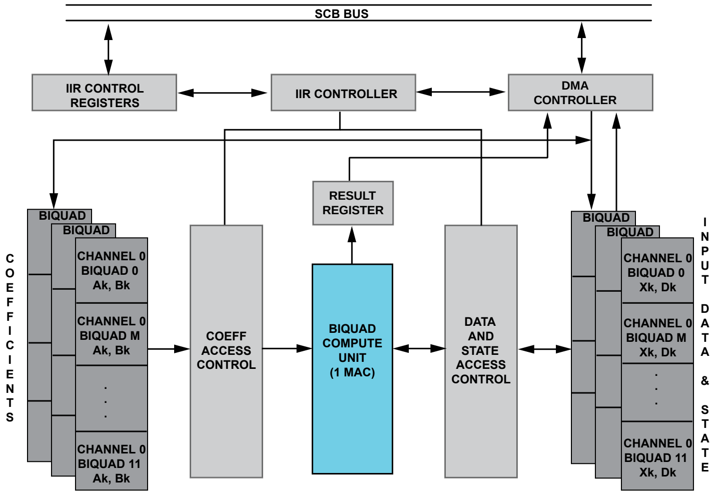

SCB BUS
IIR CONTROL REGISTERS
IIR CONTROLLER
DMA CONTROLLER
RESULT REGISTER
COEFFICIENTS
BIQUAD
BIQUAD
CHANNEL 0 BIQUAD 0 Ak, Bk
CHANNEL 0 BIQUAD M Ak, Bk
CHANNEL 0 BIQUAD 11 Ak, Bk
COEFF ACCESS CONTROL
BIQUAD COMPUTE UNIT (1 MAC)
DATA AND STATE ACCESS CONTROL
CHANNEL 0 BIQUAD 0 Xk, Dk
CHANNEL 0 BIQUAD M Xk, Dk
CHANNEL 0 BIQUAD 11 Xk, Dk
INPUT DATA & STATE

Figure 52-1: IIR Accelerator Block Diagram

The IIR accelerator is implemented using Transposed Direct Form II biquad which has less coefficient sensitivity. The ***Transposed Direct Form II Biquad*** figure shows the signal flow graph for the biquad structure.

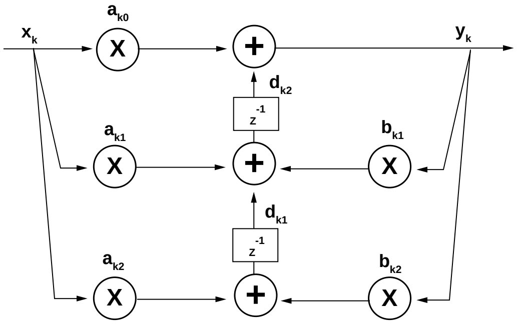

x_k
a_{k0}
y_k
d_{k2}
z^{-1}
a_{k1}
b_{k1}
d_{k1}
z^{-1}
a_{k2}
b_{k2}

Figure 52-2: Transposed Direct Form II Biquad

The accelerator has the following logical subblocks.

- A datapath unit with the following elements:
    - 32/40-bit coefficient memory (Ak, Bk) for storing biquad coefficients
    - 32/40-bit input data (Xk) and state (Dk)
    - One 40/32-bit floating-point multiplier and adder (MAC) unit
    - An input data buffer to efficiently supply data to MAC
    - One 40-bit result register to hold result of biquad

- Configuration registers for controlling various parameters such as the number of biquads, the number of channels, interrupt control, and DMA control
- A core access interface for writing the DMA/filter configuration registers and for reading the status registers
- A DMA bus interface for transferring data to and from the accelerator. This interface is also used to preload the coefficients (Ak, Bk) and state (Dk) at startup.
- DMA configuration registers for the transfer of input data, output data, and coefficients

ADSP-SC58x IIR Register List

The IIR module reduces the processing load on the core. For more information on IIR functionality, see the IIR register descriptions.

Table 52-1: ADSP-SC58x IIR Register List

Name	Description
`IIR_CHNPTR`	Chain Pointer Register
`IIR_COEFIDX`	Coefficient Buffer Index Register
`IIR_COEFLEN`	Coefficient Buffer Length Register
`IIR_COEFMOD`	Coefficient Index Modifier Register
`IIR_CTL1`	Global Control Register
`IIR_CTL2`	Channel Control Register
`IIR_DBG_ADDR`	IIR Debug Address Register
`IIR_DBG_CTL`	IIR Debug Control Register
`IIR_DBG_RDDAT_HI`	IIR Debug Read Data High Register
`IIR_DBG_RDDAT_LO`	IIR Debug Read Data Low Register
`IIR_DBG_WRDAT_HI`	IIR Debug Write Data High Register
`IIR_DBG_WRDAT_LO`	IIR Debug Write Data Low Register
`IIR_DMASTAT`	DMA Status Register
`IIR_INBASE`	Input Buffer Base Register
`IIR_INIDX`	Input Data Index Register
`IIR_INLEN`	Input Data Buffer Length Register
`IIR_INMOD`	Input Data Index Modifier Register
`IIR_MACSTAT`	MAC Status Register
`IIR_OUTBASE`	Output Buffer Base Register
`IIR_OUTIDX`	Output Data Buffer Index Register
`IIR_OUTLEN`	IIR Output Data Buffer Length Register

Table 52-1: ADSP-SC58x IIR Register List (Continued)

Name	Description
`IIR_OUTMOD`	IIR Output Data Index Modifier Register

ADSP-SC58x IIR Interrupt List

Table 52-2: ADSP-SC58x IIR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
155	IIR0_DMA	IIR0 DMA	Edge	
156	IIR0_STAT	IIR0 Status	Edge	

ADSP-SC58x IIR Trigger List

Table 52-3: ADSP-SC58x IIR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
61	`IIR0_DMA`	IIR0 DMA	Edge

Table 52-4: ADSP-SC58x IIR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

Multiply and Accumulate (MAC) Unit

The ***IIR MAC Unit*** figure shows a pipelined multiplier and accumulator unit that operates on the data and coefficient fetched from the data and coefficient memory. The MAC can perform either 32-bit floating-point or 40-bit floating-point MAC operations. 32-bit floating-point operations generate 32-bit results and 40-bit floating-point operations generate 40-bit results.

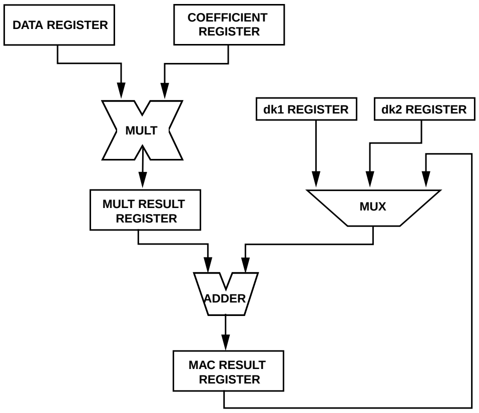


Figure 52-3: IIR MAC Unit

Input Data and Biquad State

The size of data memory is 576 × 40 bits and is used to hold the dk1 and dk2 state of all the biquads locally. The DMA controller fetches the sample data from internal memory and calculates the output as well as the dk1 and dk2 values for each biquad and stores them in local data memory.

Coefficient Memory

The size of coefficient memory is 1440 × 40 bits and is used to store all the coefficients of all the biquads. At start-up, DMA loads the coefficients from system memory into local coefficient memory.

Internal Memory Storage

This section describes the required storage model for the IIR accelerator.

Coefficient Memory Storage

Coefficients and Dk values for a particular biquad BQD[k] should be stored in internal memory in the order: Ak0, Ak1, Bk1, Ak2, Bk2, Dk2, Dk1.

NOTE: The naming convention for the filter coefficients used here is different from the one used in MATLAB. The following conversion should be used when using MATLAB generated coefficients:

(Akx = bx and Bkx = -ax).

Store the coefficients for each biquad in the order:

```
b0, b1, -a1, b2, -a2, dk2, dk1
```

For N biquad stages, store the coefficients in the order:

```
b01, b11, -a11, b21, -a21, dk21, dk11,
b02, b12, -a12, b22, -a22, dk22, dk12,
,......
b0N, b1N, -a1N, b2N, -a2N, dk2N, dk1N.
```

where bxN and axN are the coefficients ([b, a]) for the Nth biquad stage.

Operating Modes

The accelerator operates in Window Processing Mode, 40-Bit Floating-Point Mode and Save Biquad State Mode.

Window Processing Mode

Sample-based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window-based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

40-Bit Floating-Point Mode

In 40-bit floating-point mode, the input data/coefficient is treated as a 40-bit floating-point number. 40-bit floating-point MAC operations generate 40-bit results. This mode can be selected by setting the `IIR_CTL1.FORTYBIT` bit.

Since the DMA bus width is 32 bits, in 40-bit mode the IIR accelerator performs two packed 32-bit accesses to the memory to:

- fetch one 40-bit input or coefficient data, or
- to store one 40-bit output word

The first 32-bit word provides the lower 32 bits and the 8 LSBs of the second 32-bit word provides rest of the upper 8 bits of the complete 40-bit word. The ***32-Bit to 40-Bit Packing*** figure shows the 32–40 bit packing used by accelerator.

NOTE: Overheads could be required to pack the input 40-bit data into the format acceptable by the IIR accelerator and for unpacking the output of accelerator to the format acceptable by the rest of the application.

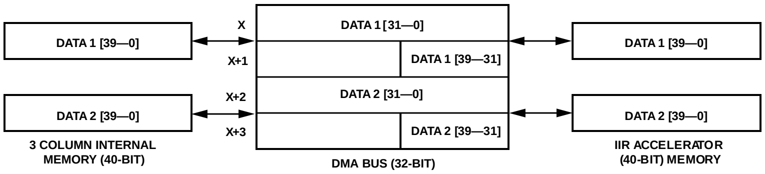


Figure 52-4: 32-Bit to 40-Bit Packing

Save Biquad State Mode

The `IIR_CTL1.SS` bit completely stores the current biquad states in local memory (writes all the DK1 and DK2 states back into the system memory states). This functionality is useful in applications that require fast switching to another high-priority accelerator task—a required IIR to FIR processing transition for example. After resuming, these states can be reloaded and IIR processing can be continued. Note that the DMA status is automatically stored after each iteration.

NOTE: The save state operation cannot be stopped after it starts, even by clearing the `IIR_CTL1.EN` `IIR_CTL1.EN` or `IIR_CTL1.DMAEN` bits. Although the bits would clear on the core side, settings take effect only after the save state operation completes. Therefore, before trying to disable the IIR accelerator, poll the corresponding status bits in the `IIR_DMASTAT` register to ensure the save state operation completed successfully. The following expressions provide the latency due to the save state operation, assuming no higher priority DMA is ON:

- For 32-bit mode: 14 × N + ((8 × M) + 2) × N)
- For 40-bit mode: 14 × N + ((15 × M) + 2) × N)

where N is the number of channels and M is the number of biquads per channel.

NOTE: Write access to any of the IIR accelerator registers loaded by chaining is not allowed while the save state operation is in progress. Attempted writes to these registers could result in the blocking of IOP core reads until the save state operation completes.

Data Transfers

The IIR filter works exclusively through DMA.

IIR Accelerator TCB

The location of the DMA parameters for the next sequence comes from the chain pointer register. This register points to the next set of DMA parameters stored in the system memory of the processor known as TCB. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral.

Chain Assignment

The structure of a TCB is conceptually the same as the structure of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB can point to itself to rerun the same DMA continuously. The FIR accelerator reads each word of the TCB and loads it into the corresponding register. A TCB with a chain pointer register value of zero indicates the end of the chain (no further TCBs are loaded). The IIR accelerator supports circular buffer chained DMA. The ***IIR TCBs for Chained DMA*** table shows the required TCBs for chained DMA. In the table, TCB refers to the start address of the TCB array.

NOTE: In the IIR accelerator DMA, two different TCB loading sequences are available: one TCB loads five parameters for the coefficients (`IIR_CTL2`, `IIR_COEFIDX`, `IIR_COEFMOD`, `IIR_COEFLEN`, and `IIR_CHNPTR`). The second loads 10 parameters for the data (`IIR_CTL2`, `IIR_INIDX`, `IIR_INMOD`, `IIR_INLEN`, `IIR_INBASE`, `IIR_OUTIDX`, `IIR_OUTMOD`, `IIR_OUTLEN`, `IIR_OUTBASE`, and `IIR_CHNPTR`).

Initialize `IIR_CHNPTR` to TCB+12.

Table 52-5: IIR TCBs for Chained DMA

Address	Register
TCB	`IIR_CHNPTR`
TCB + 0x1	`IIR_COEFLEN`
TCB + 0x2	`IIR_COEFMOD`
TCB + 0x3	`IIR_COEFIDX`
TCB + 0x4	`IIR_OUTBASE`
TCB + 0x5	`IIR_OUTLEN`
TCB + 0x6	`IIR_OUTMOD`
TCB + 0x7	`IIR_OUTIDX`
TCB + 0x8	`IIR_INBASE`
TCB + 0x9	`IIR_INLEN`
TCB + 0xA	`IIR_INMOD`
TCB + 0xB	`IIR_INIDX`
TCB + 0xC	`IIR_CTL2`

DMA Access

The IIR accelerator has two DMA channels (accelerator input and output) to connect to the system memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) must be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters (such as number of biquads, window size) for each channel
- DMA parameter register values for the input data
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

NOTE: The chain pointer (`IIR_CHNPTR`) field of the last channel's TCB should point to the first channel's TCB. This configuration exists so that when the IIR accelerator is enabled, the module:

1. Loads the coefficients (Ak, Bk) and state variables (Dk) for all the channels into the local coefficient memory of the FIR, and
2. Loops back to the first channel again to start fetching the input data for processing.

The accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the `IIR_INIDX` (input index register) and `IIR_OUTIDX` (output index register) values to the TCB in memory. Then, data processing can begin from where it left off during the next time slot of that channel.

For 32-bit mode, the write-back values for the index registers are equal to `IIRII` + W and `IIROI` + W.

For 40-bit mode, the write-back values are: `IIR_INIDX` + 2 × W and `IIR_OUTIDX` + 2 × W.

Accelerator input and output channels connect to system memory.

NOTE: The `IIR_CTL2` register is part of the IIR TCB. This configuration allows software to program individual IIR channels having different control attributes.

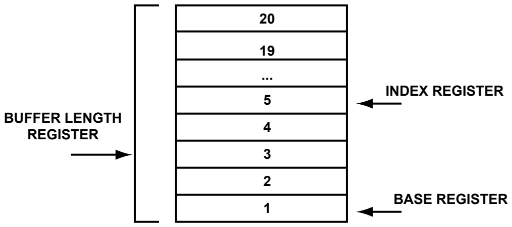


Figure 52-5: Circular Buffer Addressing

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see the *SHARC+ Processor Programming Reference*.

IIR Throughput

Data throughput is one 32-bit data word per SCLK cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two peripheral clock cycles.

IIR throughput is calculated as follows:

Total number of SCLK cycles = (13 × 11 + 5 × B × W) C + B × 11 × 8 × C where:

- B is the number of biquads
- W is the window size
- C is the number of channels
- TCB load = 36 cycles for 32-bit mode and 38 PCLK cycles for 40-bit mode
- 5 × B is the number of cycles to calculate B biquads (This expression does not include the coefficient loading cycles because coefficients need to be loaded only once.)

The loading of input data and writing of output data is pipelined with the computation operation. The expression 5 × B × W includes input data loading, compute, and output data write back operations. This expression does not include the first input data loading, last output data write back, and write back of the updated input and output index registers, the latency of which is included in the TCB load.

NOTE: 14 PCLK cycles are required for TCB loading for coefficients and save state operation.

Interrupts

The ***IIR Interrupt Overview*** table provides the source of interrupt and service instructions for the IIR interrupts.

Table 52-6: IIR Interrupt Overview

Interrupt	Sources	Masking	Service
IIR_DMA IIR_STAT	Input DMA complete Output DMA complete	N/A	ROC from `IIR_DMASTAT`+ RTI instruction
	MAC IEEE floating point exceptions		ROC from `IIR_MACSTAT`+ RTI instruction

Sources

The IIR module drives two interrupt signals, `IIR_DMA` for the DMA status and `IIR_STAT` for the MAC status. The IIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. The interrupt follows the access completion rule, where the interrupt is generated when all data are written back to system memory.

Chained DMA

For chained DMA, if the `PCI` bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the `PCI` bit is set (= 1), then a DMA interrupt is generated for each TCB.

MAC Status

A MAC status interrupt is generated under these conditions:

- Multiplier result zero - Set if multiplier result is zero
- Multiplier result infinity - Set if multiplier result is infinity
- Multiply invalid - Set if multiply operation is invalid

- Adder result zero - Set if adder result is zero
- Adder result infinity - Set if adder result is infinity
- Adder invalid - Set if addition is invalid

Service

When a DMA interrupt occurs, programs can find whether the input or output DMA interrupt occurred by reading the DMA status register (`IIR_DMASTAT`). The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this state (and clear) by reading the MAC status register (`IIR_MACSTAT`). The MAC interrupt status bits are sticky.

The status interrupt sources are derived from the `IIR_MACSTAT` register. A status interrupt can occur due to the last set of MAC operations of a processing iteration that correspond to a particular channel. The interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped when another processing iteration results in a non-zero or valid multiply or add result. However, in this situation it is difficult to isolate whether the interrupt corresponds to the previous processing iteration or that of the current one. This functionality makes using status interrupts impractical.

An alternate way is to poll status bits of the `IIR_MACSTAT` register inside the DMA interrupt service routine. However, consider the behavior of the status bits. The status bits in the `IIR_MACSTAT` registers are sticky. Once a status bit is set, it gets cleared only when the `IIR_MACSTAT` register is read and the previous set of MAC operations resulted in a non-zero, valid output. Therefore, if the last set of MAC operations of a particular processing iteration results in a zero, non-valid output, the corresponding status bit is not cleared, even after reading the `IIR_MACSTAT` register. To avoid a false indication in the next processing iteration, it is necessary to ensure that all the status bits are cleared after the current iteration finishes.

The solution is to read the `IIR_MACSTAT` register twice inside the DMA interrupt service routine. The first read is used to identify which status bits are set. The second read is used to discover if the status bit was set because of the last set of MAC operations. If the status bit was not set because of the last set of MAC operations, it provides a zero result.

If the bit was set because of the last set of MAC operations, clear the status bit by performing a simple dummy FIR processing iteration (biquads = 1 and window size = 1). Choose the appropriate coefficients and input buffer and reading the `IIR_MACSTAT` register after the processing is complete.

Programming Model

The IIR supports up to 24 channels which are time division multiplexed (TDM). Each channel can have a maximum of 12 cascaded biquads. The window size for each channel is configurable using control registers. A window size of 1 corresponds to sample based operation and the maximum window size is 64.

The coefficients are initially stored in system memory and one TCB per channel is created in system memory with each channels' TCB pointing to the next channels'. The TCB also contains channel specific control registers, input data buffer parameters and output data buffer parameters.

NOTE: The TCB of the last channel should point to the TCB of first channel.

The total number of channels is configured using the `IIR_CTL1` register and DMA is enabled.

The procedure that the accelerator uses to process biquads is shown in the ***Biquad Processing Program Flow*** figure and described in the following procedure.

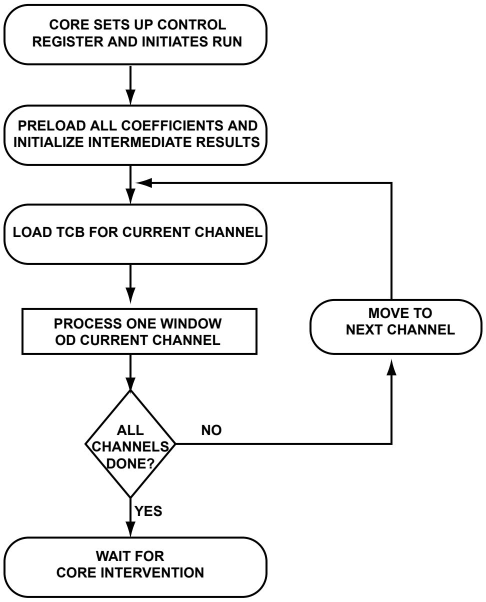

CORE SETS UP CONTROL REGISTER AND INITIATES RUN
PRELOAD ALL COEFFICIENTS AND INITIALIZE INTERMEDIATE RESULTS
LOAD TCB FOR CURRENT CHANNEL
PROCESS ONE WINDOW OD CURRENT CHANNEL
MOVE TO NEXT CHANNEL
ALL CHANNELS DONE?
NO
YES
WAIT FOR CORE INTERVENTION

Figure 52-6: Biquad Processing Program Flow

1. The controller loads all coefficients of all the channels into local storage.
2. Once all the coefficients are loaded, the controller goes to the first biquad of the first channel and calculates the output of the first biquad and updates the intermediate results for that biquad.
3. Then, the accelerator moves to the next biquad of that channel and repeats the process until all the biquads for that channel are completed and the results are stored to memory.
4. This process is repeated with next sample until one window of the corresponding channel is processed.
5. After one window of the channel accelerator is processed, the accelerator moves to the next channel and computes the results.

NOTE: All the addresses programmed in the TCB should correspond to 32-bit address boundaries and shouldn't contain the lower 2 bits (which are assumed to be zeros).

Dynamic Coefficient Processing Notes

The IIR accelerator loads the coefficients for all the channels only once when the IIR accelerator is enabled. In order to re-load the new coefficients, the accelerator has to be disabled and re-enabled.

Writing to Local Memory

1. Clear the `IIR_CTL1.DMAEN` bit.
2. Set the `IIR_DBG_CTL.EN`, `IIR_DBG_CTL.MEM` and `IIR_DBG_CTL.HLD` bits.
3. Set the `IIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write start address to the `IIR_DBG_ADDR` register. If bit 11 is set, coefficient memory is selected.
5. Wait at least 4 *CCLK* cycles.
6. Write data to the `IIR_DBG_WRDAT_LO` register.
7. Write data to the `IIR_DBG_WRDAT_HI` register.

Reading from Local Memory

1. Clear the `IIR_CTL1.DMAEN` bit.
2. Set the `IIR_DBG_CTL.EN`, `IIR_DBG_CTL.MEM` and `IIR_DBG_CTL.HLD` bits.
3. Set the `IIR_DBG_CTL.ADRINC` bit for address auto increment.
4. Write start address to the `IIR_DBG_ADDR` register. If bit 11 is set, coefficient memory is selected.
5. Wait at least 4 *CCLK* cycles.
6. Read data from the `IIR_DBG_RDDAT_LO` register.
7. Read data from the `IIR_DBG_RDDAT_HI` register.

Single Step Mode

Single step mode can be used for debug purposes. An additional debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the `IIR_DBG_CTL.HLD` bit and enable the `IIR_DBG_CTL.EN` and `IIR_DBG_CTL.RUN` bits.
3. Program the IIR module according to the application.
4. In single step each iteration is updated in the emulator session.

Save Biquad State of the IIR

The following steps are required to resume IIR processing after being interrupted by another accelerator module.

1. When starting the accelerator for the first time, set the `IIR_CTL1.EN`, `IIR_CTL1.DMAEN` and `IIR_CTL1.SS` bits.
2. The core waits for the first set of IIR processing to conclude or performs some other task.

3. The accelerator writes back the updated DMA index registers and the updated Dk values after the processing completes.
4. Disable the accelerator by clearing the `IIR_CTL1.EN` bit. Optionally, clear the `IIR_CTL1.DMAEN` bit.
5. The core and accelerator wait for the next set of data to be ready. (The FIR/FFT accelerator can be used for a completely different purpose during this time.)
6. Once the next block is ready for processing, enable the IIR accelerator again by setting the `IIR_CTL1.EN` and `IIR_CTL1.DMAEN` bits. The coefficients and the Dk values are re-loaded back into the local memory.
7. The core waits for the current set of IIR processing to conclude or performs some other task.

Programming Example

In this example, an application needs IIR filtering for two channels of data; channel 1 has six biquads and channel 2 has eight biquads. The window size for all channels is 32.

1. Create a circular buffer in system memory for each channel's data. The buffer should be large enough to avoid overwriting data before it is processed by the accelerator.
2. Configure system memory buffers containing the 6 × 5 coefficients and the 6 × 2 Dk values for the channel 1 biquads, and the 8 × 5 coefficients and 8 × 2 Dk values of the channel 2 biquads.
3. Configure two TCBs in system memory with each channel's chain pointer entry pointing to the next channel's and the last channel's chain pointer entry pointing to the first in a circular fashion.
4. Program the `IIR_CTL2` register to use channel 1 TCB for 6 biquads and a window size of 32, and channel 2 for 8 biquads and a window size of 32.
5. Configure the index, modifier, and length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer and output data buffer.

   The location of the first channel's TCB is written to the chain pointer register in the accelerator.
6. Program the global control register `IIR_CTL1.CH` bit for 2 channels.
   a. The accelerator starts and loads the first channel's TCB, loads coefficients and Dk values of all the 6 biquads into local storage, then loads the TCB of the second channel, and finally loads coefficients and Dk values of all the 8 biquads.
   b. Once all the coefficients and Dk values are loaded, the controller loads the TCB of first channel and fetches the input sample. It then starts calculating the first biquad of the first channel.
   c. The accelerator calculates the output of the first biquad and then updates the intermediate results for that biquad. Then it moves to the next biquad of that channel and repeats the biquad processing until all the biquads for that channel are done and the final result is stored to memory.
   d. The accelerator repeats this process with next sample until one window of the corresponding channel is processed. Once the window is done, the accelerator saves the index values to memory and moves to the next channel. After both channels are done, the accelerator waits for core intervention.

ADSP-SC58x IIR Register Descriptions

The IIR filter accelerator (IIR) contains the following registers.

Table 52-7: ADSP-SC58x IIR Register List

Name	Description
`IIR_CHNPTR`	Chain Pointer Register
`IIR_COEFIDX`	Coefficient Buffer Index Register
`IIR_COEFLEN`	Coefficient Buffer Length Register
`IIR_COEFMOD`	Coefficient Index Modifier Register
`IIR_CTL1`	Global Control Register
`IIR_CTL2`	Channel Control Register
`IIR_DBG_ADDR`	IIR Debug Address Register
`IIR_DBG_CTL`	IIR Debug Control Register
`IIR_DBG_RDDAT_HI`	IIR Debug Read Data High Register
`IIR_DBG_RDDAT_LO`	IIR Debug Read Data Low Register
`IIR_DBG_WRDAT_HI`	IIR Debug Write Data High Register
`IIR_DBG_WRDAT_LO`	IIR Debug Write Data Low Register
`IIR_DMASTAT`	DMA Status Register
`IIR_INBASE`	Input Buffer Base Register
`IIR_INIDX`	Input Data Index Register
`IIR_INLEN`	Input Data Buffer Length Register
`IIR_INMOD`	Input Data Index Modifier Register
`IIR_MACSTAT`	MAC Status Register
`IIR_OUTBASE`	Output Buffer Base Register
`IIR_OUTIDX`	Output Data Buffer Index Register
`IIR_OUTLEN`	IIR Output Data Buffer Length Register
`IIR_OUTMOD`	IIR Output Data Index Modifier Register

Chain Pointer Register

The `IIR_CHNPTR` register should be written with word address without the lower 2 bits.

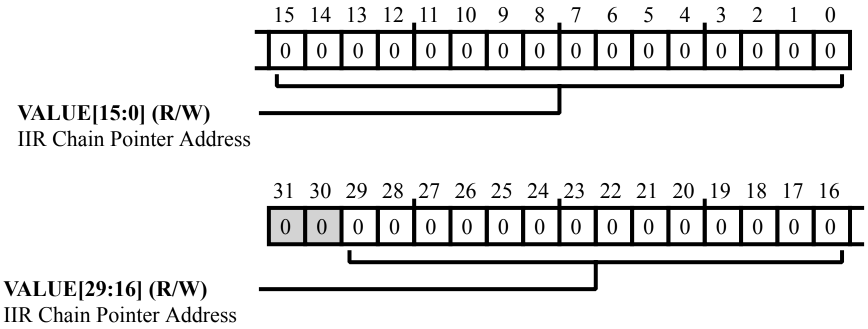

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
IIR Chain Pointer Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
IIR Chain Pointer Address

Figure 52-7: IIR_CHNPTR Register Diagram

Table 52-8: IIR_CHNPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	IIR Chain Pointer Address. The `IIR_CHNPTR.VALUE` bit field contains the chain pointer address.

Coefficient Buffer Index Register

The `IIR_COEFIDX` register contains the word address with the lower 2 bits removed.

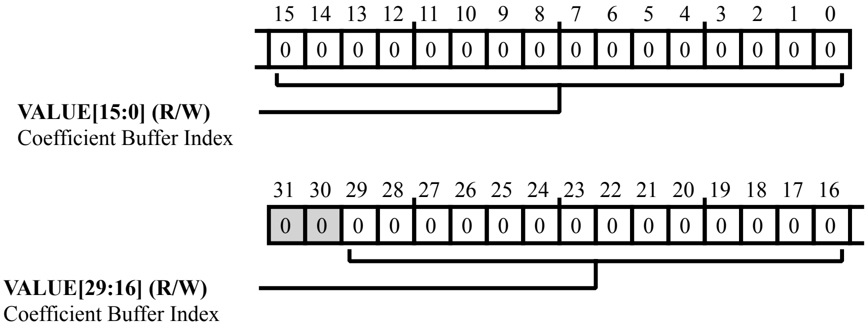

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Coefficient Buffer Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Coefficient Buffer Index

Figure 52-8: IIR_COEFIDX Register Diagram

Table 52-9: IIR_COEFIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Coefficient Buffer Index. The `IIR_COEFIDX.VALUE` bit field provides the coefficient buffer index.

Coefficient Buffer Length Register

The `IIR_COEFLEN` register provides the coefficient buffer length.

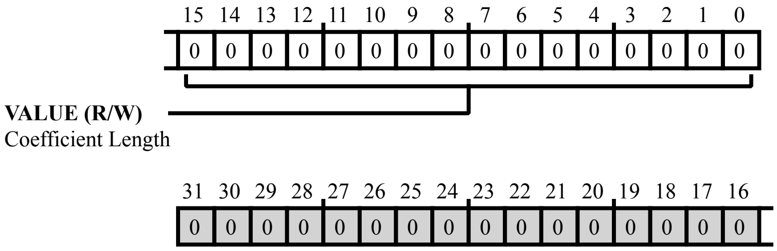

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Coefficient Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-9: IIR_COEFLEN Register Diagram

Table 52-10: IIR_COEFLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Coefficient Length. The `IIR_COEFLEN.VALUE` bit field provides the coefficient buffer length.

Coefficient Index Modifier Register

The `IIR_COEFMOD` register provides the coefficient index modifier.

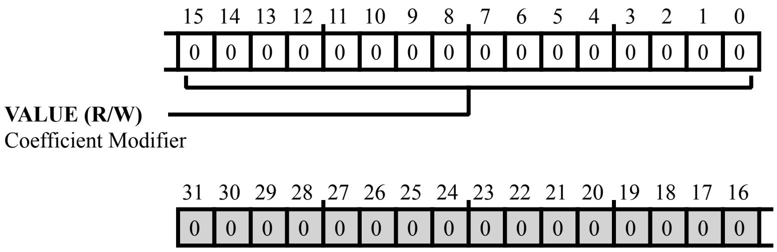

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Coefficient Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-10: IIR_COEFMOD Register Diagram

Table 52-11: IIR_COEFMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Coefficient Modifier. The `IIR_COEFMOD.VALUE` bit field provides the coefficient modifier.

Global Control Register

The IIR_CTL1 register is used to configure the global parameters for the accelerator. These parameters include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

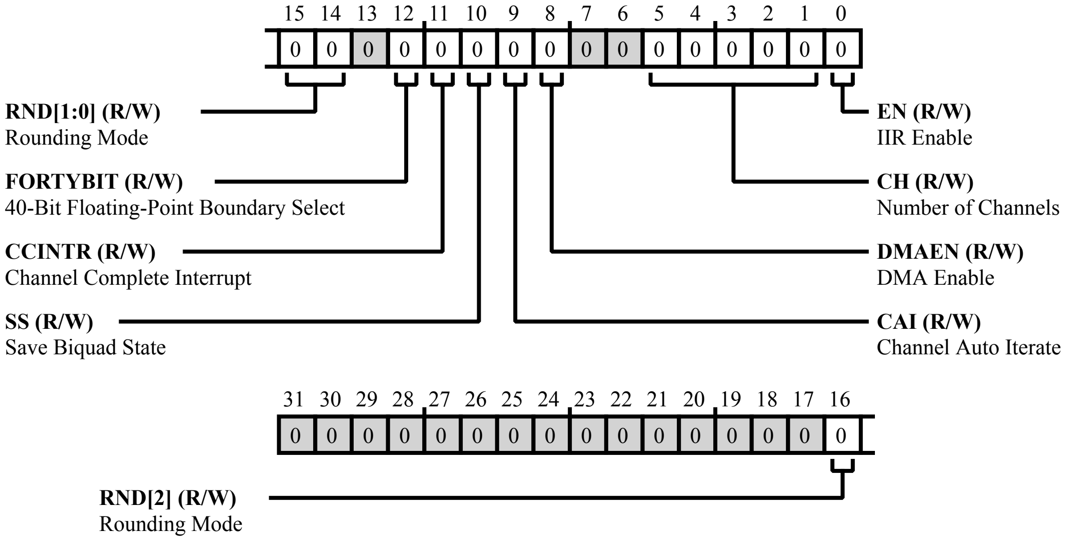

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RND[1:0] (R/W)
Rounding Mode
FORTYBIT (R/W)
40-Bit Floating-Point Boundary Select
CCINTR (R/W)
Channel Complete Interrupt
SS (R/W)
Save Biquad State
EN (R/W)
IIR Enable
CH (R/W)
Number of Channels
DMAEN (R/W)
DMA Enable
CAI (R/W)
Channel Auto Iterate
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RND[2] (R/W)
Rounding Mode

Figure 52-11: IIR_CTL1 Register Diagram

Table 52-12: IIR_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	RND	Rounding Mode. The `IIR_CTL1.RND` bit field selects the rounding mode for floating-point format.	
		0	IEEE round to nearest (even)
		1	IEEE round to zero 010 = IEEE round to +ve infinity 011 = IEEE round to -ve infinity 100 = Round to nearest Up 101 = Round away from zero 110 = Reserved 111 = Reserved
12 (R/W)	FORTYBIT	40-Bit Floating-Point Boundary Select. The `IIR_CTL1.FORTYBIT` bit selects between 32-bit IEEE floating-point format or 40-bit IEEE floating-point format.	
		0	32-bit IEEE floating-point
		1	40-bit IEEE floating-point

Table 52-12: IIR_CTL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	CCINTR	Channel Complete Interrupt. The `IIR_CTL1.CCINTR` bit configures the channel complete interrupt to generate when all channels are done or after each channel is done.	
		0	Interrupt is generated only when all channels are done (default)
		1	Interrupt is generated after each channels is done (default)
10 (R/W)	SS	Save Biquad State. The `IIR_CTL1.SS` bit configures the accelerator to store the Dk register settings into the internal memory. This can be used to save the biquad states before switching to another high priority accelerator task.	
9 (R/W)	CAI	Channel Auto Iterate. The `IIR_CTL1.CAI` bit sets whether TDM processing stops (idle) once all channels complete processing or moves to first channel and continues TDM processing in a loop when all channels complete processing.	
		0	TDM processing stops (idle) once all channels complete processing
		1	Moves to first channel and continues TDM processing in a loop when all channels complete processing
8 (R/W)	DMAEN	DMA Enable. The `IIR_CTL1.DMAEN` bit enables DMA on the accelerator.	
		0	Disable
		1	Enable
5:1 (R/W)	CH	Number of Channels. The `IIR_CTL1.CH` bit field configures the number of channels and is programmable between 0-23 (channels = NCH + 1).	
0 (R/W)	EN	IIR Enable. The `IIR_CTL1.EN` bit enables or disables the IIR accelerator.	
		0	IIR disabled
		1	IIR enabled

Channel Control Register

The `IIR_CTL2` register is used to configure the channel specific parameters. These parameters include the number of biquads and window size.

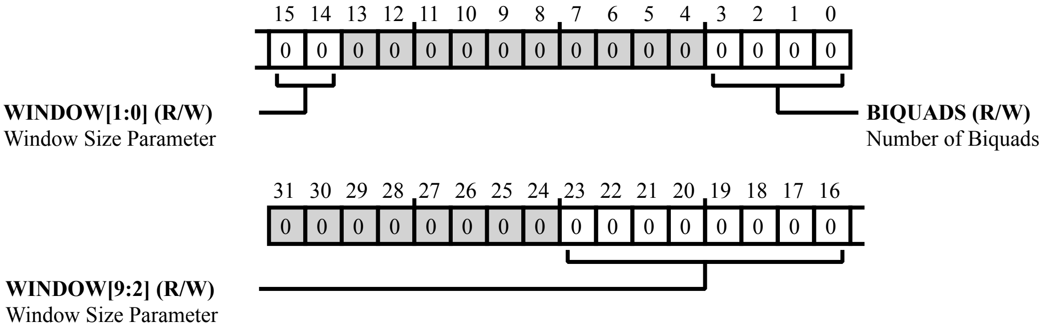

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WINDOW[1:0] (R/W)
Window Size Parameter
BIQUADS (R/W)
Number of Biquads
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
WINDOW[9:2] (R/W)
Window Size Parameter

Figure 52-12: IIR_CTL2 Register Diagram

Table 52-13: IIR_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:14 (R/W)	WINDOW	Window Size Parameter. The `IIR_CTL2.WINDOW` bit field configures the window size which specifies the number of sample/block to process (sample based processing = window size of 1). This bit field should be programmed to "actual window size required -1". For example, for sample based processing this bit field should be programmed to 0.
3:0 (R/W)	BIQUADS	Number of Biquads. The `IIR_CTL2.BIQUADS` bit field configures the number of biquads and is programmable between 0-11 (number of Biquads = BIQUADS + 1).

IIR Debug Address Register

The `IIR_DBG_ADDR` register holds the debug address. If bit 11 is set, coefficient memory is selected.

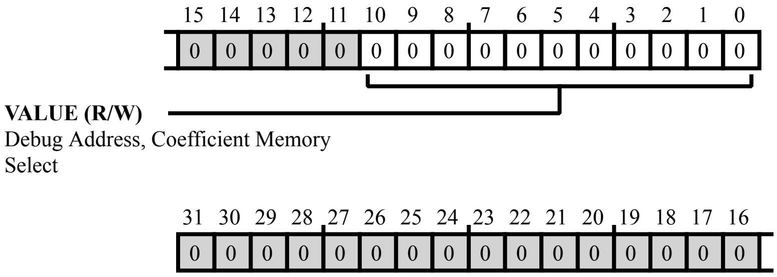

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Debug Address, Coefficient Memory Select
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-13: IIR_DBG_ADDR Register Diagram

Table 52-14: IIR_DBG_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Debug Address, Coefficient Memory Select. The `IIR_DBG_ADDR.VALUE` bit field holds the debug address (bits 0-10). Bit 11 configures whether the memory access is to coefficient memory (=0) or to delay line memory (=1).

IIR Debug Control Register

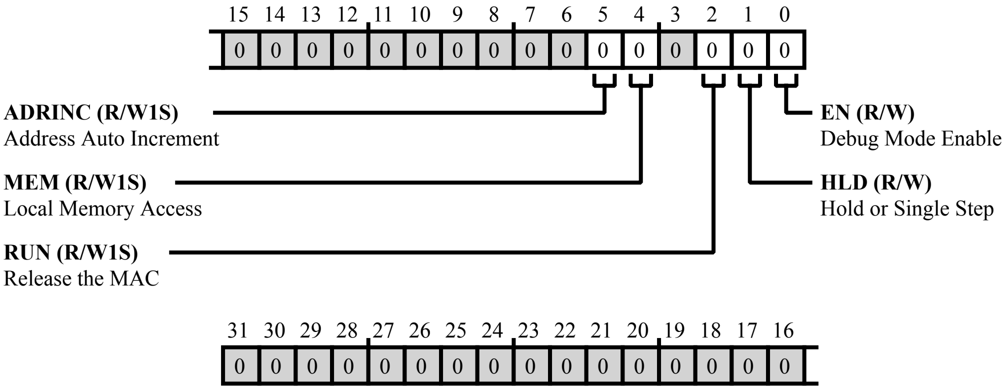

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADRINC (R/W1S)
Address Auto Increment
MEM (R/W1S)
Local Memory Access
RUN (R/W1S)
Release the MAC
EN (R/W)
Debug Mode Enable
HLD (R/W)
Hold or Single Step
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-14: IIR_DBG_CTL Register Diagram

Table 52-15: IIR_DBG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1S)	ADRINC	Address Auto Increment. The `IIR_DBG_CTL.ADRINC` bit allows the address register to auto increment on `IIR_DBG_WRDAT_HI`/`IIR_DBG_WRDAT_LO` writes and `IIR_DBG_RDDAT_HI`/`IIR_DBG_RDDAT_LO` reads.	
4 (R/W1S)	MEM	Local Memory Access. The `IIR_DBG_CTL.MEM` bit allows the data and coefficients memory to be indirectly accessed.	
2 (R/W1S)	RUN	Release the MAC. The `IIR_DBG_CTL.RUN` bit releases the MAC and is is self clearing after one IIR clock cycle.	
1 (R/W)	HLD	Hold or Single Step. The `IIR_DBG_CTL.HLD` bit function is based on the `IIR_DBG_CTL.MEM` bit setting. For `IIR_DBG_CTL.MEM` = 0 this bit sets single step. For `IIR_DBG_CTL.MEM` = 1 this bit sets hold data.	
		0	No effect
		1	Single step (IIR_DBGMEM=0) or Hold data (IIR_DBGMEM=1)
0 (R/W)	EN	Debug Mode Enable. The `IIR_DBG_CTL.EN` bit enables debug mode. For local memory access, the `IIR_CTL1` register can be cleared.	
		0	Disable
		1	Enable

IIR Debug Read Data High Register

The `IIR_DBG_RDDAT_HI` register is part of the 40-bit wide debug mode read data register and holds the upper 8 bits.

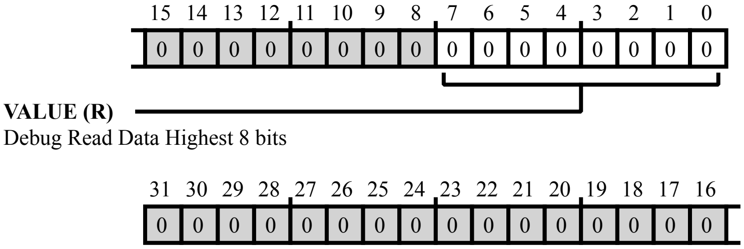

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R)
Debug Read Data Highest 8 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-15: IIR_DBG_RDDAT_HI Register Diagram

Table 52-16: IIR_DBG_RDDAT_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Debug Read Data Highest 8 bits. The `IIR_DBG_RDDAT_HI.VALUE` bit field holds the upper 8-bit read data.

IIR Debug Read Data Low Register

The `IIR_DBG_RDDAT_LO` register is part of the 40-bit wide debug mode read data register and holds the lower 32 bits.

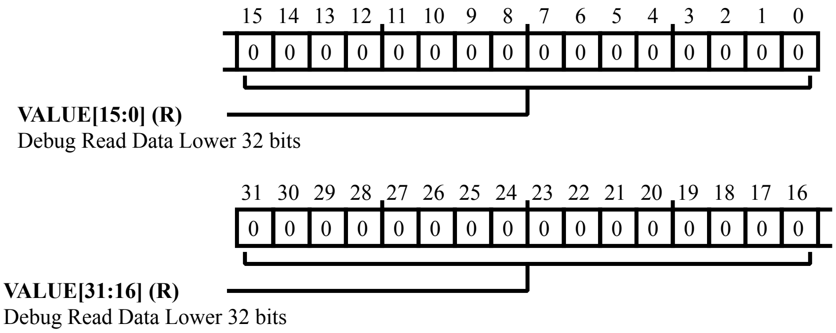

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
Debug Read Data Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
Debug Read Data Lower 32 bits

Figure 52-16: IIR_DBG_RDDAT_LO Register Diagram

Table 52-17: IIR_DBG_RDDAT_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Debug Read Data Lower 32 bits. The `IIR_DBG_RDDAT_LO.VALUE` bit field holds the lower 32-bit read data.

IIR Debug Write Data High Register

The `IIR_DBG_WRDAT_HI` register is part of the 40-bit wide debug mode write data register and holds the upper 8 bits.

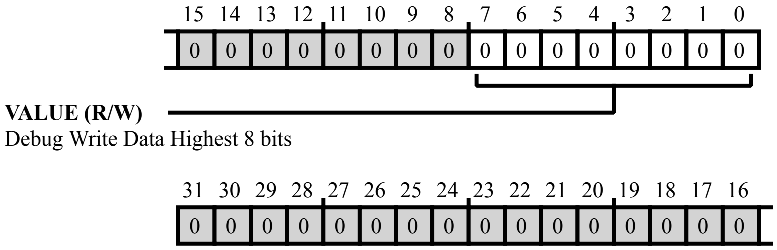

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Debug Write Data Highest 8 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-17: IIR_DBG_WRDAT_HI Register Diagram

Table 52-18: IIR_DBG_WRDAT_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Debug Write Data Highest 8 bits. The `IIR_DBG_WRDAT_HI.VALUE` bit field holds the upper 8-bit write data.

IIR Debug Write Data Low Register

The `IIR_DBG_WRDAT_LO` register is part of the 40-bit wide debug mode write data register and holds the lower 32 bits.

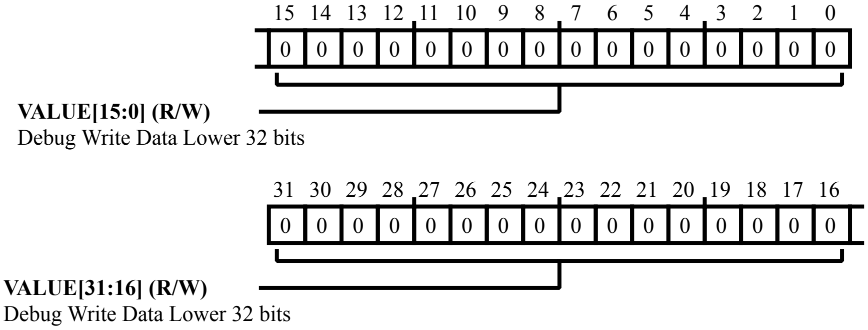

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Debug Write Data Lower 32 bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Debug Write Data Lower 32 bits

Figure 52-18: IIR_DBG_WRDAT_LO Register Diagram

Table 52-19: IIR_DBG_WRDAT_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Debug Write Data Lower 32 bits. The `IIR_DBG_WRDAT_LO.VALUE` bit field holds the lower 32-bit write data.

DMA Status Register

The IIR_DMASTAT registers indicate the status of DMA operations.

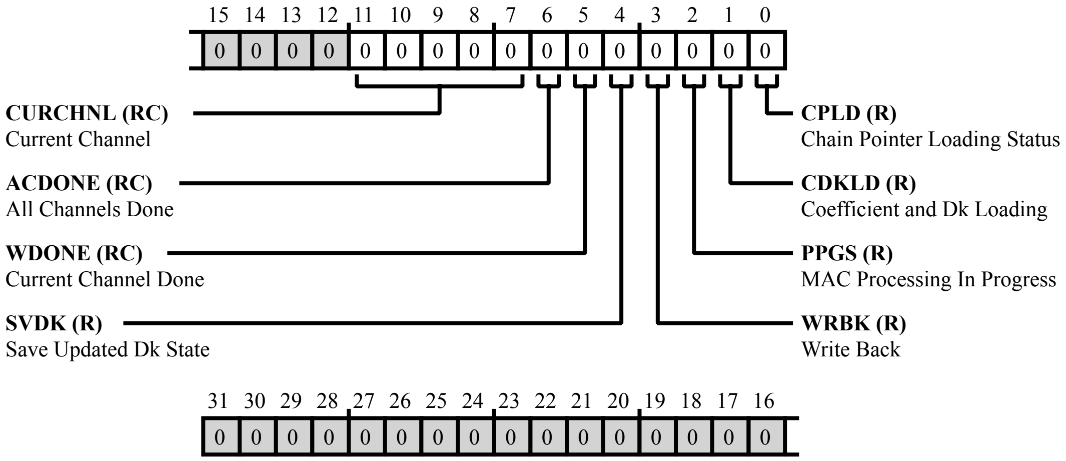

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CURCHNL (RC)
Current Channel
ACDONE (RC)
All Channels Done
WDONE (RC)
Current Channel Done
SVDK (R)
Save Updated Dk State
CPLD (R)
Chain Pointer Loading Status
CDKLD (R)
Coefficient and Dk Loading
PPGS (R)
MAC Processing In Progress
WRBK (R)
Write Back
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-19: IIR_DMASTAT Register Diagram

Table 52-20: IIR_DMASTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:7 (RC/NW)	CURCHNL	Current Channel. The IIR_DMASTAT.CURCHNL bit field indicates the channel that is being processed in the TDM slot. Zero indicates the last slot.
6 (RC/NW)	ACDONE	All Channels Done. The IIR_DMASTAT.ACDONE bit indicates all channels are done processing. Note that the IIR_CTL1.CCINTR bit does not affect this status bit. This bit is sticky and is cleared on register read.
5 (RC/NW)	WDONE	Current Channel Done. The IIR_DMASTAT.WDONE bit indicates the processing of the current channel is complete. Note that the IIR_CTL1.CCINTR bit does not affect this status bit. This bit is sticky and is cleared on register read.

Table 52-20: IIR_DMASTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	SVDK	Save Updated Dk State. If there is more than one channel (`IIR_CTL1.CH`>0), the `IIR_DMASTAT.SVDK` bit toggles between 0 and 1 as it starts and completes the save state operation on one channel at a time. Therefore, this bit is not a reliable indicator of completion of the save state operation for all channels. To ensure graceful completion of the save state operation, programs must poll both the `IIR_DMASTAT.CPLD` and `IIR_DMASTAT.SVDK` bits and ensure (`IIR_DMASTAT.CPLD` OR `IIR_DMASTAT.SVDK`) = 0 after the `IIR_DMASTAT.ACDONE` bit is set. The recommended method for minimizing core intervention is to configure the accelerator to generate an interrupt when the processing of all the channels is complete (the `IIR_CTL1.CCINTR` bit is set), then poll to ensure (`IIR_DMASTAT.CPLD` OR `IIR_DMASTAT.SVDK`) = 0 inside the interrupt service routine. To minimize the interrupt service time, the core can perform unrelated tasks before it starts polling for save state operation completion.
3 (R/NW)	WRBK	Write Back. The `IIR_DMASTAT.WRBK` bit indicates the accelerator is writing back updated index registers.
2 (R/NW)	PPGS	MAC Processing In Progress. The `IIR_DMASTAT.PPGS` bit indicates MAC processing is in progress.
1 (R/NW)	CDKLD	Coefficient and Dk Loading. The `IIR_DMASTAT.CDKLD` bit indicates the coefficient and Dk are loading.
0 (R/NW)	CPLD	Chain Pointer Loading Status. The `IIR_DMASTAT.CPLD` bit indicates the IIR is in the chain pointer load state. 0 State machine not in chain pointer load state 1 State machine in chain pointer load state

Input Buffer Base Register

The `IIR_INBASE` register contains the word address with the lower 2 bits removed.

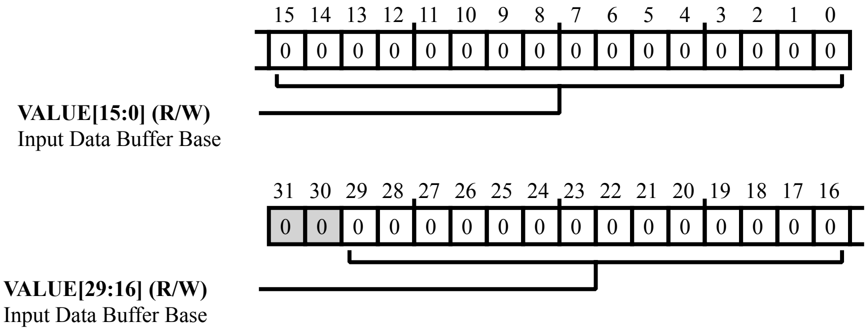

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Input Data Buffer Base
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Input Data Buffer Base

Figure 52-20: IIR_INBASE Register Diagram

Table 52-21: IIR_INBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Input Data Buffer Base. The `IIR_INBASE.VALUE` bit field value is the input data buffer base address.

Input Data Index Register

The `IIR_INIDX` register contains a word address with the lower 2 bits removed.

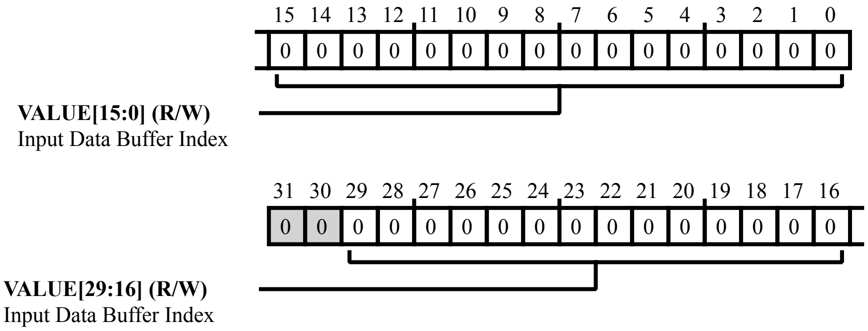

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Input Data Buffer Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Input Data Buffer Index

Figure 52-21: IIR_INIDX Register Diagram

Table 52-22: IIR_INIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Input Data Buffer Index. The `IIR_INIDX.VALUE` bit field value is the input data buffer index.

Input Data Buffer Length Register

The `IIR_INLEN` register provides the input data buffer length.

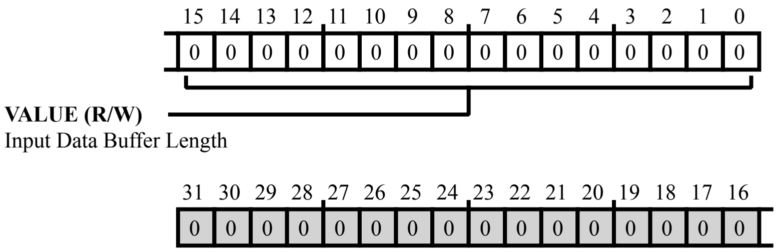

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Input Data Buffer Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-22: IIR_INLEN Register Diagram

Table 52-23: IIR_INLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Input Data Buffer Length. The `IIR_INLEN.VALUE` bit field value is the input data buffer length.

Input Data Index Modifier Register

The `IIR_INMOD` register provides the 16-bit input data buffer index modifier.

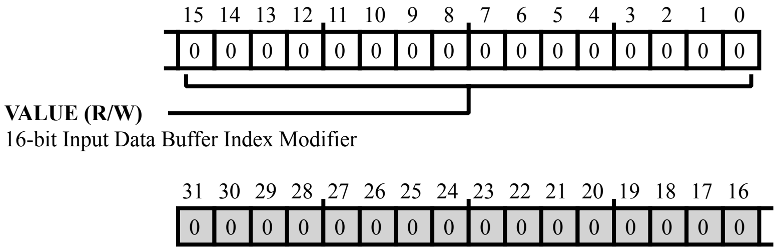

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Input Data Buffer Index Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-23: IIR_INMOD Register Diagram

Table 52-24: IIR_INMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Index Modifier. The `IIR_INMOD.VALUE` bit field value is the 16-bit input data buffer modifier.

MAC Status Register

The IIR_MACSTAT register indicates the status of MAC operations.

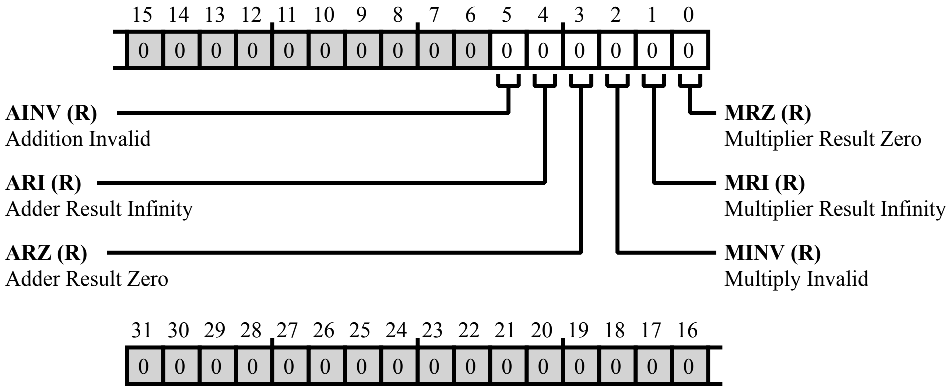

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AINV (R)
Addition Invalid
ARI (R)
Adder Result Infinity
ARZ (R)
Adder Result Zero
MRZ (R)
Multiplier Result Zero
MRI (R)
Multiplier Result Infinity
MINV (R)
Multiply Invalid
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-24: IIR_MACSTAT Register Diagram

Table 52-25: IIR_MACSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	AINV	Addition Invalid. The `IIR_MACSTAT.AINV` bit indicates the addition is invalid.
4 (R/NW)	ARI	Adder Result Infinity. The `IIR_MACSTAT.ARI` bit indicates the adder result is infinity.
3 (R/NW)	ARZ	Adder Result Zero. The `IIR_MACSTAT.ARZ` bit indicates the adder result is zero.
2 (R/NW)	MINV	Multiply Invalid. The `IIR_MACSTAT.MINV` bit indicates the multiply operation is invalid.
1 (R/NW)	MRI	Multiplier Result Infinity. The `IIR_MACSTAT.MRI` bit indicates the multiplier result is infinity.
0 (R/NW)	MRZ	Multiplier Result Zero. The `IIR_MACSTAT.MRZ` bit indicates the multiplier result is zero.

Output Buffer Base Register

The `IIR_OUTBASE` register contains the word address with the lower 2 bits removed.

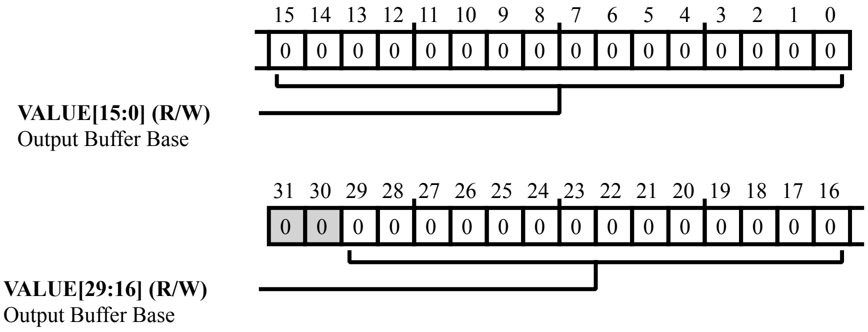

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Output Buffer Base
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Output Buffer Base

Figure 52-25: IIR_OUTBASE Register Diagram

Table 52-26: IIR_OUTBASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Output Buffer Base. The `IIR_OUTBASE.VALUE` bit field provides the output buffer base address.

Output Data Buffer Index Register

The `IIR_OUTIDX` register should be written with word address without the lower 2 bits

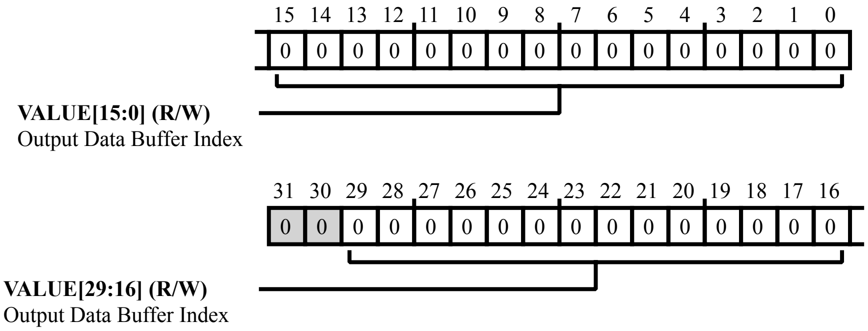

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Output Data Buffer Index
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[29:16] (R/W)
Output Data Buffer Index

Figure 52-26: IIR_OUTIDX Register Diagram

Table 52-27: IIR_OUTIDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:0 (R/W)	VALUE	Output Data Buffer Index. The `IIR_OUTIDX.VALUE` bit field provides the output data buffer index.

IIR Output Data Buffer Length Register

The `IIR_OUTLEN` register provides the output data buffer length.

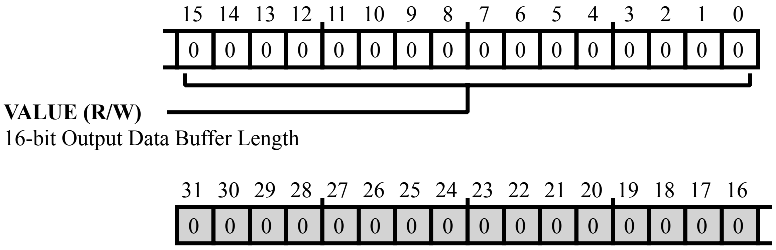

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Output Data Buffer Length
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-27: IIR_OUTLEN Register Diagram

Table 52-28: IIR_OUTLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Output Data Buffer Length. The `IIR_OUTLEN.VALUE` bit field provides the output data buffer length.

IIR Output Data Index Modifier Register

The `IIR_OUTMOD` register provides the output data index modifier.

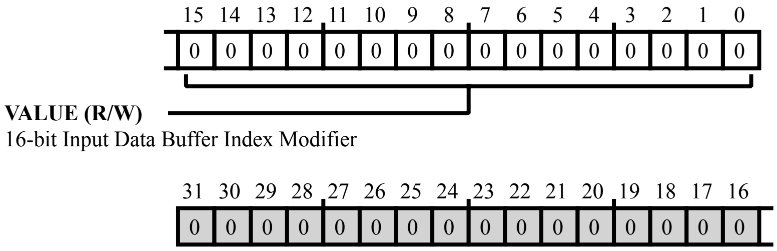

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
16-bit Input Data Buffer Index Modifier
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 52-28: IIR_OUTMOD Register Diagram

Table 52-29: IIR_OUTMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	16-bit Input Data Buffer Index Modifier. The `IIR_OUTMOD.VALUE` bit field provides the output data buffer index modifier.

53 Boot ROM and Booting the Processor

Bootstrapping or booting is the series of events that occur when the system applies power to the processor or when the processor enters a hardware reset state. This section gives an in-depth description of these events and how to integrate an application effectively.

On reset, the processor begins fetching instruction from an internal ROM. The boot code contained within the ROM is designed to facilitate loading an application. The boot code can automatically initialize certain peripherals for communication based on a chosen boot mode, then load an application. For more information on what boot modes are available, see the Boot Modes section. The boot code can efficiently load an entire application, code, and data, into appropriate locations after the development tools repackage the application into a boot stream.

A boot stream is an application or data that the boot-loader tool splits into blocks. A 16-byte header in each block provides instruction to the boot code for processing the associated data. The processor can perform several boot functions, depending on the flags set in the header. For more details on what options are available and a description of the stream format, refer to the Boot Loader Stream section.

The boot ROM provides a mechanism through available non-volatile programmable memory (OTP on this processor) to customize different aspects of the boot process. These customizations include: overriding default boot-peripheral instance, overriding default peripheral-timing parameters and disabling boot modes.

Many of the utilities of the boot code are also available to the application. These utilities include features such as copying memory, comparing memory, or loading another boot stream at run time. The APIs may be used to help ensure that application code is more compatible with future products. For more details on available APIs, see the API Reference section.

In addition to APIs, the boot code provides the ability to define a custom boot mode. This capability helps when support is not available for a desired boot mode. It allows second stage boot loaders for unsupported boot peripherals to leverage a significant amount of the existing boot ROM functionality.

SRAM Requirements

The boot process reserves 8KB of L2 ECC Protected SRAM for dedicated use. This topic describes how the reserved memory region is utilized during boot.

The boot process requires SRAM resources for stack usage and for storage of various data items that require read write access during the boot process. 8KB of L2 ECC protected SRAM is reserved for this purpose. The table below describes the various items stored in this memory region.

Table 53-1: Boot Process SRAM Requirements

Address	Size (Bytes)	Item	Description
0x200BE000	4	Reserved	
0x200BE004	4	Pointer to the `ADI_ROM_BOOT_CONFIG` object	Pointer to the boot configuration structure that is located on the stack. This location can be used to find the location of the boot structure on the stack for debug purposes.
0x200BE008	8	Reserved	
0x200BE010	1024	Internal Intermediate Buffer 0	The first of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x200BE410	1024	Internal Intermediate Buffer 1	The second of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x200BE810	16	`ADI_ROM_BOOT_HEADER` object	Storage location for all the block headers of the boot stream.
0x200BE820	2912	Storage for Secure Boot related descriptors	Contains a number of buffers for the various descriptors used by the Cryptographic Accelerators as well as providing storage for the secure header of a secure boot stream.
0x200BF380	128	Cortex-A5 IRQ, FIQ, ABT and UND stacks	Exception handler stacks for the IRQ, FIQ, ABT and UND exceptions.
0x200BF400	3072	Stack for the boot process	The primary booting cores stack. Any processor core that is mastering a boot operation should locate the stack in this region in order to preserve security in secure boot operations.

NOTE: In order to preserve the security of the product the 8KB region described here is not a bootable region of memory. If the boot process determines that a block of data in the boot stream is targeted towards this memory region the boot process will terminate and enter either the default error handler or if applicable a user defined error handler if installed. This reserved memory region is free for use after the boot process completes. In order to preserve security when using the boot API to boot a secure boot stream, the stack used during the execution of the boot API must be located to the default location in this reserved 8KB region of memory.

Preboot Operations

Preboot is responsible for configuration of all system resources prior to executing the required boot operation.

The steps performed by the preboot process are described here in the order of execution. Numerous stages of the preboot process are conditional based upon the content of `RCU_BCODE`.

NOTE: Upon completion of a power on reset, hard reset and software triggered system reset events the processor is initially running by default in PLL Bypass mode. Partway through the preboot sequence the processor will be brought into full-on mode at the default settings unless the user has provisioned custom CGU settings in the OTP. The Oscillator Watchdog Fault, enabled by default upon reset completion will be disabled at this stage unless Oscillator Watchdogs settings are also supplied.

Start-up Sequence

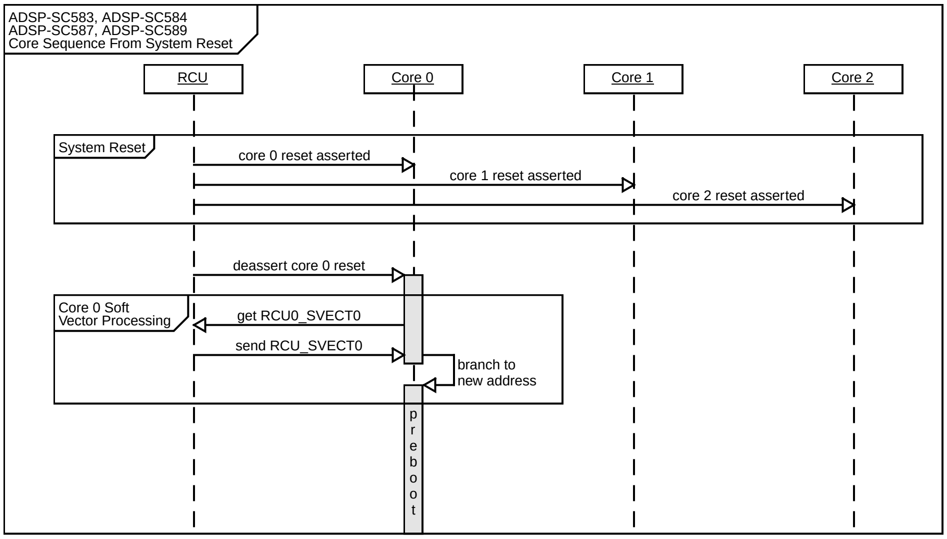
This section describes the initial start-up sequence of all cores in the processor.

Upon completion of a power-on reset, hardware reset or system reset event, only a single core is released from reset and is responsible for managing the boot process. The following sections describe in detail the sequence of events that occur for each of the cores on the various product derivatives.

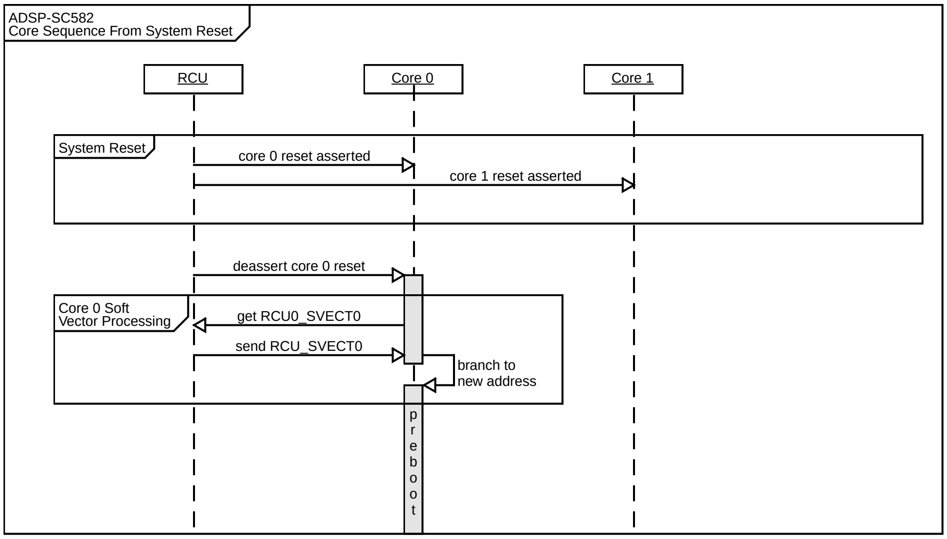
Core Reset Sequencing

System and Hardware reset events result in the processor state being reset and the boot sequence is executed. Only a single core is initially released from reset and starts execution of the preboot software from the boot ROM. The sequencing between the RCU and the various cores is shown for each of the product derivatives.

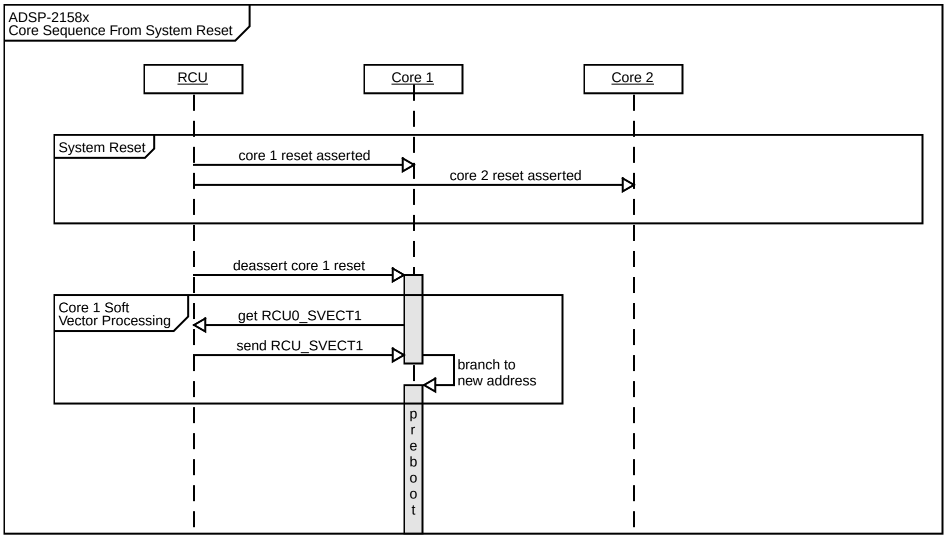
For the ADSP-SC583, ADSP-SC584, ADSP-SC587 and ADSP-SC589 derivatives, Core 0 is responsible for mastering the boot process. Cores 1 and 2 are held in reset until Core 0 releases them from reset during the preboot phase of the boot process. The initial sequence of events after the completion of the reset sequence are shown.


ADSP-SC583, ADSP-SC584
ADSP-SC587, ADSP-SC589
Core Sequence From System Reset
RCU
Core 0
Core 1
Core 2
System Reset
core 0 reset asserted
core 1 reset asserted
core 2 reset asserted
deassert core 0 reset
Core 0 Soft
Vector Processing
get RCU0_SVECT0
send RCU_SVECT0
branch to
new address
preboot

On the ADSP-SC582 processor, Core 0 is responsible for mastering the boot process. Core 1 is held in reset until Core 0 releases the core from reset during the preboot phase of the boot process. The initial sequence of events after the completion of the reset sequence are shown.


ADSP-SC582
Core Sequence From System Reset
RCU
Core 0
Core 1
System Reset
core 0 reset asserted
core 1 reset asserted
deassert core 0 reset
Core 0 Soft
Vector Processing
get RCU0_SVECT0
send RCU_SVECT0
branch to
new address
preboot

The ADSP-2158x derivative contains two cores with IDs of 1 and 2. In this device the Core 1 is responsible for mastering the boot process and Core 2 is held in reset until released by Core 1 during the preboot phase of the boot process. The initial sequence of events after the completion of the reset sequence are shown.


ADSP-2158x
Core Sequence From System Reset
RCU
Core 1
Core 2
System Reset
core 1 reset asserted
core 2 reset asserted
deassert core 1 reset
Core 1 Soft
Vector Processing
get RCU0_SVECT1
send RCU_SVECT1
branch to
new address
preboot

Core 0 Start-up

Describes the initial operations performed by Core 0 immediately after being released from reset and soft vector processing has completed. This topic is only applicable to the ADSP-SC58x processors.

Upon initial release from the reset event the soft vectoring process results in Core 0 executing the boot process as long as the `RCU_SVECT0` register contained the default reset value during the soft vector processing. The core performs the following initial operations:

- Set `VBAR` register to 0x00000020. This step installs a second vector table that is located in the boot ROM that has a reset vector pointing directly to the start of the boot process as opposed to the soft vector routine.
- Disable ECC in all 8 banks via `L2CTL_CTL` of `L2CTL1` and `L2CTL2` controllers that interface to the on-chip ROM memories
- Disable the MMU
- Disable L1 Cache
- Invalidate the TLB
- Disable Branch Prediction
- Read and store the contents of `RCU_STAT`
- Install the stack for the IRQ exception
- Install the stack for the FIQ exception
- Install the stack for the ABORT exception
- Install the stack for the UND exception
- Install the stack for the SVC mode of operation. The boot process runs in SVC mode and the stack installed is the main stack used for the complete boot process.

- Initialize the top 1024 bytes of the stack to initialize the corresponding ECC parity to allow for read access to the initial stack space. This step is bypassed if `RCU_BCODE.NOMEMINIT` is set.
- Call the preboot function passing the previously read `RCU_STAT` contents as an argument

NOTE: The USR and SYS modes of operation are never entered during the boot process and as such their stack are not configured during boot. The boot process runs entirely in SVC mode. User application software is required to install USR and SYS stacks in the secure mode of operation. For systems requiring security, user application software should also install a secure monitor and implement any additional security configuration before transitioning to the non-secure mode of operation.

Core 1 Start-up

Describes the initial operations performed by Core 1 immediately after being released from reset and soft vector processing has completed.

Upon initial release from reset the soft vectoring process results in Core 1 executing from the boot ROM as long as the `RCU_SVECT1` register contained the default reset value during the soft vector processing. The operations the core performs varies depending on the product.

ADSP-SC58x Processors

The following steps apply only to the ADSP-SC58x processors. Core 1 is not responsible for the booting of the processor from reset and thus enters a safe IDLE state allowing access to the cores L1 resources by any other core.

- Disable Software Return feature in the BTB
- Disable the BTB
- Disable ECC in all 8 banks via `L2CTL_CTL` of `L2CTL1` and `L2CTL2` controllers that interface to the on-chip ROM memories
- Set `RCU_MSG.C1IDLE`
- Enter and endless `IDLE` loop

ADSP-2158x Processors

The following steps only apply to the ADSP-2158x processors. Core 1 is responsible for mastering the boot process and the following actions are performed:

- Disable Software Return feature in the `BTB`
- Disable the `BTB`
- Disable ECC in all 8 banks via `L2CTL_CTL` of `L2CTL1` and `L2CTL2` controllers that interface to the on-chip ROM memories
- Disable cache via `MODE2` register
- Flush cache

- Set `IRPTL` to 0x00000000
- Configure primary and secondary DAG configurations to setup the C run-time environment
- Clear `CBUFEN`, `SRD1H`, `SRD1L`, `SRD2H`, `SRD2L`, `ALUSAT`, `TRUNCATE` in the `MODE1` register
- Set `IRPTEN`, `IPERREN`, `DPERREN`, `SPERREN`, in the `MODE1` register
- Set `MMASK` to 0x00000000
- Set `IMASK` to 0x00000000
- Initialize the top 1024 bytes of the stack to initialize the corresponding ECC parity to allow for read access to the initial stack space. This step is bypassed if `RCU_BCODE.NOMEMINIT` is set.
- Call the preboot function passing the contents of `RCU_STAT` as an argument

Table 53-2: Primary/Secondary DAG Configuration for C Run-Time Setup

Primary/Secondary DAG register	Value	Description
`M7, M15`	-1	Dedicated registers must always be set to -1
`M6, M14`	1	Dedicated registers must always be set to 1
`M5, M13`	0	Dedicated registers must always be set to 0
`L0-L5`	0	Preserved registers set initially to 0
`L6, L7`	0	Stack Length Register set initially to 0
`L8 - L15`	0	Preserved registers set initially to 0
`B6, B7`	0x200BF400	Stack Base Registers
`I7`	0x200BFFFC	Stack Pointer
`L6, L7`	0x00000BFD	Stack Length Registers
`I6`	0x200BFFFC	Frame Pointer

Core 2 Start-up

Describes the initial operations performed by Core 2 immediately after being released from reset and soft vector processing has completed. This topic applies to all derivatives except the ADSP-SC582.

Upon initial release from reset the soft vectoring process results in Core 2 executing from the boot ROM as long as the `RCU_SVECT2` register contained the default reset value during the soft vector processing. The operations the core performs varies depending on the product.

The following steps apply only to the ADSP-SC58x processors. Core 1 is not responsible for the booting of the processor from reset and thus enters a safe IDLE state allowing access to the cores L1 resources by any other core.

- Disable Software Return feature in the BTB
- Disable the BTB

- Disable ECC in all 8 banks via `L2CTL_CTL` of `L2CTL1` and `L2CTL2` controllers that interface to the on-chip ROM memories
- Set `RCU_MSG.C2IDLE`
- Enter and endless `IDLE` loop

Idle On Entry

The Idle On Entry implementation allows a means for a debugger to perform a system of the processor and halt the boot code before continuing with any further preboot operations.

When this feature is enabled the processor will execute a WFI/IDLE instruction and then continue once an event such as an emulator exception is serviced.

NOTE: Idle On Entry processing is disabled by default and may be optionally enabled by setting `RCU_BCODE.IDLEONENTRY` prior to performing a system reset operation.

Fault Configuration

Describes the initial fault sources that are enabled allowing the processor to signal a fault to the system.

The following faults are enabled via the `SEC`. Please note that only the faults are enabled, the boot process does not install any `SEC` interrupts.

Table 53-3: Initial Faults installed during Preboot

SEC Fault ID	SEC Fault Name	Description
1	`INTR_SEC0_ERR`	SEC Error
3	`INTR_WDOG0_EXP`	WDOG Expire
4	`INTR_WDOG1_EXP`	WDOG Expire
178	`INTR_CRC0_ERR`	CRC Error
179	`INTR_CRC1_ERR`	CRC Error
5	`INTR_OTPC0_ERR`	OTPC Error
47	`INTR_SOFT3_INT`	`Software Driven Interrupt 3`. This is raised in the boot rom error handler should it be entered.

The SEC is configured to have a fault delay of 0x100 via `SEC_FDLY.COUNT` and `SEC_FSRDLY.COUNT`. This allows for a delay to be implemented before assertion of the fault should a customer error handler be installed and any SEC interrupts enabled and handled for more advanced second stage boot scenarios.

The `SYS_FAULT`pins are configured via the SEC to support both incoming and out going faults by enabling both `SEC_FCTL.FIEN` and `SEC_FCTL.FOEN`. This allows the boot process to capture an incoming fault if asserted by the external system upon handover to the user application after boot.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed via `RCU_BCODE.NOFAULTS`.

Faults are also disabled when the `NOBOOT` boot mode is enabled.

SPU Configuration

The SPU is configured differently depending upon the detected security state of the device. The first operation clears any existing security violations that may be indicated via `SPU_STAT`.

For an open device, a device that has not had the locked bit set in OTP. The boot process makes all peripherals ignore security signals by setting `SPU_SECURECTL.SSECCLR`.

If the security state of the processor is locked then none of the peripherals are configured to ignore security signals and only the following masters are configured to generate secure transactions.

Table 53-4: Locked Processor SPU Secure Masters During Boot

SPU Endpoint ID	Master Name
88	MDMA0 Source DMA Channel
89	MDMA0 Destination DMA Channel
90	MDMA1 Source DMA Channel
91	MDMA1 Destination DMA channel
10	CRC0
11	CRC1
162	PKTE

SMPU Configuration

The SMPU is used to restrict access to various memory regions in the processor. The configuration applied during boot differs depending on the locked state of the processor.

By default the SMPU instances only allow secure read and write transactions. The tables below describe the configurations for the different security states.

Table 53-5: SMPU Configuration

SMPU Instance	SMPU Instance	Open `SMPU_SECURECTL` Value	Locked `SMPU_SECURECTL` Value
SMC	0	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
Core_L2_RAM_Boot_ROM0	2	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
DMA_L2_RAM_Boot_ROM0	3	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)

Table 53-5: SMPU Configuration (Continued)

SMPU Instance	SMPU Instance	Open `SMPU_SECURECTL` Value	Locked `SMPU_SECURECTL` Value
Core_L2_ROM1_Boot_ROM1	4	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
DMA_L2_ROM1_Boot_ROM1	5	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
Core_L2_ROM2_Boot_ROM2	6	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
DMA_L2_ROM2_Boot_ROM2	7	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
DMC0	9	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)
DMC1	10	`SMPU_SECURECTL.WNSEN` | `SMPU_SECURECTL.RNSEN`	0x00000000 (Default Reset Value)

Speculative reads are also disabled by setting `SMPU_CTL.RSDIS`.

Secure Debug Key Processing

In the event the processor is locked, a secure debug key is required to be submitted via the debug tools and matched with a key on the processor. A 128-bit Secure Debug Key must be provisioned by user prior to locking the device.

The secure debug key is read from the OTP memory and then written to the corresponding register in the TAPC. After the key has been written the `TAPC_SDBGKEY_CTL.VALID` bit is set then allowing for a key compare operation to be performed once the debug tools then submit their key.

It is important that the debug tools wait for the boot software to load the key then set the `TAPC_SDBGKEY_CTL.VALID` before submitting the key for comparison.

The 128-bit Secure Debug Key is loaded as follows from the storage area in OTP.

Table 53-6: Secure Debug Key Load Procedure

Secure Debug Key[127:0]	Register
Secure Debug Key[31:0]	`TAPC_SDBGKEY0`
Secure Debug Key[63:32]	`TAPC_SDBGKEY1`
Secure Debug Key[95:94]	`TAPC_SDBGKEY2`
Secure Debug Key[127:96]	`TAPC_SDBGKEY3`

CAUTION: A key of 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF provisioned in OTP will result in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by user software. If the processor fails to boot perhaps due to corrupted firmware then the user will have no debug access. The only way to gain access would be to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger.

CGU Configuration

Reconfigures the internal clocks on the processor for improved boot performance.

The boot process can optionally configure the CGU in order to improve boot performance. The settings to be applied to the CGU are located within the `ADI_ROM_OTP_BOOT_CGU_INFO` structure that has storage allocated in the OTP as part of the `ADI_ROM_OTP_BOOT_INFO` structure.

Typically, CGU configuration would be performed through the use of an Init Block in the boot stream. This provides greatest flexibility. In situations where boot time must be kept to a minimum, provide settings in the OTP that can be applied at this stage of preboot as opposed to during the boot process itself. When a processor is locked, the boot process does not support an Init Block in the boot stream. For a locked processor users must use the OTP in order to reconfigure the clocks without adopting a multi-stage boot strategy.

When the processor is initially released from reset, the CGU is configured for PLL Bypass mode. In order to improve boot performance the boot software reconfigures the CGUsuch that Full-On mode is entered with the default CGU settings. If the user has not supplied settings in OTP to configure the Oscillator Watchdog then the Oscillator watchdog fault, enabled by default after reset, is disabled prior to reconfiguring the CGU. If settings are provisioned in the OTP to configure the Oscillator Watchdog the fault is left enabled and if applicable the CGU is configured per the provided settings. The remainder of the boot process completes in Full-On mode.

Table 53-7: ADI_ROM_OTP_BOOT_CGU_INFO Members

Type	Name	Description
`uint32_t`	`ctl_WEN:1` (bitfield)	Enable write to the `CGU_CTL` register
`uint32_t`	`div_WEN:1` (bitfield)	Enable write to the `CGU_DIV` register
`uint32_t`	`reserved0:1` (bitfield)	Reserved
`uint32_t`	`div_DSEL:5` (bitfield)	`CGU_DIV.DSEL` value
`uint32_t`	`div_CSEL:5` (bitfield)	`CGU_DIV.CSEL` value
`uint32_t`	`div_S0SEL:3` (bitfield)	`CGU_DIV.S0SEL` value
`uint32_t`	`div_SYSSEL:5` (bitfield)	`CGU_DIV.SYSSEL` value
`uint32_t`	`div_S1SEL:3` (bitfield)	`CGU_DIV.S1SEL` value
`uint32_t`	`div_OSEL:7` (bitfield)	`CGU_DIV.OSEL` value
`uint32_t`	`ctl_DF:1` (bitfield)	`CGU_CTL.DF` value
`uint32_t`	`ctl_MSEL:7` (bitfield)	`CGU_CTL.MSEL` value
`uint32_t`	`auto_disable:1` (bitfield)	disable polling on auto-alignment of clocks, NOT RECOMMENDED!
`uint32_t`	`clkoutsel_USBCLKSEL:6` (bitfield)	`CGU_CLKOUTSEL.USBCLKSEL` value

Table 53-7: ADI_ROM_OTP_BOOT_CGU_INFO Members (Continued)

Type	Name	Description
`uint32_t`	`clkoutsel_CLKOUTSEL:5` (bitfield)	`CGU_CLKOUTSEL.CLKOUTSEL` value
`uint32_t`	`clkoutsel_WEN:1` (bitfield)	Enable write to the `CGU_CLKOUTSEL` register
`uint32_t`	`oscwctl0_WEN:1` (bitfield)	Enable write to the `CGU_OSCWDCTL` instance 0 register
`uint32_t`	`oscwctl0_HODF:6` (bitfield)	`CGU_OSCWDCTL.HODF` value
`uint32_t`	`oscwctl0_HODEN:1` (bitfield)	`CGU_OSCWDCTL.HODEN` value
`uint32_t`	`oscwctl0_CNGEN:1` (bitfield)	`CGU_OSCWDCTL.CNGEN` value
`uint32_t`	`oscwctl0_BOUF:5` (bitfield)	`CGU_OSCWDCTL.BOUF` value
`uint32_t`	`oscwctl0_BOUEN:1` (bitfield)	`CGU_OSCWDCTL.BOUEN` value
`uint32_t`	`oscwctl0_FAULTEN:1` (bitfield)	`CGU_OSCWDCTL.FAULTEN` value
`uint32_t`	`oscwctl0_MONDIS:1` (bitfield)	`CGU_OSCWDCTL.MONDIS` value
`uint32_t`	`oscwctl0_FAULTPINDIS:1` (bitfield)	`CGU_OSCWDCTL.FAULTPINDIS` value
`uint32_t`	`oscwctl1_WEN:1` (bitfield)	Enable write to the `CGU_OSCWDCTL` instance 0 register
`uint32_t`	`oscwctl1_HODF:6` (bitfield)	`CGU_OSCWDCTL.HODF` value
`uint32_t`	`oscwctl1_HODEN:1` (bitfield)	`CGU_OSCWDCTL.HODEN` value
`uint32_t`	`oscwctl1_CNGEN:1` (bitfield)	`CGU_OSCWDCTL.CNGEN` value
`uint32_t`	`oscwctl1_BOUF:5` (bitfield)	`CGU_OSCWDCTL.BOUF` value
`uint32_t`	`oscwctl1_BOUEN:1` (bitfield)	`CGU_OSCWDCTL.BOUEN` value
`uint32_t`	`oscwctl1_FAULTEN:1` (bitfield)	`CGU_OSCWDCTL.FAULTEN` value

Table 53-7: ADI_ROM_OTP_BOOT_CGU_INFO Members (Continued)

Type	Name	Description
`uint32_t`	`oscwctl1_MONDIS:1` (bitfield)	`CGU_OSCWDCTL.MONDIS` value
`uint32_t`	`oscwctl1_FAULTPINDIS:1` (bitfield)	`CGU_OSCWDCTL.FAULTPINDIS` value
`uint32_t`	`reserved2:28` (bitfield)	Reserved

If a `CGU_STAT.WDIVERR`, `CGU_STAT.WDFMSERR`, `CGU_STAT.LWERR` or `CGU_STAT.ADDRERR` is either present upon entry to the configuration routine or upon completion of the configuration then the default error handler is called and the boot process terminates.

NOTE: The configuration of the CGU will be bypassed if `RCU_BCODE.NOPREBOOT` is set when this part of the boot process is reached.

Releasing All Cores From Reset

The master booting core releases all other cores from reset allowing them to then run by default into a safe endless loop state. By releasing all other cores from reset any dedicated core L1 memories then become accessible to all cores via the system address space.

In order for one core to load or read data from another cores dedicated L1 memory. The core must be released from the reset state. By default the other cores in the processor execute a safe endless loop in the boot ROM.

L1/L2 Memory Initialization

The processor initializes all parity and ECC protected memories allowing for subsequent read operations to be performed without generation of an ECC or parity error.

The table below describes the methods used to initialize the various parity and ECC supported memories on the processor.

Table 53-8: L1/L2 Memory Initialization

Resource To Fill Memory	Memory Type	Address	Count	Fill Value	Flag Set Upon Completion
Core 1	Core 1 L1 Bank 0	0x00048000	0x5000 (LW)	0x00000000	`RCU_MSG.C1L1INIT`
Core 1	Core 1 L1 Bank 1	0x00058000	0x5000 (LW)	0x00000000	
Core 1	Core 1 L1 Bank 2	0x00060000	0x4000 (LW)	0x00000000	
Core 1	Core 1 L1 Bank 3	0x00070000	0x4000 (LW)	0x00000000	

Table 53-8: L1/L2 Memory Initialization (Continued)

Resource To Fill Memory	Memory Type	Address	Count	Fill Value	Flag Set Upon Completion
Core 2	Core 2 L1 Bank 0	0x00048000	0x5000 (LW)	0x00000000	`RCU_MSG.C2L1INIT`
Core 2	Core 2 L1 Bank 1	0x00058000	0x5000 (LW)	0x00000000	
Core 2	Core 2 L1 Bank 2	0x00060000	0x4000 (LW)	0x00000000	
Core 2	Core 2 L1 Bank 3	0x00070000	0x4000 (LW)	0x00000000	
MDMA0\CRC0	System L2 RAM	0x20080000	0x20000 (BW)	0x00000000	`RCU_MSG.L2INIT`
MDMA1\CRC1	System L2 RAM	0x200A0000	0x1FC00 (BW)	0x00000000	

NOTE: The Memory Initialization process is enabled by default and may be optionally bypassed by setting `RCU_BCODE.NOMEMINIT`.

Default Application Entry Points

The core sets default application entry points for all cores in the processor. This step is performed in the event a boot stream does not contain blocks that specify the application entry point.

The table below defines the default application entry points for each core in the processor.

Table 53-9: RCU_SVECTn Default Application Entry Points

Core ID	Corresponding RCU_SVECTn Register	Application Entry Point
0	`RCU_SVECT0`	0x20080000
1	`RCU_SVECT1`	0x00090004
2	`RCU_SVECT2`	0x00090004

The values defined here get overwritten by First Block located within the boot stream.

NOTE: The initialization of the `RCU_SVECT0` register is enabled by default and may be optionally bypassed by setting `RCU_BCODE.NOVECTINIT`.

Memory Faults Configuration

Describes the memory fault sources that are enabled allowing the processor to signal a fault to the system in the even of a memory error.

The following faults are enabled via the `SEC`. Please note that only the faults are enabled, the boot process does not install any `SEC` interrupts.

Table 53-10: Memory Faults installed during Preboot

SEC Fault ID	SEC Fault Name	Description
8	`INTR_L2CTL0_ECC_ERR`	L2 ECC Error
23	`INTR_C0_L1_PERR`	Core 0 L1 Parity Error

The SEC is configured to have a fault delay of 0x100 via `SEC_FDLY.COUNT` and `SEC_FSRDLY.COUNT`. This allows for a delay to be implemented before assertion of the fault should a customer error handler be installed and any SEC interrupts enabled and handled for more advanced second stage boot scenarios.

The `SYS_FAULT`pins are configured via the SEC to support both incoming and out going faults by enabling both `SEC_FCTL.FIEN` and `SEC_FCTL.FOEN`. This allows the boot process to capture an incoming fault if asserted by the external system upon handover to the user application after boot.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed via `RCU_BCODE.NOFAULTS`.

Faults are also disabled when the `NOBOOT` boot mode is enabled.

NO-BOOT Processing

No-Boot mode is executed on supported devices when selected by the `SYS_BMODE[n]` pins. The boot mode is intended as a recovery boot mode or for debug purposes. The core simply executes in an endless loop in the boot ROM thus terminating further execution of the boot process.

This boot mode is primarily intended for debug sessions when no boot source may be configured. It allows for a debugger to safely connect to the device and take control assuming the debugger has been granted access rights as defined by the processor security implementation.

NOTE: NO-BOOT processing is usually only entered as a result of the boot mode pin sampling resulting in execution of the No-Boot boot mode. This processing can also be optionally enabled by setting `RCU_BCODE.HALT`. The setting of `RCU_BCODE.HALT`can be especially useful for debug sessions to force the execution of the NO-BOOT mode regardless of the `SYS_BMODE[n]` state allowing a user application to be loaded via the debug tools without fear of the image being corrupted as a result of attempting to boot through another source.

SYS_RESOUTb Processing

The `SYS_RESOUT` pin can be used to signal to the external system that the processor is now in a state where it is ready boot

In order to signal to the external system that the processor in now in a configured state and ready to start the boot process, the boot software de-asserts the `SYS_RESOUT` pin via `RCU_CTL.RSTOUTDSRT`.

DMC Configuration

Configures the DMC to allow booting directly to external DDR memory. Primarily intended to support booting to external memory when the processor is locked.

In order to boot to external DDR memories the DMC must be configured. Typically DMC configuration would be done with the support of

Typically, DMC configuration would be performed through the use of an Init Block in the boot stream. This provides greatest flexibility. When a processor is locked however, the boot process does not support an Init Block in the boot stream. For a locked processor users must use the OTP in order to configure the DMC without adopting a multi-stage boot strategy.

The table below provides details of the OTP region that is used to store theDMC configuration to be applied. In addition to the settings described in the table below `ADI_ROM_OTP_BOOT_CFG::dmcEn` must be set in order for the settings to be applied.

There is an additional single bit located in OTP, `ADI_ROM_OTP_BOOT_CFG::dmcInv`, allowing users to invalidate the DMC settings stored in OTP.

NOTE: Once `ADI_ROM_OTP_BOOT_CFG::dmcInv` has been set in OTP there is no means to configure the DMC during the preboot phase.

Table 53-11: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
`uint32_t`	`reserved0:10` (bitfield)	Reserved
`uint32_t`	`ulDDR_DLLCTL:12` (bitfield)	Contents of `DMC_DLLCTL [11:0]`
`uint32_t`	`ulDDR_EMR2:8` (bitfield)	Contents of `DMC_EMR2 [7:0]`
`uint32_t`	`reserved1:2` (bitfield)	Reserved
`uint32_t`	`ulDDR_CFGCTL`	Packed content of `DMC_CTL` Register, `DMC_CFG` registers.
`uint32_t`	`ulDDR_MREMR1`	Packed content of `DMC_EMR1` Register, `DMC_MR` registers.
`uint32_t`	`ulDDR_TR0`	Content of `DMC_TR0`
`uint32_t`	`ulDDR_TR1`	Content of `DMC_TR1`
`uint32_t`	`ulDDR_TR2`	Content of `DMC_TR2`
`uint32_t`	`ulDDR_PHYCTL0`	Content of `DMC_PHY_CTL0`
`uint32_t`	`ulDDR_PHYCTL145`	Packed content of `DMC_PHY_CTL1` , `DMC_PHY_CTL4` and `DMC_PHY_CTL5` registers.
`uint32_t`	`ulDDR_PHYCTL2`	Content of `DMC_PHY_CTL2`
`uint32_t`	`ulDDR_PHYCTL3`	Content of `DMC_PHY_CTL3`

Table 53-11: ADI_ROM_OTP_DMC_CONFIG Members (Continued)

Type	Name	Description
`uint32_t`	`ulDDR_CAL_PADCTL0_PHY_STAT3_0`	Packed content of `DMC_CAL_PADCTL0` , `DMC_PHY_STAT0` , and `DMC_PHY_STAT3` registers.
`uint32_t`	`ulDDR_CAL_PADCTL2`	Content of `DMC_CAL_PADCTL2`

NOTE: The configuration of the DMC will be bypassed if `RCU_BCODE.NOPREBOOT` is set when this part of the boot process is reached.

Bypassing the Boot Process

It is possible to bypass the actual booting process and execute from the address stored in the cores soft vector register. This can be useful when working in emulation sessions as it provides a mechanism to be able to execute directly from an accessible memory that already contains executable code.

The boot process can be bypassed by setting `RCU_BCODE.NOKERNEL` and the core will instead vector to the address stored in the cores corresponding `RCU_SVECT[n]` register instead of calling the required boot mode.

This feature would often be used along with other features such as disabling of memory initialization.

Boot Mode Disable

Specific boot modes can be permanently disabled via OTP resulting in a boot error should the disabled boot mode be enabled via the `SYS_BMODE[n]` pins.

A byte of storage is provided in the OTP allowing for disabling of up to 8 boot modes. The boot mode disable field can be programmed using the `adi_rom_otp_pgm()` routine. The `otp_data::bootModeDisable` member is used in the program operation to disable the various boot modes.

`otp_data::bootModeDisable` Bit Position	Corresponding Boot Mode
0	SPI Master Boot Mode
1	SPI Slave Boot Mode
2	Reserved
3	Reserved
4	Reserved
5	LINKPORT Slave Boot Mode
6	UART Slave Boot Mode
7	Reserved

Boot Command Customization

Boot Command Customization allows for the permanent customization of a particular boot mode. For example it is possible to change the peripheral instance that is used for the boot operation.

Storage is provide in OTP for a `command` item for each of the supported boot modes. The storage is provided in the `ADI_ROM_OTP_BOOT_CMD_INFO` member of `ADI_ROM_OTP_BOOT_INFO` . Refer to the corresponding boot modes boot command description for details on the supported command options.

NOTE: Before programming the boot command to the OTP. It is advisable to thoroughly evaluate the boot command using the `adi_rom_Boot()` API to ensure the boot command provides the desired functionality. Once the command is programmed to OTP there is no means to revert to the original default settings.

Boot Mode Specific SPU Configuration

Prior to performing the actual boot process, the processors SPU resources specific to the boot mode selected are configured. This is performed in the preboot phase as opposed to within the boot mode itself when calling the boot API as it isolates the security functionality of the processor allowing it to be handled specifically by a separate process.

The following additional SPU resources are configured as secure masters according to the boot mode selected.

Table 53-12: Boot Mode Specific SPU Configuration

Boot Mode	SPU Endpoint ID	Master Name
SPI Master Boot (Memory Mapped Mode)	88, 89	MDMA0 Source DMA Channel, MDMA0 Destination DMA Channel
SPI Master Boot (Peripheral Mode)	106, 104, 102	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
SPI Slave Boot	106, 104, 102	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
UART Slave Boot	82, 84, 86	UART 0 Receive DMA, UART1 Receive DMA, UART2 Receive DMA
LINKPORT Slave Boot	5, 6	LINKPORT0 DMA, LINKPORT1 DMA

NOTE: Note that for a given boot mode in the table not all the SPU resources are configured. Only a single peripheral instance is enabled according the peripheral instance selected for boot. For example if the boot command for the boot mode indicates boot from UART0 only the UART0 Receive DMA is configured, the other UART Receive DMAs are not configured for secure access.

Executing the Boot Mode

The boot mode is called using the `adi_rom_Boot()` routine resulting in the fetching and processing of the boot stream from the configured boot source.

The table below provides default parameters passed to each of the supported boot modes. For details on the API usage please refer to `adi_rom_Boot()`.

Table 53-13: Default Boot ROM API Parameters

Boot Mode	`pAddress`	`flags`	`blockCount`	`pHook`	`command`
No Boot	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000000
SPI Master Boot	0x60000000	0x00040000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00010207
SPI Slave Boot	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000212
Reserved	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000000
Reserved	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000000
Reserved	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000000
LINKPORT Slave Boot	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000014
UART Slave Boot	0x00000000	0x00000000	0x00000000	ARM: 0x00000301 SHARC: 0x00b003e5	0x00000013

The hook function installed via `pHook`on this product performs no additional configuration.

Boot Modes

The boot implementation provides built-in support for booting from various peripherals. The ***Booting Modes*** table describes the supported boot modes.

In *slave* boot modes, the processor functions as a slave to any host device. In these modes, the host device usually controls the processor `SYS_HWRST` input. Typically, the host applies the reset sequence and waits until the processor is ready to boot, depending on the peripheral in use, and transmits the boot stream data to the processor. Handshake signals are most likely used to signal to the host that the processor is ready to accept more data.

In a *master* boot modes, the processor controls the peripheral and requests data via the peripheral as and when required.

Individual boot modes can be disabled. For more information about disabling boot modes, see Boot ROM OTP Customizations.

Table 53-14: Booting Modes

SYS_BMODE[2:0]	Boot Source	Description
000	No Boot	The processor does not boot. Rather the boot kernel executes some of the preboot operations then enters and endless `WFI`/`IDLE`state.
001	SPI Master Boot	Boot from integrated Flash memory through the SPI2 peripheral configured for memory mapped mode.
010	SPI Slave Boot	Boot through the SPI2 peripheral configured as a slave
011	Reserved	
100	Reserved	
101	Reserved	
110	LINKPORT Boot	Boot through LINKPORT0 peripheral configured as a slave receiver
111	UART Boot	Boot through UART0 peripheral configured as a slave receiver

No-Boot Mode

No-Boot mode is intended for device-recovery purposes caused by incorrect programming of the boot source memory allowing for target connection through an emulator. Emulation tools can also leverage the No-Boot functionality allowing for debug sessions to run the preboot software prior to loading an application while preventing the boot process from continuing and clobbering data loaded by the emulator.

This boot mode results in a number of preboot operations being performed before then placing the core into a safe endless loop located in the boot ROM. For a complete list of operations performed when No-Boot is selected please refer to Preboot Operations. The core terminates at the NO-BOOT Processing stage of the preboot process.

SPI Master Boot Mode

The SPI master boot routine provides support for booting the processor from SPI flash memories.

The SPI boot mode utilizes a device auto-detection feature that is enabled by default allowing for the boot stream itself to instruct updates to the SPI configuration and the read command used allowing for more efficient transactions.

Boot From External SPI Flash Devices

The SPI boot mode supports booting from 24-bit addressable flash devices. The boot mode uses the MDMA channels by default and configures the SPI flash for memory mapped functionality. Peripheral DMA mode is also supported when calling the boot mode via `adi_rom_Boot()`. Support for 32-bit addressable flash devices can be achieved by disabling the device auto-detection and supplying the required configuration via the `command` parameter.

When auto-device detection is enabled, the SPI memory is initially read using the standard 0x03 SPI read command with a reduced clocking frequency for maximum compatibility. The first nibble of the boot stream is then used to reconfigure the SPI interface and possible the SPI flash. Refer to SPI Device Detection Routine.

NOTE: Support for automatic device detection via the first nibble of the boot stream is not supported when booting secure boot streams. Instead when signing the boot image an attribute can be set in the image header that specifies the configuration to use.

For booting, the SPI memory is connected as shown in the ***SPI Memory Connections*** figure.

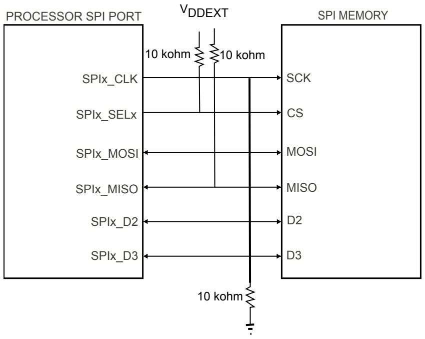


Figure 53-1: SPI Memory Connections

The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in a high-impedance mode such as during reset.

Initialization codes are allowed to manipulate `ADI_ROM_BOOT_CONFIG::dBootCommand` to extend the boot mechanism to a second SPI memory connected to another slave select pin. Updating the field that specifies the slave select signal for use allows the boot process to manage larger boot streams than are able to fit in a single SPI device.

NOTE: If modifying the slave select signal used during the boot process, configure the pin multiplexing to enable the correct functionality for the pin. Once the boot process has proceeded past the configuration function and the boot process has actually started, the boot kernel will not perform any further pin multiplexing operations.

For SPI master boot peripheral mode, the `SPE`, `MSTR`, and `SZ` bits are set in the `SPIx_CTL` register. The `TIMOD=2` bits enable the receive DMA mode. The `CPOL` and `CPHA` bits are set by default, resulting in SPI mode 3. The boot kernel does not allow SPIx hardware to control the `SPI_SEL[n]` pin. Instead, software toggles this pin.

SPI Device Detection Routine

Since the boot mode supports booting from various SPI memories, the boot kernel automatically detects what type of memory is connected. To determine whether the SPI memory device requires an 8, 16, 24, or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The `SPI_MISO` signal requires a pull-up resistor. The routine relies on the fact that memories do not drive their data outputs unless the right number of address bytes are received.

Initially, the boot kernel transmits the read command on the `SPI_MOSI` line. Once the command has been sent, the boot kernel proceeds to transmit a single address byte and waits until the receive FIFO indicates that the buffer is no longer empty. The first received byte is discarded. The boot code then proceeds to issue another address byte while simultaneously receiving a byte. The process continues until a non-0xFF or 0x00 byte is received or until the full 4 address bytes is sent without any valid data being returned.

The receiving of a non-0x00 or 0xFF byte tells the boot code whether the memory device requires 8, 16, 24, 32 address bits. The lower nibble of the received byte is then used to further customize the boot mode. This nibble is referred to as the `BCODE`. The boot code applies settings to the SPI peripheral according to the ***SPI Master Boot BCODE Descriptions***.

If the received value equals 0x00 or 0xFF, it is assumed that the memory device has not driven its data output thus, another zero byte is transmitted and the received data is tested again.

If the value still equals 0xFF, device detection continues. Device detection aborts immediately when a byte different than 0xFF is received. The boot process continues with normal boot operation and it re-issues a command to re-read from address 0. Two read sequences load the first block header. Separate read sequences load further block headers and block payload fields.

The ***SPI Device Detection Principle*** figure illustrates how individual devices behave.

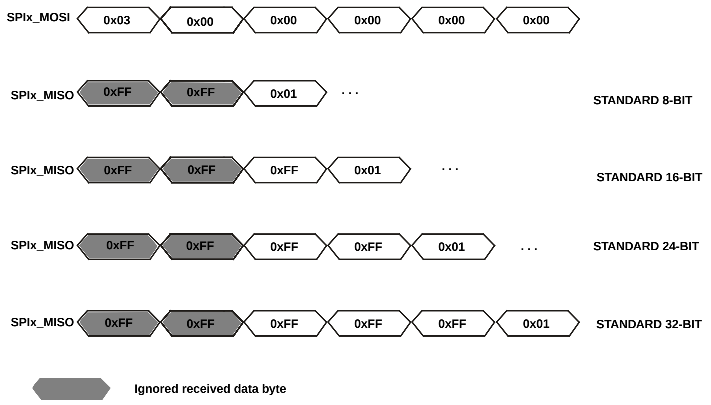

SPIx_MOSI
0x03
0x00
0x00
0x00
0x00
0x00
SPIx_MISO
0xFF
0xFF
0x01
STANDARD 8-BIT
SPIx_MISO
0xFF
0xFF
0xFF
0x01
STANDARD 16-BIT
SPIx_MISO
0xFF
0xFF
0xFF
0xFF
0x01
STANDARD 24-BIT
SPIx_MISO
0xFF
0xFF
0xFF
0xFF
0xFF
0x01
STANDARD 32-BIT
Ignored received data byte

Figure 53-2: SPI Device Detection Principle

Table 53-15: SPI Master Boot BCODE Descriptions

BCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x0	Init	Init	Init	Init	Init	Init	Unused
0x1	STANDARD	0x03	0	1	1	SCLK0/16	Legacy single-bit SPI mode
0x2	STANDARD	0x03	0	1	1	SCLK0/4	Legacy single-bit SPI mode
0x3	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit with dummy address byte

Table 53-15: SPI Master Boot BCODE Descriptions (Continued)

BCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x4	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit with dummy address byte. `SPI_CTL.FMODE` is enabled for full cycle access.
0x5	STANDARD	0x03	0	1	1	SCLK0/3	Legacy single bit. `SPI_CTL.FMODE` is enabled for full cycle access.
0x6	FAST READ	0x0B	1	2	1	SCLK0/1	Single bit with dummy byte. `SPI_CTL.FMODE` is enabled for full cycle access.
0x7	RAPID-S	0x1B	2	2	2	SCLK0/1	Single bit with dummy bytes. `SPI_CTL.FMODE` is enabled for full cycle access.
0x8	DOR	0x3B	1	2	1	SCLK0/2	Dual bit data. `SPI_CTL.FMODE` is enabled for full cycle access.
0x9	DIOR	0xBB	1	2	2	SCLK0/2	Dual data and address. `SPI_CTL.FMODE` is enabled for full cycle access.
0xA	QOR READ (Quad Mode Method 1)	0x6B	1	4	1	SCLK0/2	Quad bit data mode using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xB	QIOR READ (Quad Mode Method 1)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xC	QOR READ (Quad Mode Method 2)	0x6B	1	4	1	SCLK0/2	Quad data using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access
0xD	QIOR READ (Quad Mode Method 2)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access
0xE	QIOR READ (Quad Mode Method 3)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 3. `SPI_CTL.FMODE` is enabled for full cycle access
0xF	Init	Init	Init	Init	Init	Init	Unused

NOTE: For all the above configurations the addressing scheme is also set to a fixed 3-bytes for 24-bit addressable flash support only. The SPI mode byte issued for all the SPI Master peripheral based configurations is 0x00. The mode byte is the first byte transmitted after the address cycles and is used to control the continuous read mode functionality in which the next read operation is not required to issue a command cycle. Continuous read mode is not supported during the boot process.

Supported Quad Mode Enable Methods

The boot rom supports the following methods for enabling quad mode on the SPI flash device

NOTE: Please note that the procedures listed here write to a non-volatile register in the SPI flash. There are typically delays implemented with writes to such registers and these need to be accounted for in the boot time. If the flash supports a non-volatile setting using a different procedure from those supported here then it may be more beneficial to boot initially in dual mode and use an initcode to enable quad mode.

Quad Mode Method 1.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.
2. Issue the Read Configuration Register command (0x3F) and read in the configurations register value.
3. Issue the Write Enable command (0x06).
4. Issue the Read Status Register command (0x05) to verify the write latch status is set.
5. Set bit 7 of the read configuration register value to enable quad mode.
6. Issue the Write Register command (0x3E) and write the value of the configuration register.
7. Issue the Read Status Register command (0x05) until the device is ready and write latch is cleared.
8. Issue the Read Configuration Register command (0x3F) to verify quad mode is enabled.

Quad Mode Method 2.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.
2. Issue the Read Configuration Register command (0x35) and read in the value of the configuration register.
3. Issue the Write Enable command (0x06).
4. Issue the Read Status Register command (0x05) to verify the write latch status is set.
5. Set bit 1 of the read configuration register value to enable quad mode.
6. Issue the Write Register command (0x01) and write the value of the status register and the updated value of the configuration register.
7. Issue the Read Status Register command (0x05) until the device is ready.

Quad Mode Method 3.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.

2. Issue the Write Enable command (0x06).
3. Issue the Read Status Register command (0x05) to verify the write latch status is set.
4. Set bit 6 of the read status register value to enable quad mode.
5. Issue the Write Status command (0x01) and write the value of the status register back to enable quad mode.
6. Issue the Read Status Register command (0x05) until the device is ready.

Run-time API

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 53-16: SPI Master Boot `command`Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
31:28	ROM_BCMD_SPIM_SPEED	SPI clock divider to be used. The value written to the SPI Peripherals clock divider register.	
26:25	ROM_BCMD_SPIM_IOPROT	SPI I/O protocol. If multiple I/O pins are required, then I/O mode is enabled on the flash device. The number of data bits required for the data transfer determines the I/O mode enabled. This field is only applied if NOAUTO is set to 1. Otherwise, the protocol for enabling the additional pins is defined through the BCODE value supplied.	
		0x0	No protocol required to implement multiple I/O
		0x1	Enable quad functionality using protocol 1
		0x2	Enable quad functionality using protocol 2
		0x3	Enable quad functionality using protocol 3
24:22	ROM_BCMD_SPIM_DUMMY	Number of dummy bytes to be issued. Specifies the number of dummy bytes to be issued after the address bytes are issued for the required read command.	
		0x0	Do not issue dummy bytes
		0x1	Issue 1 dummy byte
		0x2	Issue 2 dummy bytes
		0x3	Issue 3 dummy bytes
		0x4	Issue 4 dummy bytes
		0x5	Issue 5 dummy bytes
		0x6	Issue 6 dummy bytes
		0x7	Issue 7 dummy bytes

Table 53-16: SPI Master Boot `command`Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20	ROM_BCMD_SPIM_ADDR	Number of address bytes to be issued. Specifies the number of address bytes that are required to be issued to the SPI flash for the required read command.	
		0x0	Issue 1 address byte
		0x1	Issue 2 address bytes
		0x2	Issue 3 address bytes
		0x3	Issue 4 address bytes
19:16	ROM_BCMD_SPIM_BCODE	Boot mode-specific code. Specifies the boot mode-specific code that can further customize and control the boot process.	
14:12	ROM_BCMD_SPIM_SSEL	SPI Slave select signal. Specifies the SPI slave select signal to be used to enable the SPI Flash. Not all slave selects are available for each SPI port. Refer to the product data sheet for further details.	
		0x0	SEL1
		0x1	SEL2
		0x2	SEL3
		0x3	SEL4
		0x4	SEL5
		0x5	SEL6
		0x6	SEL7
		0x7	Reserved
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2	SPI2
		0x3 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	

Table 53-16: SPI Master Boot `command`Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI
		0x7	Memory Mapped SPI (For use with SPI2 only)

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

SPI Slave Boot Mode

When using SPI slave mode boot, the processor consumes boot data from an external SPI host device. This mode supports single, dual, and quad-bit modes. The boot kernel always starts in single bit mode and can be changed using the appropriate command. The following figures show the hardware configuration for the modes. As in all slave boot modes, the host device controls the `SYS_HWRST` input of the processor.

NOTE: Secure Boot Stream Padding

For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

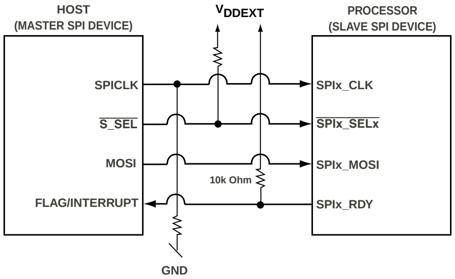

HOST
(MASTER SPI DEVICE)
V_{DDEXT}
PROCESSOR
(SLAVE SPI DEVICE)
SPICLK
$\overline{\text{S\_SEL}}$
MOSI
FLAG/INTERRUPT
SPIx_CLK
$\overline{\text{SPIx\_SELx}}$
SPIx_MOSI
SPIx_RDY
10k Ohm
GND

Figure 53-3: Connection Between Host (SPI Master) and Processor (SPI Slave)

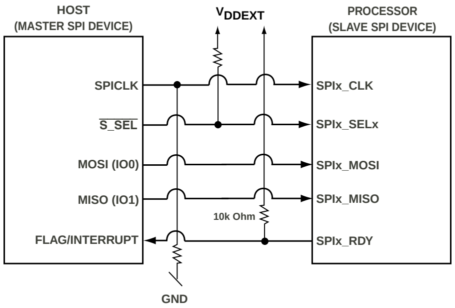

HOST
(MASTER SPI DEVICE)
VDDEXT
PROCESSOR
(SLAVE SPI DEVICE)
SPICLK
S_SEL
MOSI (IO0)
MISO (IO1)
FLAG/INTERRUPT
SPIx_CLK
SPIx_SELx
SPIx_MOSI
SPIx_MISO
SPIx_RDY
10k Ohm
GND

Figure 53-4: Connection Between Host (SPI Master) and Processor (SPI Slave) DIOM

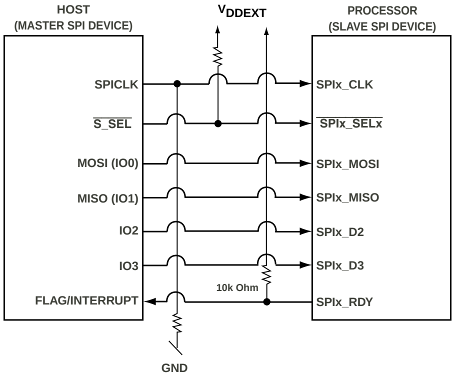

HOST
(MASTER SPI DEVICE)
VDDEXT
PROCESSOR
(SLAVE SPI DEVICE)
SPICLK
S_SEL
MOSI (IO0)
MISO (IO1)
IO2
IO3
FLAG/INTERRUPT
SPIx_CLK
SPIx_SELx
SPIx_MOSI
SPIx_MISO
SPIx_D2
SPIx_D3
SPIx_RDY
10k Ohm
GND

Figure 53-5: Connection Between Host (SPI Master) and Processor (SPI Slave) QSPI

The host drives the SPI clock and is responsible for timing. The host must provide an active-low chip select signal that connects to the `SPIx_SS` input of the processor. It can toggle with each byte transferred or remain low during the entire procedure. 8-bit data is expected and 16-bit mode is not supported.

In SPI slave boot mode, the boot kernel sets the `SPI_CTL.CPHA` bit and clears the `SPI_CTL.CPOL` bit in the `SPI_CTL` register. Therefore the `SPI_MISO` pin is latched on the falling edge of the `SPI_MOSI` pin.

The SPI slave processor detects the correct boot mode from the host SPI device by reading the first byte sent, defined as `SPICMD`. The ***SPICMD Descriptions*** table describes the available codes. These additional bytes must be sent prior to transmitting the data to configure the SPI device accordingly.

The table describes two cases, one in which the host starts in single bit mode, and one in which the host starts in a mode other than single bit.

Table 53-17: SPICMD Descriptions

SPICMD	Description
If host Starts in Single bit Mode	
0x3	Keep single-bit mode
0x7	Switch to dual-bit mode
0xB	Switch to quad-bit mode
If host device starts in DIOM or QSPI	

Table 53-17: SPICMD Descriptions (Continued)

SPICMD	Description
0xAA,0xBF	Switch to dual-bit mode
0xEE,0xEE,0xFE,0xFF	Switch to quad-bit mode

In SPI slave boot mode, `SPIx_RDY` functionality is critical. The `SPIx_RDY` output is used for back pressure and requires a pulling resistor. The boot code requires the `SPIx_RDY` signal function as active-low. The host is only permitted to transfer data when `SPIx_RDY` is in the active state. This functionality allows the processor to hold off the host while the processor is in reset or executing the pre-boot and processor initialization sequences. The SPI is configured to de-assert `SPIx_RDY` when the receive FIFO is filled to 75% or more. The ***SPI Program Flow on the Host Side*** figure illustrates the required program flow on the host side.

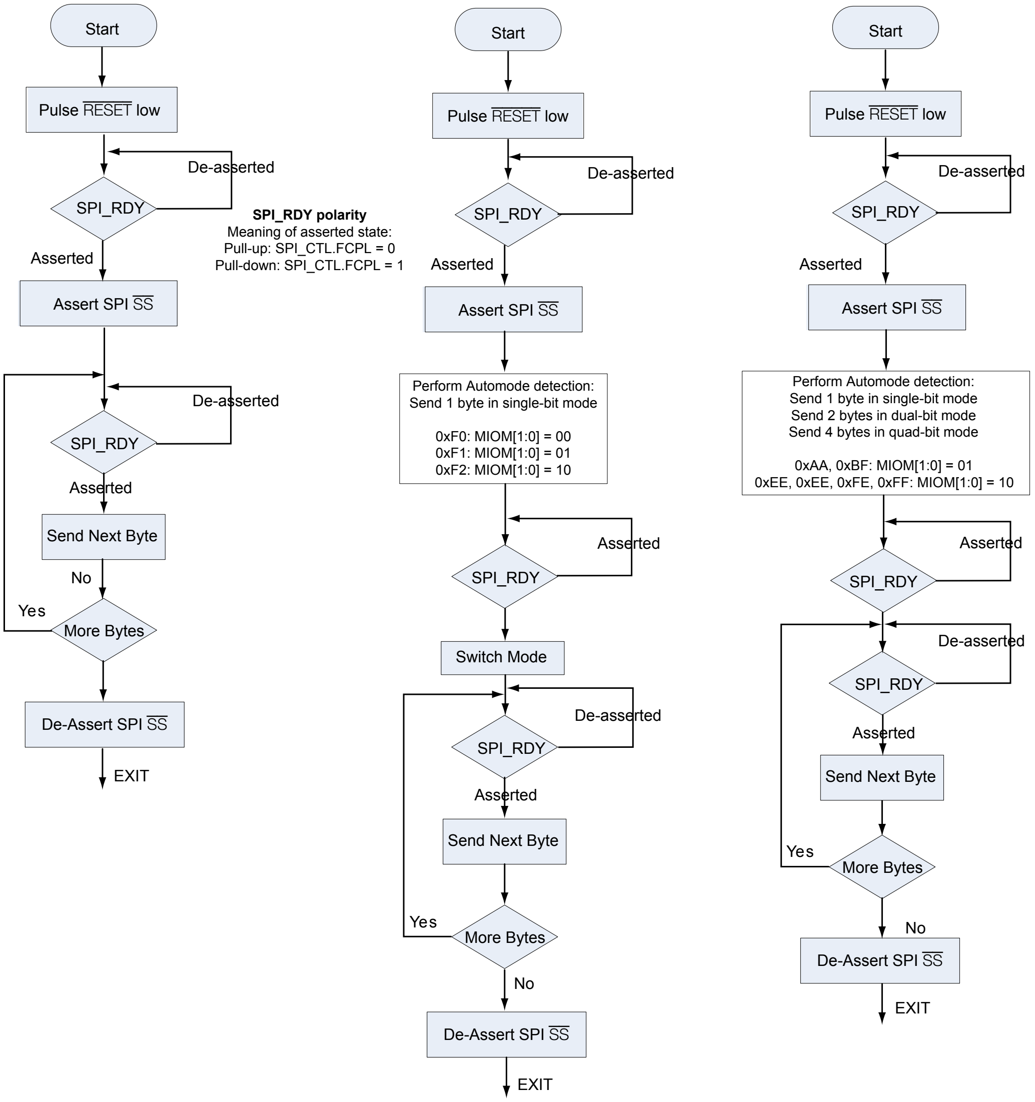

Start
Pulse RESET low
De-asserted
SPI_RDY
Asserted
SPI_RDY polarity
Meaning of asserted state:
Pull-up: SPI_CTL.FCPL = 0
Pull-down: SPI_CTL.FCPL = 1
Assert SPI SS
De-asserted
SPI_RDY
Asserted
Send Next Byte
No
Yes
More Bytes
De-Assert SPI SS
EXIT
Start
Pulse RESET low
De-asserted
SPI_RDY
Asserted
Assert SPI SS
Perform Automode detection:
Send 1 byte in single-bit mode
0xF0: MIOM[1:0] = 00
0xF1: MIOM[1:0] = 01
0xF2: MIOM[1:0] = 10
Asserted
SPI_RDY
Switch Mode
De-asserted
SPI_RDY
Asserted
Send Next Byte
Yes
More Bytes
No
De-Assert SPI SS
EXIT
Start
Pulse RESET low
De-asserted
SPI_RDY
Asserted
Assert SPI SS
Perform Automode detection:
Send 1 byte in single-bit mode
Send 2 bytes in dual-bit mode
Send 4 bytes in quad-bit mode
0xAA, 0xBF: MIOM[1:0] = 01
0xEE, 0xEE, 0xFE, 0xFF: MIOM[1:0] = 10
Asserted
SPI_RDY
De-asserted
SPI_RDY
Asserted
Send Next Byte
Yes
More Bytes
No
De-Assert SPI SS
EXIT

Figure 53-6: SPI Program Flow on the Host Side

Run-time API

The SPI Slave Boot mode can be called through the Boot Routine API function at run-time. Initiating a boot through the run-time API allows for extra customization such as disabling automatic device configuration or specifying a different SPI device other than the default.

When `ROM_BCMD_NOCFG` flag is specified, it is necessary to program pin multiplexing and other SPI configuration as required, while keeping the `SPI_CTL.EN` bit cleared.

The `ROM_BCMD_NOAUTO` flag can suppress auto mode detection. In that case, the desired configuration must be passed through the `ROM_BCMD_SPIS_BCODE` bit field, even if the `ROM_BCMD_NOCFG` flag is set.

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 53-18: SPI Slave Boot `command` Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16	ROM_BCMD_SPIS_BCODE	Boot Mode Specific BCODE. Specifies the boot mode-specific code that can further customize and control the boot process.	
		00xxb	Single bit SPI bus
		01xxb	Dual SPI bus
		10xxb	Quad SPI bus
		11xxb	Reserved
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2	SPI2
		0x3 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	

Table 53-18: SPI Slave Boot `command` Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

Link Port Slave Boot Mode

This section describes booting from the link port with the processor as a slave.

Link port boot is a slave boot mode in which the processor receives boot data from an external link port master through link port 0. The link port is configured for receive mode and all transfers from the link port to memory are performed under the control of DMA. The maximum supported operating frequency of the link port is 66 MHz for which the master boot source is responsible for deriving the clock frequency. The link port receiver operates at an asynchronous frequency up to the maximum supported operating frequency.

The link port protocol supports a way to generate link port transmit and receive service requests. The transmit service request is generated on the processor to transmit data when the transmitter is disabled. The receiver drives the `LACKx` signal high to initiate this activity. The receive service request is generated on a receiver when it is disabled. The transmitter drives the `LCLKx` signal high to initiate this activity.

Because the transmitter and receivers can be enabled at different times, external pull-down resistors are required on both the `LCLKx` and `LACKx` signals to eliminate any false service request assertions.

The link port slave boot mode initialization phase waits for the receive service request before passing control back to the main kernel. Once this initial receive service request has been detected, the receiving link port is enabled and the boot process completes. The receiving link port is not disabled again until after boot is complete. Once the link port is enabled, the receive DMA channel controls all transfers. The load function for the link port receive boot mode can then point to the peripheral DMA routine of the main kernel in a similar way to the SPI slave boot mode.

Run-time API

The LINKPORT slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both device auto-detection and device configuration can be disabled, and a device other than the default LINKPORT0 can be specified.

If `ROM_BCMD_NOCFG` flag is specified, it is the programs responsibility to configure pin multiplexing as required.

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 53-19: LINKPORT Slave Boot `command` Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the LINKPORT device to use.	
		0x0	LP0
		0x1	LP1
		0x2 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.	
		0x4	LINKPORT

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

UART Slave Boot Mode

When using UART slave mode boot, the processor receives boot data from a UART host device connected to the UART interface. The device connected to UART0 is initially detected using an autobaud detection sequence. After finishing the UART slave boot process, all control and status registers of the used resources are restored.

Further customization, such as disabling autobaud detection, and changing the device, use the boot routine API.

During the boot operation, the host device usually relies on the `RTS` output of the UART device. At boot time, the processor does not evaluate RTS signals driven by host. Since the RTS is in a high impedance state when the processor is in reset, or while executing a pre-boot, an external pull-up resistor to `VDDEXT` is recommended. The ***Connection Between Host and Processor*** figure shows the interconnection required for booting. The figure does not show physical line drivers and level shifters that are typically required to meet the individual UART-compatible standards.

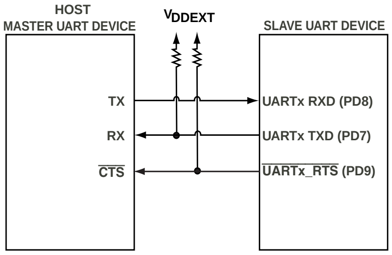


Figure 53-7: Connection Between Host and Processor

When the UART is enabled, the RTS goes immediately low, encouraging the host to send the first boot stream data as shown in the *Host Relying on RTS* figure. For half-duplex UART connections, the host must avoid this action. The host must wait until it has received the 4 bytes from the slave processor before sending any data.

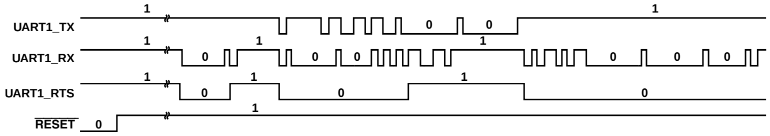


Figure 53-8: Host Relying on RTS

When the boot kernel is processing fill or Initcode blocks, it can require extra processing time and must delay the host from sending more data. This request is signaled using the RTS output.

The *Host Relying on RTS* figure shows RTS timing when an extended Initcode routine executes. Since code execution is distracting from the data loading, the host device must be prevented from sending more data. The timing of the RTS depends on the state of the `RFRT` bit in the UART control register (`UART_CTL`). This bit is cleared during UART slave boot mode when RTS is de-asserted, the UART receive FIFO contains 4 or more data words, and another start bit is detected.

NOTE: Secure Boot Stream Padding

For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

Autobaud Detection

The kernel supports autobaud detection using the '@' character as data. The host is expected to have its clock set to a rate supported in the UART.

To determine the bit rate when performing autobaud detection, use the following steps:

1. The boot kernel expects an '@' character (0x40, eight bits data, one start bit, one stop bit, no parity bit) on the UART RXD input.
2. The `EDBO` and `UART_CLK` register is cleared.
3. The boot kernel acknowledges, and the host then downloads the boot stream. The acknowledgment consists of 4 bytes: 0xBF, `UART_CLK` [15:8], `UART_CLK` [7:0], 0x00.

4. The host is requested to not send further bytes until it has received the complete acknowledge string.
5. Once the 0x00 byte is received, the host can send the entire boot stream.

The host knows the total byte count of the boot stream, but it is not required to know the content of the boot stream.

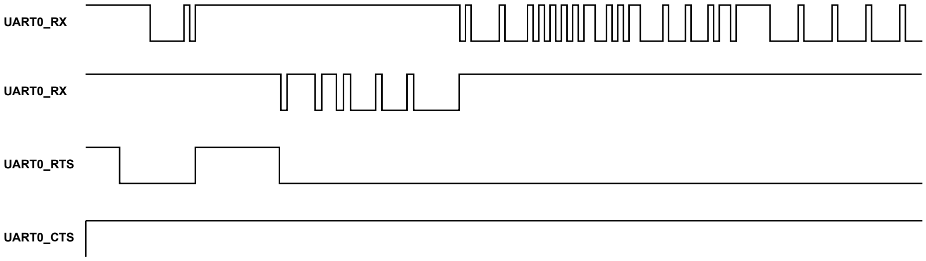


Figure 53-9: UART Autobaud Detection Waveform

The ***UART Autobaud Detection Waveform*** figure provides timing information for UART booting. After the bit rate is known, the UART is enabled and the kernel transmits the 4 acknowledge bytes.

Run-time API

The UART slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both autobaud detection and device configuration can be disabled, and a device other than the default UART0 can be specified.

If `ROM_BCMD_NOCFG` flag is specified, it is the programs responsibility to configure pin multiplexing as required.

Autobaud detection can be suppressed using the `ROM_BCMD_NOAUTO` flag. In this case, the desired configuration can be passed through the `ROM_BCMD_UART_CLK` bit field. If the `ROM_BCMD_UART_CLK` bit field is zero, `UART_CLK` is evaluated. If a value of 0xFFFF was present, the default error routine of the boot kernel is called and the booting process is aborted. Otherwise, the value in `UART_CLK` remains untouched.

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 53-20: UART Slave Boot `command` Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration
31:16	ROM_BCMD_ UART_CLK	UART Clock Divider. When set to zero this field is ignored.
15	ROM_BCMD_ UART_EDBO	UART Clock Divider Mode When set enables EDBO functionality.

Table 53-20: UART Slave Boot `command` Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8	ROM_BCMD_ DEVENUM	Device enumeration. Specifies the UART device to use.	
		0x0	UART0
		0x1	UART1
		0x2	UART2
		0x3 - 0xF	Reserved
6	ROM_BCMD_ NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_ NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x3	UART

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

Boot Loader Stream

A loader stream is a set of formatted blocks containing instructions for the boot kernel, as well as the application and data for loading to the chip. This section details the different aspects of the stream, its blocks, some common use cases, and the ROM functionality.

Each block begins with a block header which contains attributes of the block as well as flags to control its processing by the boot ROM. On power-up or reset, the processor begins executing the on-chip boot ROM. The boot stream is either read from memory or received from a peripheral, depending on the boot mode specified. Each block in the boot stream instructs the boot kernel to perform an action, most commonly to load data to a specified location. For a complete list of actions, refer to Block Types. Some common actions include:

- running code that initializes a peripheral
- processing data then loading it to a location

As the ***Project Flow*** figure illustrates, a utility is required to process the resulting output from the tool chain to create a valid boot stream. This utility can be in the form of a standalone application or script that parses an application image file, elf output file, or text-based file such as Intel hex. It creates a valid boot stream. A flash programmer utility can format a boot stream internally.

The elfloader utility parses the application data, and creates a set of blocks representing the segmented application. When processing an actual image that must be loaded to a single contiguous memory block, this representation can contain as little as a single header block. The output of the standalone utility is stored in a loader file (.ldr). The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral such as the UART during boot time

A loader stream must always begin with a first block and end with a final block. The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral during boot time

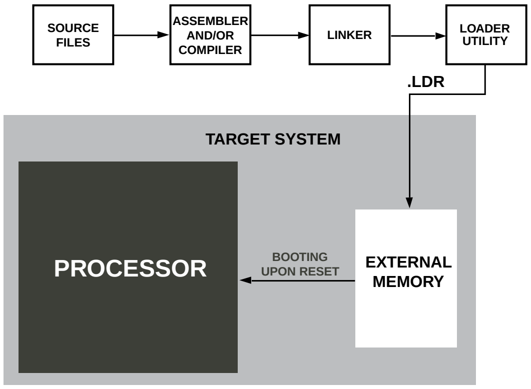


Figure 53-10: Project Flow

The ***Booting Process*** figure shows the parallel or serial boot stream contained in a flash memory device. In host boot scenarios, the non-volatile memory usually connects to the host processor rather than directly to the processor. After reset, the on-chip boot kernel reads and parses the headers and the loader stream is processed block-by-block. Finally, payload data is copied to destination addresses, either in on-chip L1 and L2 memory, or off-chip to SDRAM or SRAM.

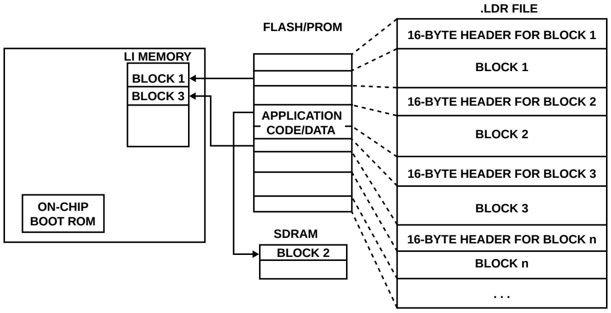

.LDR FILE
FLASH/PROM
16-BYTE HEADER FOR BLOCK 1
BLOCK 1
16-BYTE HEADER FOR BLOCK 2
BLOCK 2
16-BYTE HEADER FOR BLOCK 3
BLOCK 3
16-BYTE HEADER FOR BLOCK n
BLOCK n
...
LI MEMORY
BLOCK 1
BLOCK 3
APPLICATION CODE/DATA
ON-CHIP BOOT ROM
SDRAM
BLOCK 2

Figure 53-11: Booting Process

In some cases (for example, secure boot or when the `BFLAG_INDIRECT` flag for any block is set), the boot kernel uses another memory block for intermediate data storage. In order to preserve the security of the device processors will not allow these storage regions to be initialized with boot data. The boot stream is loaded to the intermediate storage then processed by the kernel and loaded to the final destination. Thge final destination cannot be the intermediate storage location otherwise the boot process will terminate in error.

Block Header

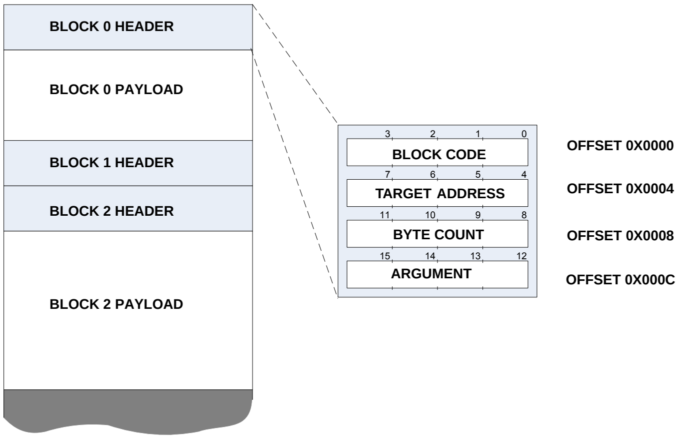

BLOCK 0 HEADER
BLOCK 0 PAYLOAD
BLOCK 1 HEADER
BLOCK 2 HEADER
BLOCK 2 PAYLOAD
3 2 1 0
BLOCK CODE
7 6 5 4
TARGET ADDRESS
11 10 9 8
BYTE COUNT
15 14 13 12
ARGUMENT
OFFSET 0X0000
OFFSET 0X0004
OFFSET 0X0008
OFFSET 0X000C

Figure 53-12: Loader Stream Block Structure

A boot stream consists of multiple boot blocks as shown in the figure. A 16-byte block header begins every block. The 16 bytes are functionally grouped into four 32-bit words:

- the block code
- the target address
- the byte count
- the argument field

This section describes the fields in general. The uses can vary depending on the particular block type and boot mode. Refer to the block type descriptions and boot modes for further information.

Block Code

Table 53-21: Block Code flags

Bit	Name	Description
0–3	`BCODE`	Specific to boot modes (see Boot Modes)
4	`BFLAG_SAVE`	Intended to allow for a user application to mark blocks for saving the memory of this block to off-chip memory in case of power failure . The on-chip boot kernel does not use this flag.
5	`BFLAG_AUX`	When set indicates that the byte address space translation for SHARC core boot blocks requires translation to the 48-bit PM address space. When cleared translation to the 32-bit address space is performed.
6	reserved	
8	`BFLAG_FILL`	Fill the target location with a specified 32-bit value
10	`BFLAG_CALLBACK`	Calls the previously registered callback function
11	`BFLAG_INIT`	Calls function at target address. If the block contains a payload, the payload is loaded prior to the call.
12	`BFLAG_IGNORE`	Block payload is ignored
13	`BFLAG_INDIRECT`	Boots the payload to the intermediate storage location
14	`BFLAG_FIRST`	Indicates the block to be the beginning of a new application
15	`BFLAG_FINAL`	Indicates the last block of a loader stream. Booting will complete after processing the block. This flag does not denote the end of an application in a Multi-Application Boot Streams boot stream.
16–23	`HDRCHK`	A simple 8-bit XOR checksum of the other 24 bits in the boot block header.
24–31	`HDRSIGN`	0xAD, 0xAC or 0xAB. Indicates if the boot block is intended for core 0, core 1 or core 2 respectively.

TARGET_ADDRESS

The `TARGET_ADDRESS` holds the applicable address for the block, (where the code or data is loaded). However, the interpretation of the field differs depending on what specific flags are set in the block code. Refer to the documentation for each block type for details.

The following attributes must be true:

- The target address must be divisible by 4, as the boot kernel uses 32-bit DMA for certain operations.
- The target address must point to valid on-chip or off-chip memory locations.

BYTE_COUNT

The byte count must be divisible by 4, and can also be zero. This 32-bit field generally holds the size of the block. In some cases, it has a different use (such as when `BFLAG_FILL` is set). See the Block Types section for information on the variations.

ARGUMENT

The 32-bit field is a user-variable for most block types. The `Initcode` or the `callback` routine can access this value and use it for optional instructions.

The different block types use the `ARGUMENT` field in various ways. See the Block Types descriptions for further information.

Block Types

A loader stream is a set of linked blocks and each block is responsible for performing a certain function dependent on the block type. The flags in the block header define a block type. Operations include functions such as loading data, filling a memory region with data, and instructing the kernel to stop processing. This section describes each block type and its usage within a boot loader stream.

Normal Block

The primary function of a block is to load data into a specified location of memory. A normal block instructs the boot kernel to load the data contained in its payload to the location specified in the `TARGET_ADDRESS` field. The `BYTE_COUNT` defines the size of the payload. Once the correct amount of data has been loaded, the kernel moves on to process the next block in the stream.

Table 53-22: Flags

Flag	Required Value	Init
`TARGET_ADDRESS`	Y	Address where payload is loaded (must be valid)
`BYTE_COUNT`	Y	Size of block in bytes

First Block

A first block indicates the start of a boot stream and is always needed at the beginning of the boot stream. When a loader stream contains Multi-Application Boot Streams, a first block occurring within the loader stream indicates the beginning of a new application.

When the kernel processes the first block in a loader stream, the boot kernel uses the `TARGET_ADDRESS` to determine the application entry location. For more details, refer to Boot Termination and Application Execution .

NOTE: A first block cannot be combined with a fill block.

Table 53-23: Flags

Flag	Required Value	Init
`BFLAG_FIRST`	Y	1

Table 53-23: Flags (Continued)

Flag	Required Value	Init
`ARGUMENT`	Y	Offset to the next application, or first address following loader stream. Commonly referred to as the `NEXTDXE` field.
`TARGET_ADDRESS`	Y	When the block is the first block in a loader stream, also defines the start address for the application. If the block is not the first in a loader stream, use the target address as in normal operation.

Final Block

The final block marks the last block in a boot stream. After processing a final block, the boot kernel jumps to the start address of the application. For more information on the definition of the start address, refer to Boot Termination and Application Execution .

There is further customization to the kernel behavior available. For example, the kernel can be instructed to return from the boot routine rather than jump to the application using initialization codes or the `adi_rom_Boot()` API.

Before the boot kernel passes program control to the application, it does some housekeeping. Most of the registers in use are set to their default state. However, some register values can differ depending on the boot mode. See Boot Modes for more information.

Table 53-24: Flags

Flag	Required Value	Init
`BFLAG_FINAL`	Y	1

Indirect Block

An indirect block is first loaded to a storage location before being copied to the destination. The following situations motivate this functionality:

- Some boot modes do not use DMA from the boot peripheral. The core is not always able to access some memory locations directly or efficiently. An intermediate load to a different location improves overall efficiency.
- In some booting scenarios, the data in the payload must be operated-on or analyzed before it is fully loaded (such as decryption or checksum calculation). By using an intermediate location, such scenarios are simplified and can be more efficient when loading to off-chip memories (see Callback Block).

In some cases, a boot block does not fit into temporary storage memory. Having a larger buffer can improve boot performance. If an entire block cannot fit into the buffer, it is processed in pieces. Initialization code or callback functions can alter the temporary buffer region, including its location and size, by modifying `ADI_ROM_BOOT_CONFIG::pTempBuffer` and `ADI_ROM_BOOT_CONFIG::dTempByteCount` variables in the `ADI_ROM_BOOT_CONFIG` structure.

Table 53-25: Flags

Flag	Required Value	Init
`BFLAG_INDIRECT`	Y	1
`BFLAG_CALLBACK`	N	Defines a callback function to operate on intermediate data. These 2 flags are often used together.

Ignore Block

An ignore block is a block that is (in most cases) ignored by the loader stream. Ignore blocks are useful when it is not possible to pass information in another block header. For example, if the first block contains data such as a firmware revision rather than application code, then `BFLAG_IGNORE` can be set with the correct application start address. This step ensures that the loader stream uses the correct start address. Since this block has no other function, identify it as an ignore block. Then, the kernel does not attempt to process any payload.

Ignore blocks result in the clearing of the following flags, disabling the corresponding blocks from being processed if set along with `BFLAG_IGNORE`:

- `BFLAG_INIT`
- `BFLAG_CALLBACK`
- `BFLAG_FINAL`
- `BFLAG_AUX`

NOTE: When `BFLAG_IGNORE` is set along with `BFLAG_FIRST`, only the payload associated with the first block is ignored. The application entry point retrieved from the first block is always processed.

Table 53-26: Flags

Flag	Required Value	Init
`BFLAG_IGNORE`	Y	1
`BYTE_COUNT`	Y	Size of block to ignore; can be zero

Fill Block

A fill block instructs the boot kernel to perform a 32-bit memory fill of the memory region. Fill blocks help minimize the size of a boot stream when an application contains large arrays of data that need to be initialized upon startup. Zero fill is the most common fill block type. However, any 32-bit fill value is supported.

Table 53-27: Flags

Flag	Required Value	Init
`BFLAG_FILL`	Y	1
`TARGET_ADDRESS`	Y	Address where payload is loaded (must be valid)
`BYTE_COUNT`	Y	Size of block in bytes (must be multiple of 4)
`ARGUMENT`	Y	32-bit fill value

AUX Block

The AUX block is used to indicate whether addresses not within the system wide byte address space for the processor are to be converted using the regular 32-bit address space translation or 48-bit program memory space conversion. When this flag is cleared, 32-bit address conversion is used. When set, 48-bit address conversion is used to obtain the byte address to load the payload.

This flag is only required to be used when the utility used for generating the boot stream does not automatically convert payload addresses to the byte address space. When the boot kernel detects that an address in the block header is already targeted towards the byte address space, this flag has no purpose.

It should also be noted that the converted address is only used for the loading of the payload via the DMA channels. If a block such a callback block or an init block is required to execute from SHARC L1 48-bit PM space, the address is converted for the load operation. However, the call is performed to the original 48-bit PM address.

Init Block

An initialization block instructs the boot kernel to do a function call to the target address after the entire block has loaded. The function called is referred to as the *initialization code (Initcode) routine*. Refer to the API reference `initcode()` for full details.

If the Initcode routine has been previously loaded, the block can declare a zero-size and have no payload.

Initcode routines can be used to speed up and customize booting mechanisms exposed by the boot kernel. Traditionally, an Initcode routine is used to setup system PLL, bit rates, wait states, and the external memory controllers. If executed early in the boot process, the boot time can be significantly reduced.

Initcode routines are required to follow the C language calling conventions. The expected C prototype is:

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

NOTE: When programming in assembly, use a return from subroutine instruction for returns.

The structure provided to the Initcode routine by the boot kernel contains various information about the block being processed. It includes header information, locations of temporary block data (for indirect blocks), target address, and byte count. See `ADI_ROM_BOOT_CONFIG` for a full list and details on the provided data.

In the simplest case, an Initcode routine consists of only a single block in which the `BFLAG_INIT` flag is set. For larger routines, a sequence of blocks can incrementally load the routine, and only the last block sets the `BFLAG_INIT` flag. In the latter case, the last block has no payload attached, and simply instructs the boot kernel to issue a call to subroutine instruction.

An Initcode routine can be overwritten by a successive block when it is no longer needed. Otherwise, the routine can be called at multiple points during the boot process, and even remain in memory after the application completes booting.

NOTE: The following list provides requirements for Initcode that is written in C or C++.

- Ensure that the Initcode routine does not contain calls to the run-time libraries
- Do not assume that parts of the run-time environment, such as the heap, are fully functional
- Ensure that all run-time components are loaded and initialized before the routine executes

The `loader` utility and tool set provide mechanisms to aid in implementing initialization codes and organizing them properly within the boot loader stream. A special project type is provided to allow the creation of Initcode routines as separate projects. Options are available to assign particular pieces of the application to be Initcode routines. For details and more information on the utility, see to the *Loader and Utilities manual*.

Table 53-28: Flags

Flag	Required Value	Init
`BFLAG_INIT`	Y	1
`TARGET_ADDRESS`	Y	Location to load payload data. Call to subroutine issued to the same location.
`ARGUMENT`	N	Can be used to supply block specific arguments
`BYTE_COUNT`	Y	Size of payload; can be zero

NOTE: Init blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenrio, initcode routines are not supported. The secure boot authentication process is performed at the end of the boot process. Execution of any user software prior to the authentication process would violate the secure boot requirements.

Callback Block

A callback block instructs the boot kernel to call a pre-registered function upon completion of loading the payload of the block. The purpose of a callback routine is to apply standard processing to the block payload. The callback routine is registered through an Initcode routine prior to loading a block using the routine. Typically, callback routines contain checksum, decryption, decompression or hash algorithms. Please refer to `callback()` API reference.

To register a callback, create an Init Block whose Initcode modifies `ADI_ROM_BOOT_CONFIG::pCallBackFunction` with the address of the callback function to execute. A callback function must be registered prior to processing a callback block.

Since callback routines require access to the payload data of the boot blocks, load the block data before processing. Often an Indirect Block block is used in combination with a callback block. Indirect blocks in combination with callback blocks can allow for post processing of the loaded data before it is then transferred to the final destination.

Callback routines are expected to meet the C language calling conventions. The prototype is as follows:

```
ROM_BOOT_RESULT callback(
                         ADI_ROM_BOOT_CONFIG * pBootConfig,
                         ADI_ROM_BOOT_BUFFER * pBuffer,
                         uint32_t nFlags
                        )
```

The `pBootConfig` argument contains a pointer to the `ADI_ROM_BOOT_CONFIG` information, and `pBuffer` provides access to the address and size of the block (can vary when using indirect). The `nFlags` parameter is specifically used when `BFLAG_INDIRECT` is also used. `CBFLAG_DIRECT` flag indicates that the `BFLAG_INDIRECT` bit is not active and so that the program only calls the callback routine once per block. When the `CBFLAG_DIRECT` is set, `CBFLAG_FIRST` and `CBFLAG_FINAL` are also set.

NOTE: Callback blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenario callback routines are not supported as the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process would violate the secure boot requirements.

Callback Block Used in Conjunction with Indirect Block

When a block using a callback routine is also loaded indirectly, there are slight behavior differences. The procedure for loading is:

1. Load data into the temporary buffer defined by `ADI_ROM_BOOT_CONFIG::pTempBuffer`.
2. Issue a call to `ADI_ROM_BOOT_CONFIG::pCallBackFunction`.
3. After the callback routine returns, if the return value is zero, the memory DMA copies data to the destination.

When a block does not fit entirely into the temporary buffer, loading is performed similar to indirect blocks. The software calls the callback function after each chunk is loaded into the temporary storage. The `nFlags` parameter gives information on the specific iteration.

When a block does not fit entirely into the temporary storage area, the `nFlags` tells the callback routine whether it is invoked for the first time (`CBFLAG_FIRST`) or called the last time (`CBFLAG_FINAL`) for a specific block.

When the software invokes DMA to copy the data, it relies on the supplied `pBuffer` data, not the `ADI_ROM_BOOT_CONFIG::pTempBuffer` and `ADI_ROM_BOOT_CONFIG::dTempByteCount` members of the boot structure. The callback routine can control the source of the memory DMA by altering the content of the `pBuffer` structure. This alteration can be necessary when the callback routine performs data manipulation such as decompression.

When the software uses an indirect block, the return value of the callback routine determines whether the DMA transfer occurs. If the value is non-zero, then the transfer does not occur.

Table 53-29: Flags

Flag	Required Value	Init
`BFLAG_CALLBACK`	Y	1

NOTE: Callback blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenrio callback routines are not supported as the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process would violate the secure boot requirements.

Quick Boot Block

There are some booting scenarios in which not all memories are required to be reinitialized during the boot process and as a result the boot kernel supports conditional processing of boot blocks in specific circumstances. Dynamic RAM is also not always impacted if it was put into a self-refresh mode before the processor powered down.

The processing of a quick boot block is determined by the state of the `BFLAG_WAKEUP`, `BFLAG_QUICKBOOT` and `BFLAG_IGNORE` flags.

Table 53-30: Quick Boot Block Processing

BFLAG_WAKEUP	**BFLAG_QUICKBOOT**	**BFLAG_IGNORE**	Block Processed
0	0	0	Yes
0	0	1	No
0	1	0	Yes
0	1	1	No
1	0	0	Yes
1	0	1	No
1	1	0	No
1	1	1	Yes

The `BFLAG_WAKEUP` flag is applied to the entire boot process when calling the boot kernel. Refer to `adi_rom_Boot()` for further details on the global flags available that can be applied to the boot process.

When the boot process is triggered via a hard reset or a software triggered system reset event the boot kernel is always called with `BFLAG_WAKEUP` cleared. Processors not supporting the wakeup from hibernate feature would generally not have this quickboot feature available for typical boot events. However, when using `adi_rom_Boot()` or `adi_rom_BootKernel()` from the user application to boot the processor, the quick-boot feature and selective processing of boot blocks is fully available .

NOTE: When `BFLAG_WAKEUP` is set and a block also has `BFLAG_QUICKBOOT` set, `BFLAG_IGNORE` flag is toggled within the boot kernel. This event occurs before the processing of an Ignore Block and as such also has an influence on the processing of blocks that are disabled as a result of `BFLAG_IGNORE` being set.

NOTE: If an initcode contains an implementation that requires some operations to be performed for wakeup type events, do not set `BFLAG_QUICKBOOT` in the initblock, disabling it completely. Instead, ensure `BFLAG_QUICKBOOT` is clear and perform any conditional processing based on the state of the `BFLAG_WAKEUP` flag.

Table 53-31: Flags

Flag	Required Value	Init
`BFLAG_QUICKBOOT`	Y	1

Save Block

A save block is intended to mark blocks in a boot stream that are to be saved to off-chip memory. The on-chip boot kernel does not use this flag. A user application can process the boot steam to find address of regions of memory that are to be saved off to external memory. Upon a reboot the user application may then restore the previously saved contents. It provides a means of doing a context restore after a reboot.

Table 53-32: Flags

Flag	Required Value	Init
`BFLAG_SAVE`	Y	1

Single-Block Boot Streams

The simplest boot stream consists of a single block header and one contiguous block of instructions and possibly data. When the appropriate flags are set in the block header, the kernel loads the block to the target address, terminates the boot process, and begins executing from the entry address of the application.

The ***Initial Header for Single-Block Stream*** table shows an example of a single-block boot stream header with settings that can be loaded using any boot mode. The `BFLAG_FIRST` and `BFLAG_FINAL` flags are both set at the same time. The desired location and size of the application determines the target address and byte count.

When using single block boot streams on products with multiple cores, the boot stream must be targeted towards the primary booting core that manages the boot process. If core 0 is the primary booting core, the boot stream must contain code that is intended for execution by that core. It is possible to boot an single block boot stream when using the API to load an application to the non booting core. In this case the `BFLAG_RETURN` flag must be set so the boot process returns to user application on the core mastering the boot process.

Table 53-33: Initial Header for Single-Block Stream

Field	Description of Value
BLOCK_CODE	0xAD000000|XORSUM|`BFLAG_FINAL`|`BFLAG_FIRST`
TARGET_ADDRESS	Start address of block
BYTE_COUNT	Number of bytes in the block
ARGUMENT	Functions as next-application pointer in multi-application boot streams.

Direct Code Execution

Applications can avoid long booting times and execute code from flash or SDRAM memory with minimal processing by the boot kernel. This feature is called direct code execution.

An initial boot block header is required for the processor to fetch and execute program code from the boot device as early as possible. The block uses safety mechanisms of the block to avoid unpredictable processor behavior when boot memory is not yet programmed with valid data. Safety mechanisms include the header signature and the byte-wise XOR checksum. Rather than blindly executing code, the boot kernel first executes the pre-boot routine for system customization. It then loads the first block header and checks it for consistency. If the block header is corrupt, the boot kernel calls the error handler and does not start code execution.

If the initial block header check is good, the boot kernel interrogates the block flags. If the block has the `BFLAG_FINAL` flag set, the boot kernel terminates and executes the sequence as described in the Boot Termination and Application Execution section. To cause the boot kernel to customize the starting address in advance, the first block must also have the `BFLAG_FIRST` flag set. The target address field is then saved as the application start address.

When processing direct code execution blocks, the block instructs the processor executing the boot code to execute from the address specified. It is not possible to have core 0 to boot the block and have it instruct core 1 to immediately start execution from the address provided.

Table 53-34: Example Direct Code Execution Header

Field	Value	Comments
BLOCK_CODE	0xAD7BD006	0xAD000000|XORSUM|`BFLAG_FINAL`|`BFLAG_FIRST`|`BFLAG_IGNORE`|BCODE
TARGET_ADDRESS	0x20000020	Start address of application code. Provided as an example this would be dependent upon the start address required for a given product.
BYTE_COUNT	0x00000010	Ignores 16 bytes to provide space for control data such as version code and build data. This field is optional and can be zero. The payload is ignored due to the `BFLAG_IGNORE` flag being set.
ARGUMENT	0x00000010	Functions as next-application pointer in multi-application boot streams.

Multi-Application Boot Streams

This section describes loader streams that contain multiple applications.

A boot stream is typically generated from an application file. It is therefore common to refer to loader streams with more than one application (multi-application) booting. A loader utility often accepts multiple application files as input parameters and generates a contiguous boot image. The subsequent applications are appended to the first.

The loader utility must also update the argument field of all `BFLAG_FIRST` blocks. The argument field of a `BFLAG_FINAL` block is called the *next-application* pointer.

The next-application pointer of the first application boot stream points relatively to the start address of the second application boot stream. A multi-application boot image can be seen as a linked list of boot streams. The next-application pointer of the last application boot stream points relatively to the next free address. The ***Multi-Application Boot Stream Example*** figure illustrates an example.

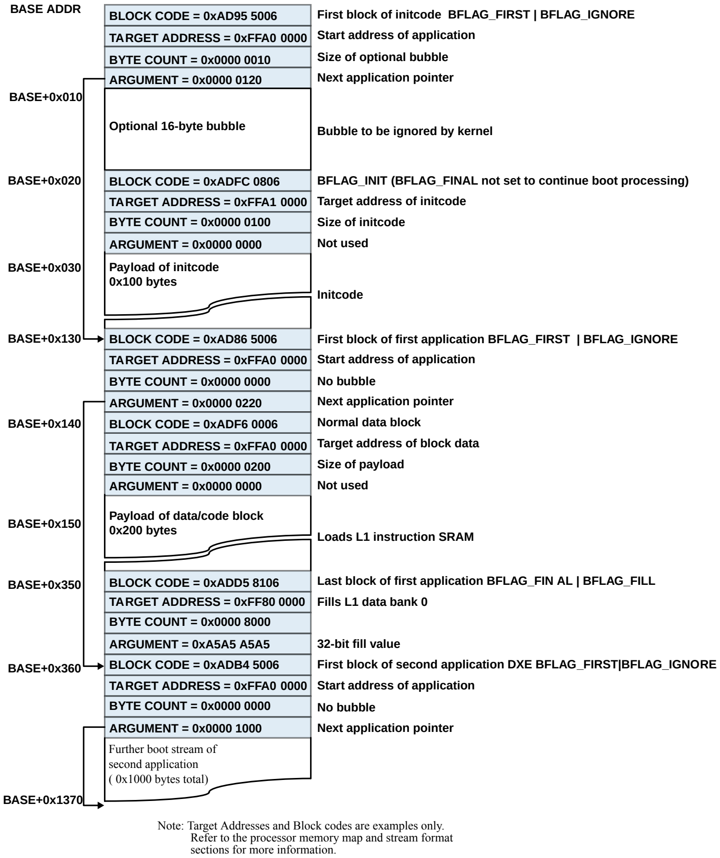

BASE ADDR
BLOCK CODE = 0xAD95 5006
First block of initcode BFLAG_FIRST | BFLAG_IGNORE
TARGET ADDRESS = 0xFFA0 0000
Start address of application
BYTE COUNT = 0x0000 0010
Size of optional bubble
ARGUMENT = 0x0000 0120
Next application pointer
BASE+0x010
Optional 16-byte bubble
Bubble to be ignored by kernel
BASE+0x020
BLOCK CODE = 0xADFC 0806
BFLAG_INIT (BFLAG_FINAL not set to continue boot processing)
TARGET ADDRESS = 0xFFA1 0000
Target address of initcode
BYTE COUNT = 0x0000 0100
Size of initcode
ARGUMENT = 0x0000 0000
Not used
BASE+0x030
Payload of initcode
0x100 bytes
Initcode
BASE+0x130
BLOCK CODE = 0xAD86 5006
First block of first application BFLAG_FIRST | BFLAG_IGNORE
TARGET ADDRESS = 0xFFA0 0000
Start address of application
BYTE COUNT = 0x0000 0000
No bubble
ARGUMENT = 0x0000 0220
Next application pointer
BASE+0x140
BLOCK CODE = 0xADF6 0006
Normal data block
TARGET ADDRESS = 0xFFA0 0000
Target address of block data
BYTE COUNT = 0x0000 0200
Size of payload
ARGUMENT = 0x0000 0000
Not used
BASE+0x150
Payload of data/code block
0x200 bytes
Loads L1 instruction SRAM
BASE+0x350
BLOCK CODE = 0xADD5 8106
Last block of first application BFLAG_FIN AL | BFLAG_FILL
TARGET ADDRESS = 0xFF80 0000
Fills L1 data bank 0
BYTE COUNT = 0x0000 8000
ARGUMENT = 0xA5A5 A5A5
32-bit fill value
BASE+0x360
BLOCK CODE = 0xADB4 5006
First block of second application DXE BFLAG_FIRST|BFLAG_IGNORE
TARGET ADDRESS = 0xFFA0 0000
Start address of application
BYTE COUNT = 0x0000 0000
No bubble
ARGUMENT = 0x0000 1000
Next application pointer
Further boot stream of
second application
(0x1000 bytes total)
BASE+0x1370
Note: Target Addresses and Block codes are examples only.
Refer to the processor memory map and stream format
sections for more information.

Figure 53-13: Multi-Application Boot Stream Example

The ***Multi-Application Direct Code Execution Example*** figure shows a linked list of initial block headers. The blocks instruct the boot kernel to terminate immediately and to start code execution at the address provided by the target address field of the individual blocks. There is nothing in the boot code that prevents multi-application streams from mixing regular boot streams and direct code execution blocks.

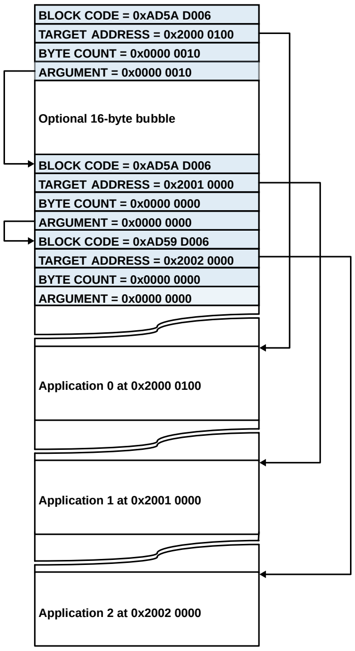

BLOCK CODE = 0xAD5A D006
TARGET ADDRESS = 0x2000 0100
BYTE COUNT = 0x0000 0010
ARGUMENT = 0x0000 0010
Optional 16-byte bubble
BLOCK CODE = 0xAD5A D006
TARGET ADDRESS = 0x2001 0000
BYTE COUNT = 0x0000 0000
ARGUMENT = 0x0000 0000
BLOCK CODE = 0xAD59 D006
TARGET ADDRESS = 0x2002 0000
BYTE COUNT = 0x0000 0000
ARGUMENT = 0x0000 0000
Application 0 at 0x2000 0100
Application 1 at 0x2001 0000
Application 2 at 0x2002 0000

Figure 53-14: Multi-Application Direct Code Execution Example

CRC32 Protection

This section describes the CRC32 protection provisions.

The boot kernel provides mechanisms to allow verification of each blocks payload using a 32-bit CRC. The boot rom contains a function in the rom that can be called as an initcode to register the CRC callback and initialize the CRC peripheral with a user specified polynomial. The utilities provided by the supporting toolchain for the processor allow for generation of CRC32 protected loader streams.

To enable this feature an Init Block must be located in the boot stream with a `TARGET_ADDRESS` that points to the function in the ROM. The `ARGUMENT` field contains the CRC32 checksum polynomial to be used to initialize the CRC lookup table. Once this CRC initcode function in the ROM has been executed CRC verification is enabled for all subsequent blocks except:

- Ignore
- First

NOTE: Due to the fact that the enabling of the CRC functionality is dependent upon the use of an Initblock. The feature is not supported in secure boot situations.

Secure Boot

The secure boot process provides a means of integrating security in the processor boot sequence. A chain of trust is established within the system by ensuring the integrity and authenticity of the boot image. Confidentially support is also supported.

Secure boot increases protection against malicious, unsecured accesses to critical and confidential resources of the processor. The boot stream application code and data must be digitally signed in order to build up a chain of trust in the system. This allows the processor to distinguish between authentic and trusted code from non-authentic and untrusted code.

Secure boot also provides confidentiality support. The digitally signed boot image can be optionally encrypted as well. When loading an encrypted image, the ROM decrypts while loading, then authenticates, before any application code is executed.

Secure boot is an optional feature of the processor and is disabled by default. The feature is enabled using the OTP lock API, and secure boot cannot be disabled after it has been enabled. When security is enabled, developers are not dependent upon on Analog Devices to provision the devices, sign code or provide security certificates. The required tools for signing and encrypting the boot images are provided with the development tools for the processor.

Integrity and Authenticity Protection

Integrity protection is based on the secure hash SHA-2 224 bit algorithm. Authenticity protection is based on the ECDSA algorithm.

ECDSA uses public key cryptography consisting of two keys, a private key and a public key. The public key is stored in OTP memory on the processor so that the secure boot process can verify the authenticity of the signed boot image. Only parties in possession of the private key are able to sign the images.

Confidentiality Protection

Confidentiality protection uses the AES algorithm. Two variants are supported, wrapped and unwrapped.

The wrapped variant utilizes a 128-bit Key Encryption Key (KEK) stored on the processor to decrypt the 128-bit AES decryption key embedded in the secure header. The unwrapped variant stores the AES description key on the processor and utilizes it to decrypt the entire image.

The privacy of the key stored on the device (whether AES or KEK) is paramount to the security of the system. Disclosure of this key compromises security of the entire system.

Anti-Cloning Protection

Anti-cloning protection is based on the confidentiality protection. If each processor uses a unique private key for the confidentiality protection, then cloning between these devices can be prevented. The boot image is incompatible with devices using a different private key for the decryption.

Anti-Rollback Protection

The secure boot process supports anti-rollback protection through a 32-bit counter in the OTP memory. A value of `0x00000000` in the OTP results in anti-rollback being disabled by default. If anti-rollback protection is required, then the user may set the Rollback ID when signing the boot image. Upon successful authentication of the boot image, the secure boot software then updates the counter in the OTP. The software updates the counter if the rollback ID in the boot image is greater than the value currently stored in the OTP counter.

The rollback ID stored in the secure boot image header is integrity-protected preventing altering of the rollback ID.

NOTE: In order to initially enable anti-rollback protection for secure boot operations, a non-zero must be written to the 32-bit counter in the OTP memory. As long as this register filed remains at the default value of zero anti-rollback protections will not be enabled regardless of the rollback ID located in a secure boot stream.

CAUTION: As the rollback ID is implemented in the OTP module, there are a number of restrictions for its use. It is therefore recommended to only use the OTP boot program ROM API to set the counter. Refer to the OTP counters section for information detailing the implementation strategy.

Terminology

ECDSA

Elliptical Curve Digital Signature Algorithm

BLp

Boot Loader plain text, Plaintext Format

BLx

Boot Loader without key, Keyless Format

BLw

Boot Loader wrapped, Wrapped Format

BLe

Boot Loader encrypted, Encrypted Format

SBLS

Secure Boot Loader Stream

SBH

Secure Boot Header

SBCR

Secure Boot Confidentiality Root

AES

Advanced Encryption Standard

Secure Boot Image Signing

All boot images must be digitally signed to create secure boot images. The boot image is processed by the security utilities included with the development tools to sign and optionally encrypt the boot image. The security utilities operate with key-pairs consisting of a private and a public key. The private key is used for signing the images, and the public key is used to validate an image being loaded into the processor.

CAUTION: The private key generated from the singing utility, used for signing images, is never required by the processor for successful secure boot. The private key is only ever required by the signing utility and should be made available only within the system responsible for the image signing process.

The image signing utility provides the following functionality:

- Signing and encrypting of images
- Generation of ECDSA key pairs
- Generation of random encryption keys
- Extraction of the public key from an ECDSA key pair
- Setting Secure Boot Image Attributes

For more information on the use of the signing utility, refer to the Loader and Utilities manual.

Secure Boot Image Types

This section provides an overview of the different image types supported, as well as supporting information on how to use them.

Plaintext Format (BLp)

Provides integrity plus authentication protection of the boot image. The boot image is produced using a 224-bit Elliptical Curve Digital Signature Algorithm (ECDSA) private key. To authenticate the image, program the corresponding public key into the OTP `public_key` field using the OTP boot program API.

SBH	Boot Loader Stream

Wrapped Format (BLw)

Provides the highest level of protection: integrity plus authentication, confidentiality, and anti-cloning protection. The image contains an ECDSA wrapped image encryption key (denoted by [K]) within the secure header. The image data is encrypted with the wrapped key, preventing cloning. An extra key is required to unwrap the header, program this key into OTP `pvt_128key` field using the OTP boot program API.

SBH [K]	Encrypted Boot Loader Stream

Keyless Format (BLx)

Similar to the BLw format except that the image does not contain the key at all. This format provides anti-cloning protection only if the secure key is unique per device. Program the decryption key for the data into OTP `pvt_128key` field using the OTP boot program API.

SBH	Encrypted Boot Loader Stream

Secure Boot Image Format

Secure Boot images provide authenticity and integrity protection during the boot process. A secure boot image is comprised of a secure boot header and an optionally encrypted loader stream.

Signed images consist of the following sections to comprise a complete secure boot image:

- Secure Boot Header
- Image Attributes
- Image Section

The Figure 53-15 Secure Boot Image figure shows that the image attributes are encapsulated within the secure boot header. The image attributes are actually integrity protected along with the image section. The image section contains a standard Boot Loader Stream. Some block types are not allowed as described in Unsupported Boot Stream Blocks.

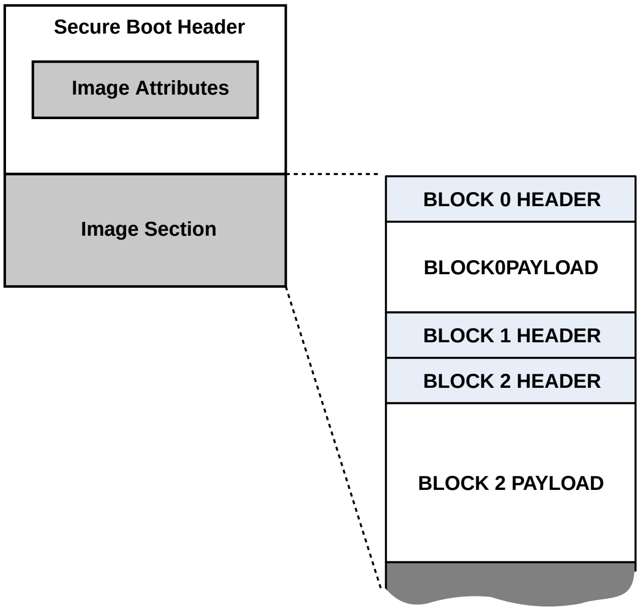


Figure 53-15: Secure Boot Image

Secure Boot Header

Table 53-35: Secure Boot Header

Bytes	Name	Description	Values		
			Keyless Format (BLx)	Wrapped Format (BLw)	Plaintext Format (BLp)
3:0	Type	Format and version of the image. Upper 24 bits is the image format and lower 8 bits is the image version	0x424c7801	0x424c7701	0x424c7001
67:4	Signature	The ECDSA signature of the image	Two 256-bit numbers		
91:68	Key	Confidentiality (only applicable for certain formats)	Reserved	192-bit AES-WRAP data holding a 128-bit AES key	Reserved
107:92	IV	Initialization Vector (only applicable for certain formats)	Reserved	16-byte IV generated during signing process	
111:108	Length	The length of the image section in bytes	Maximum supported byte count 0x10000000 bytes		
175:112	Attributes	Image attributes	Support for up to 8 image attributes		
179:176	Reserved	Reserved	Reserved		

Overview of Secure Boot Processing

The ***Secure Boot Processing*** figure illustrates how the block is processed regarding secure features. The figure does not detail all the block header type handling when processing the 16-byte block headers of the image section. For details on the various block types and their functionality, refer to Block Types. Some image types are decrypted. [Decrypt] indicates that the data is decrypted when applicable to the Secure Boot Image Types at that particular stage.

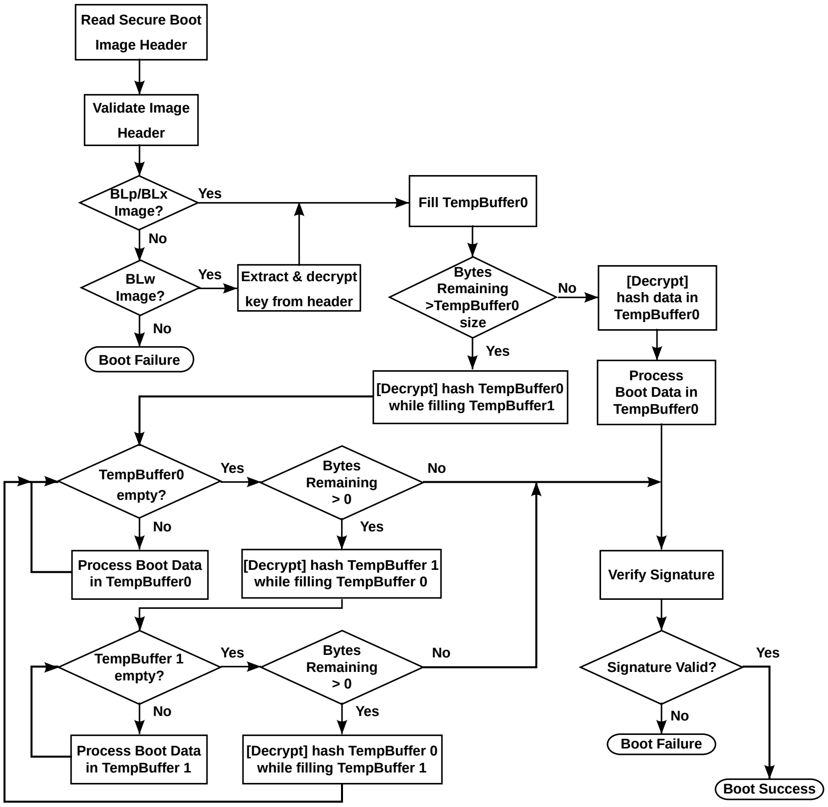

Read Secure Boot Image Header
Validate Image Header
BLp/BLx Image?
Yes
No
BLw Image?
Yes
No
Boot Failure
Extract & decrypt key from header
Fill TempBuffer0
Bytes Remaining >TempBuffer0 size
No
Yes
[Decrypt] hash data in TempBuffer0
Process Boot Data in TempBuffer0
[Decrypt] hash TempBuffer0 while filling TempBuffer1
TempBuffer0 empty?
Yes
No
Bytes Remaining > 0
No
Yes
Process Boot Data in TempBuffer0
[Decrypt] hash TempBuffer 1 while filling TempBuffer 0
Verify Signature
TempBuffer 1 empty?
Yes
No
Bytes Remaining > 0
No
Yes
Signature Valid?
Yes
No
Boot Failure
Process Boot Data in TempBuffer 1
[Decrypt] hash TempBuffer 0 while filling TempBuffer 1
Boot Success

Figure 53-16: Secure Boot Processing

Unsupported Boot Stream Blocks

To ensure the security of the processor, the following block types are not supported in a secure boot image. If the boot kernel finds one of these block types, the boot process terminates.

- Init Block

  Init blocks require a call to user application code prior to the authentication of the boot image, and, therefore, cannot be supported. If customizations or optimizations are necessary to improve the load performance, use a second stage loader style implementation. The first application will contain only the custom code. Issue a call using to boot using the desired device.

- Callback Block

  Callback blocks require a call to a user-defined address prior to the authentication of the boot image, and therefore cannot be supported.

NOTE: Secure boot streams use double buffer Page Mode to optimize the boot process. This functionality allows for the performance of decrypt and hash operations on received data while new data is fetched from the boot source. This host in slave boot mode must ensure that more data is sent after the boot stream to ensure that the temp buffer is filled completely. The size of the secure boot stream minus the size of the secure boot header must be a multiple of the size of the temp buffer. The temp buffer default size is 1024 bytes.

Secure Boot Image Attributes

Secure boot image attributes form part of the secure boot header . The attributes provide more information about the content of the secure boot image.

All image attributes are integrity protected using the same algorithm as the image section. When the image authentication process completes and the image is successfully authenticated, the image attributes are known to be trustworthy.

Attributes are specified as type value pairs with both the type and value being a 32-bit value. The boot code supports the following image attributes.

Table 53-36: Secure Boot Image Attributes

ID	Name	Description	Values	
0x00000000	Unused	Unused attribute	Value must be 0x00000000	
0x00000001	Version	Version of the attribute format	Value must be 0x00000000	
0x00000002	Rollback ID	Current value of the rollback counter	0x00000000	Rollback disabled
			0x00000001 - 0x0000001F	Current firmware revision. Must be greater than or equal to the value retrieved from OTP Refer to Anti-Rollback Protection and the OTP chapter for details on the counter implementation.
0x80000001	NoRestore	Controls whether the boot process restored registers back to default values and clears sensitive security related information from the stack and dedicated structures.	0x1	Do not restore registers and clear security information from the stack and dedicated structures
			other	Restore registers and clear security sensitive information from the stack and dedicated structures
0x80000002	BCODE	Used by the boot mode drivers that support auto-detection to configure the device from a range of pre-configured settings	Values between 0x00000000 and 0x0000000F supported. Please refer to the boot mode specific documentation for details on a boot modes supported BCODE.	

Secure Boot Streams

The secure boot image sections contain the code and data for loading to the various memory regions of the processor. The content is in the form of a boot stream format consisting of block headers that provide a descriptor for the associated block payloads.

The image section within the secure boot stream consists of a standard boot Boot Loader Stream consisting of block headers and payloads as generated by the supporting tools and utilities. Refer to the section describing the format of the boot stream for further details.

The secure boot image section can contain application images for just a single core or for multiple cores allowing for a flexible booting strategy.

A single boot image containing program and data for multiple cores permits a full system boot initializing all internal memories without needing a second stage loader approach. If boot images are required to initialize external memories, then a multi-stage loader approach can be required to configure the memory interfaces. The advantage of a single boot image for multiple cores is that authentication and decryption of the boot image requires only one execution. However, it is over a larger boot image.

Multiple single-core boot images permit one core to boot then execute that core's boot image without booting the other cores. This booting strategy can be needed when a single processor must be brought up quickly to deal with some initial tasks before booting the rest of the system. Or, it can be used to initialize extra peripherals to use for external memory interfaces. The core that has previously booted can then control the boot process for the remaining cores.

Single Core Images

Single core boot images are the result of processing a single core's output from the linker and converting it to a compliant boot stream. The resulting boot stream has a single first block at the beginning of the boot stream and a final block at the end of the boot stream.

The Block Header `HDRSIGN` field of the resulting boot stream is used to identify which core the image is intended for. This identification is required so that the boot code can set the correct `RCU_SVECT[n]` when the first block is read to set the application start address for that core. When the boot image is loaded and authentication is successful, the boot code jumps to the location stored in the cores corresponding `RCU_SVECT[n]` register.

This boot stream must then be Secure Boot Image Signing using the secure boot utilities resulting in the final secure boot image for a single core.

A second stage loader option is also available giving maximum flexibility both in terms of using ROM functionality, or creating a custom booting strategy. The simplest option is to have the application utilize `adi_rom_Boot()` to boot the main application image or indeed a third stage loader when one is required.

The boot stream is generated such that all image contents are in the system address space. Core specific L1 memory sections are converted during the boot stream generation process such that they are loaded through the multiprocessor memory space. The core executing the second stage loader can boot an application image intended for another core.

NOTE: The `ROM_BFLAG_RETURN` flag is set when calling the boot routine for the other cores than the core executing the second stage loader then cleared when loading the cores own image. Failure to do so would result in unintentional behavior.

When a core is used to boot a separately generated boot stream for another core, the `ROM_BFLAG_RETURN` flag should be used when calling `adi_rom_Boot()` instructing the boot kernel to return to the calling application, which in this case is back to the core that was running the second stage loader. Failure to set this flag would result in the core vectoring to the location in its own `RCU_SVECT[n]` register which would not be the intention. In order to allow the core that was just booted to start running the newly loaded application.

By default when implementing a scheme such as previously described where one processor is responsible for booting images for another core, the other cores in the system will remain in their existing idle state that they should be placed in prior to the boot commencing. In order to allows the cores to execute their new application the corew that was responsible for booting the other cores must reset the cores via the `RCU_CRCTL` then release them again from reset. If there is a requirement to release the core from the reset state, then this must be done within the application code of the running core.

Releasing other cores from reset to run application software while other processors are running a boot process requires careful system design. The drivers used by the boot kernel for the boot process assume the various peripheral and infrastructure resources such as MDMA channels and peripherals are available for use, and are not being used by another core. If the boot routine is being executed while other cores are running applications, then those applications must ensure that all the required boot resources are freed up and remain free in order for the boot process to complete on the remaining cores.

Multi-core Boot Images

Multi-core boot images are generated as a result or processing the multiple linker output files for multiple cores together to create a single compliant boot stream. The resulting boot stream has a first header block at the beginning of each core boot stream and a single final block at the end of the boot stream. The boot stream must only have a single final block to allow the boot kernel to continue processing the entire boot stream. A final block results in the boot kernel terminating triggering the final public key authentication sequence.

The block headers `BLOCK_CODE.HDRSIGN` field of the resulting boot stream is used to identify which core the image is intended for. This identification is required such that the boot code can set the correct `RCU_SVECT[n]` when the first block is read to set the application start address for that core. When the boot image is loaded and authentication was successful, the core executing the boot sequence jumps to the location stored in the cores corresponding `RCU_SVECT[n]` register. However, as the boot stream has multiple first blocks present for each of the cores, all other cores `RCU_SVECT[n]` register is set for their applications. Upon the booting core running its application, it must release the other cores from reset in order for them to then start running their loaded applications.

If a product supports three or more cores, it is acceptable to create a dual core boot image and then load the other cores later using single core boot images stored elsewhere in the boot source. There are no restrictions on a multi-core boot stream requiring to contain an application for all the cores in a product.

The resulting multi-core boot stream must then be signed and optionally encrypted resulting in a complaint secure boot stream that can be used to boot a secure device.

Multiple core boot images are advantageous in the fact that applications can be loaded to all cores in the system in a single boot sequence resulting in the requirement to decrypt and authenticate only a single boot image.

NOTE: If there is a requirement for a multi-core boot stream to load code to memory spaces that require a memory controller to be initialized that is not supported by the boot process a multi-stage booting approach is required that can initialize the peripheral. This boot stream is then authenticated, and executed prior to the loading of the boot stream.

Secure Debug Access

The TAPCcontroller provides a means of restricting access to secure resources of the processor. Secure access through the debug port is protected through a 128-bit security key that must match a key that has been loaded into OTP for access.

To access a locked processor, the TAPC must allow access to the part. The TAPC only allows access to the part if it is provided with a matching key to the data loaded into its `TAPC_USERKEYn` registers.

With the processor in a locked state, on initial boot the boot ROM reads the 128-bit `secure_emu_key` from the OTP memory and programs the key into the `TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, and `TAPC_SDBGKEY3` registers before then setting the `TAPC_USERKEY_CTL.USERKEY_VALID` bit. Then, the TAPC is able to access a matching outside key to allow access.

CAUTION: A key of 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF provisioned in OTP will result in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by user software. If the processor fails to boot perhaps due to corrupted firmware then the user will have no debug access. The only way to gain access would be to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger.

The key is set in OTP using the OTP boot program API to program the `secure_emu_key` , this key is read and loaded by the ROM in the following sequence:

1. Bits 31:0 of the key in OTP are stored to `TAPC_SDBGKEY0` bits 31:0
2. Bits 63:32 of the key in OTP are stored to `TAPC_SDBGKEY1` bits 31:0
3. Bits 95:64 of the key in OTP are stored to `TAPC_SDBGKEY2` bits 31:0
4. Bits 127:96 of the key in OTP are stored to `TAPC_SDBGKEY3` bits 31:0

Once the ROM has loaded the user key, a test key can be provided to the TAPC through JTAG. Refer to the Emulator manual for details for providing the key.

A key failure indication can be detected through the `TAPC_SDBGKEY_STAT` register. The boot code does not check the key status, nor does it enable any associated interrupts to signal key failure. The boot code continues to boot upon a key failure in a secure manner. The key failure status remains intact so that the application loaded can check for a failed challenge on the debug port.

The boot code can be configured to bypass the loading of the key during the boot sequence by setting the value of the `otp_data::secure_emu_key` in the OTP to all ones. In this case, the only way to gain access to the secure resources through the debug port is to load an alternate key using the application. The alternative key must always reside in a secure region of memory. Or, if sent remotely, it should be transmitted over a secure connection.

Errors and Failures

Any errors encounters while processing a secure boot image results in the ROM jumping to the Error Handler . This includes decryption failures, authentication failures, and configuration errors.

As the boot process doe no checking for a matched secure debug key, should an incorrect key be supplied during boot, no boot failure will occur and the processor will continue to boot as normal. A user supplying an incorrect key will not be able to gain access to any secure resources of the processor.

Boot ROM Programming Model

This section describes the programming model for booting the processor. The programming model includes booting functions, API calls, and data structures.

Boot Mode Driver API

The kernel provides a mechanism to provide a customization of supported boot modes or for implementation of completely new boot modes as second stage boot loaders. This allows users to customize booting while still taking advantage of the rest of the booting framework. A custom boot mode could provide support for a peripheral that is not supported for boot by the ROM, or it could support one of the same peripherals but with a different configuration.

All the same security features can be supported when using a custom boot mode.

A full boot mode, as perceived by the boot implementation, is a collection of five functions.

- Register - installs the driver functions listed below so they can be accessed by the boot process
- Initialization - initialize the boot source
- Configuration - configure the boot source
- Load - read from the boot source
- Cleanup - called after booting

Of these later four functions the boot kernel is only ever aware and has a requirement to support the Load function. It is this function that is responsible for the fetching of the boot stream from the boot peripheral. The other functions are used prior to executing the kernel or for cleaning up after the kernel has completed processing the boot stream.

To install a custom boot mode:

- Create a first stage boot application to define a Load function

- Use the `adi_rom_BootKernel()` API to call the boot kernel once the boot peripheral and pinmuxing has configured. Ensure all the fields of `ADI_ROM_BOOT_CONFIG` are configured accordingly prior to performing the call.

The boot mode can use the `pModeData` member of `ADI_ROM_BOOT_CONFIG` to preserve and access shared data across the different function calls if required.

All functions have the following prototype:

```
          void apiFunction(
ADI_ROM_BOOT_CONFIG* pBootStruct);
```

Load Function

The load function is required to read data from the source into the specified destination, according to the parameters given through the configuration struct parameter. The structure provides all of the required information read from the block header, or specified by the kernel to read the block header. The load function often makes use of the supplied DMA APIs in order to simplify the load function implementation.

As the kernel processes the stream, it calls the load function to request data. Initially, the request is for the header, then the kernel requests according to the block flags it parses. The load function must only read from the device, and write where requested.

Relevant fields within the `ADI_ROM_BOOT_CONFIG` object for the load function can be (not limited to): `uwDataWidth`, `pSource`, `dByteCount`, `pDestination`, `loadType`.

Custom load functions must meet the following requirements.

- Protect against `dByteCount` values of zero
- Use multiple DMA units if `dByteCount` is greater than 65536 and the peripheral does not support byte count transfers greater than 65536
- The `pSource` and `pDestination` pointers must be properly updated after loading.

In slave boot modes, the boot kernel uses the address of the `dArgument` field in the `pHeader` block as the destination for the required dummy DMAs when payload data is consumed from `ROM_BFLAG_IGNORE` blocks. If the load function requires access to the `ARGUMENT` word of the block, it should be read early in the function.

Initialization/ Configuration Function

The initialization and configuration functions are called in sequence when calling a boot operation using an already supported boot peripheral via the `adi_rom_Boot()` API. These functions are used to configure the boot peripheral prior to calling the boot kernel. Both functions are called in sequence separated only by a call to a user-defined hook function. This hook function is useful when using built-in boot modes to further customize their functionality. The initialization and configuration functions are responsible for applying any required settings to any devices in use. For example, pin multiplexing may need to be applied, and any data or pointers that are used by the load function must be initialized. The specific actions depend on the device and functionality used.

Cleanup Function

The cleanup function is called after the entire boot stream is read, and the kernel has completed its boot mode-specific function. This is only performed when using the `adi_rom_Boot()` API. Resetting of any status registers, or device parameters is done to prepare the environment for the execution of the newly loaded application.

Error Handler

This section describes the default error handler for the ROM including information on how to customize the error handling.

The default error handler eventually puts the core into an idle state. This functionality can be overridden by using an Init Block (see Block Types) to modify the error function point in the `ADI_ROM_BOOT_CONFIG` structure. The error handler has access to the entire boot info structure and receives the instruction address that triggered the error.

When a part is locked, and the boot type has not disabled secure boot, only the default error handler is called.

The expected prototype is:

```
void ErrorFunction(
ADI_ROM_BOOT_CONFIG* pBootStruct, void *pFailingAddress);
```

The error handler saves the failing address to the `ADI_ROM_BOOT_CONFIG` structure then raises the `INTR_SOFT3` fault signaling a fault condition to the system before then entering an endless loop in the boot rom.

NOTE: When using the `adi_rom_Boot()` function to perform a boot action. Users may need to manually configure the `INTR_SOFT3` fault signaling depending on previous application software executed. Calling the boot process via `adi_rom_Boot()` does not result in all the SEC and Fault configuration being reset and installed as described in Preboot Operations.

Page Mode

For the benefit of page oriented boot source devices, and to improve boot performance for secure boot operations, the boot kernel provides support for page operations. Page mode optimizes memory reads for block organized devices by always reading a page, rather than reading data on demand. Two 1024 byte buffers are used in page mode.

Two buffers are used allowing the contents of one buffer to be processed by the boot kernel while DMA is used to load the next data into the second buffer.

The load to the active buffer uses a blocking DMA, forcing the process to pause until the DMA is complete. Loading to the non-active buffer uses non-blocking DMA allowing the active buffer to be processing while loading the new data in parallel.

Page mode can be enabled when calling a boot mode via the `adi_rom_Boot()`. Refer to the API documentation for the various supported by this API. Additionally users can set the flag via `ADI_ROM_BOOT_CONFIG::dFlags` when using hook functions

NOTE: Due to security requirements it is not recommended to customize the page mode settings from the default installed by the boot process.

Boot Hook Function

The boot software allows installation of callback hooks through the use of the `adi_rom_Boot()` APIs hook function parameter. By using this feature, it is possible to alter the state of the processor, at different stages of the boot process and customize the boot structures to alter the behavior of the boot process.

The hook function must adhere to the following prototype:

```
int32_t
hookFunction(
ADI_ROM_BOOT_CONFIG* pBootconfig,
ROM_HOOK_CALL_CAUSE cause);
```

By modifying settings in the `ADI_ROM_BOOT_CONFIG` structure, many alterations of the boot process can be achieved. Much of the same functionality that is available in an Init Block can be provided through the hook function, with even more flexibility for customization. The hook function is called once after executing the boot modes Init routine then once again after executing the boot modes Config routine. A flag passed to the hook function allows software to determine at which point the call took place to allow for conditional processing to occur at different stages of the setup phase.

The hook function must return a zero value in order for normal booting to continue. A non-zero return value will cause the ROM to skip over loading of any data and immediately transfer control according to Boot Termination and Application Execution .

When the hook function is called, a parameter is passed indicating why the hook function was called. Refer to `ROM_HOOK_CALL_CAUSE` for further details.

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

When calling a boot mode via adi_rom_Boot, the user may provide an optional hook routine as a callback. This hook routine is called by the boot software firstly after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 53-37: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
	Call was as a result of completion of the boot modes initialization function

Table 53-37: ROM_HOOK_CALL_CAUSE Members (Continued)

Enumerator	Description
`ROM_HOOK_CALL_INIT_COMPLETE`	
`ROM_HOOK_CALL_CONFIG_COMPLETE`	Call was as a result of the completion of the boot modes configuration function

ROM_HOOK_CALL_INIT_COMPLETE

Call was as a result of completion of the boot modes initialization function

ROM_HOOK_CALL_CONFIG_COMPLETE

Call was as a result of the completion of the boot modes configuration function

Boot Return Feature

The `adi_rom_Boot()` API provides a feature to bypass calling of the loaded application upon boot completion, and to simply return to the routine that made the call instead. This can be useful when using the . The boot software returns the next address after the last loaded application block in the boot source when this feature is enabled..

To enable this feature, set the `ROM_BFLAG_RETURN` flag in the `flags` argument when calling the API.

Boot Termination and Application Execution

When the boot kernel completes the processing of the boot stream, a sequence of events is required to then pass control to the loaded application.

When the boot process is complete, the core sis required to vector to the application start address and start executing the newly loaded application. Typically, the first block of a boot stream, which is marked with the `BFLAG_FIRST` flag, contains the address of the application. In a multi-core system there may be multiple first blocks in the boot stream indicating the start address of the application for each core. The application entry point for each core is loaded into the cores corresponding `RCU_SVECTn` register.

Upon boot completion only the core that performed the boot process will vector and start executing the loaded application. This core must then manage the process of resetting then releasing from the reset the other cores in the system in order to make them execute their newly loaded applications.

Execution of the loaded application can be bypassed when calling the boot mode via `adi_rom_Boot()` and setting the `ROM_BFLAG_RETURN` flag.

Table 53-38: Application Entry Point Registers

Core ID	Corresponding RCU_SVECTn Register
0	`RCU_SVECT0`
1	`RCU_SVECT1`

Table 53-38: Application Entry Point Registers (Continued)

Core ID	Corresponding RCU_SVECTn Register
2	RCU_SVECT2

API Reference

The APIs defined in this section are exposed for general use.

adi_rom_Boot()

Provides access to boot an application at run-time through a supported peripheral.

API Details

```
void * adi_rom_Boot(
                    void * pAddress,
                    uint32_t flags,
                    int32_t blockCount,
                    ROM_BOOT_HOOK_FUNC * pHook,
                    uint32_t command
                   )
```

pAddress

Pointer to source address of the boot stream.

flags

contains the global flags to be applied to the entire boot process

blockCount

Number of block to be booted. Zero results in processing until a final block is reached.

pHook

Pointer to user implemented hook function for enabling callbacks during the registering of the boot mode with the boot kernel

command

The boot command defining the boot mode to use, the peripheral instance to boot from as well as some boot mode specific configuration

Returns

The 32-bit address of the next address in the boot source to be processed

Function Description

This function may be used for any kind of second-stage boot for an already supported boot mode. It provides options to boot from any peripheral enumeration and in the case of SPI Master boot using any SPI slave select signal.

Boot modes may support an auto-detection mechanism to identify the type of connected device and the function provides options to bypass such auto-detection and use custom configuration options. Options are also provided to bypass peripheral configuration such as pinmux settings or peripheral configuration if existing peripheral already configured are more appropriate to allow communication with the boot source.

These features are all provided via the command parameter which is specific for each particular boot mode.

The source address of the boot stream is required for master boot modes that require an address to be issued in order to request data from the boot source. Slave boot modes are under full control of the host and use a handshake mechanism to indicate that the processor is ready to receive data. For boot modes such as UART Slave and SPI Slave this parameter is of little value in regards to the boot process itself however it can prove useful in debug to see how far through the boot stream the boot process got in the event of a boot failure.

NOTE: The processor supports both SPI Memory Mapped boot as well as Peripheral Based SPI Boot. When the boot mode is called to boot from the memory mapped boot mode via the command argument the address must coincide with the processors memory mapped SPI address space as defined by the processors internal memory map. When using the peripheral based boot mode the absolute address of the boot stream in flash must be used.

Flags passed via the flags argument are global flags and the functionality gets applied throughout the entire boot process. These must not be confused with the boot block specific flags which are part of the boot stream and indicate how a particular block in the boot stream is processed. Internally the boot kernel will take the global flags supplied via this function call and combine them with a boot blocks local flags to determine all the operations to be performed on a given block. After processing the boot block the local flags get cleared ready to be populated from the next boot block and the global flags remain.

The global flags supported by the product are:

Bit Position	Flag Name	Description
18	ROM_BFLAG_HOOK	Calls the user supplied hook function after execution of bootmode init and config routines
19	ROM_BLAG_PAGEMODE	Enables page mode processing where blocks of data are fetched and processed from internal memory
20	ROM_BFLAG_NOFIRSTHEADER	Set this if calling the boot mode and the first block header has already been fetched and present in the block header storage location
21	ROM_BFLAG_HEADER	Not intended to be set by the user, set by the boot code each time it fetched a block header
22	Reserved	Reserved
23	Reserved	Reserved
24	Reserved	Reserved
25	ROM_BFLAG_PERIPHERAL	Boot mode is a peripheral boot mode as opposed to a memory boot mode

Bit Position	Flag Name	Description
26	ROM_BFLAG_SLAVE	Boot mode is a slave boot mode. This results in different handling of ignore blocks by the kernel
27	ROM_BFLAG_WAKEUP	Set this to enable conditional processing of boot blocks intended for wakeup events but not exclusively
28	ROM_BFLAG_NEXTDXE	Parse stream via Next DXE pointer
29	ROM_BFLAG_RETURN	Return the application after calling the API instead of running the new application
30	Reserved	Reserved
31	ROM_BFLAG_NORESTORE	Do not execute the boot peripherals cleanup routine to restore register contents

The blockCount argument specifies the number of blocks to be processed before terminating the boot process. The default would normally be 0x00000000. A value of 0x00000000 instructs the boot software to continue processing a boot stream until the `ROM_BFLAG_FINAL` flag is set. Should users wish to load only a specified number of blocks they can instruct the boot kernel to do so via this parameter.

When the block count is used in combination with the `ROM_BFLAG_NEXTDXE` flag then the block count is re purposed as a next application count. The boot kernel will navigate the first blocks of multiple boot streams similar to a linked list and upon reaching the requested application count will return the pointer to this application in the boot source. This allows users to use the boot kernel to find a specific application when multiple application boot stream are stored contiguously in the boot source.

The pHook argument is a function pointer to a hook routine. When set along with the `ROM_BFLAG_HOOK` global flag the boot mode will call the hook routine after calling the boot modes `init` and functions in the boot modes driver allowing customization and altering of the configuration performed by the boot software.

This can be used to enabled new features not supported by the boot software and allows for installing of a user defined load function or error handler as an example.

The command argument describes the boot peripheral to boot from, the peripheral instance and contains additional boot mode specific configuration information and flags specific to the boot mode.

NOTE: When calling a boot mode via this API the user must first ensure that the boot peripheral is confirgured accordingly in the SPU as a secure master. The boot software does not configure the peripheral security via this API in order to allow device security to be fully controlled by a dedicated task.

adi_rom_BootKernel()

Calls the boot kernel allowing for implementation of custom boot modes.

API Details

```
void * adi_rom_BootKernel(ADI_ROM_BOOT_CONFIG * pBoot)
```

pBoot

Pointer to the `ADI_ROM_BOOT_CONFIG` boot structure containing the complete context of the boot configuration

Returns

Pointer containing the address of the byte immediately following the end of the boot stream.

Function Description

The boot kernel performs the core processing of the boot stream. The boot kernel calls a load function to load in data from the peripheral to the required destination. The boot kernel itself has no concept of what the boot peripheral is or how that peripheral is configured. The kernel calls the registered load function and the load function must then analyze the boot structure and provide the requested amount of data to the required destination.

The load function called by the kernel is provided via the `ADI_ROM_BOOT_REGISTRY::pLoadFunction` member of `ADI_ROM_BOOT_CONFIG::bootRegistry`.

The boot kernel basically works in a cycle of fetching a boot stream block header then a payload if one exists. The boot kernel takes care of the size of the data being requested and the destination address.

The load function that is registered with the kernel is required to update the `ADI_ROM_BOOT_CONFIG::pSource` member. Keeping this control under the load function as opposed to the boot kernel itself allows load functions to better control where the next block of data is fetched in the event the boot stream is fragmented or split into different areas of the boot source.

This function would typically be used to implement a second stage boot loader for a peripheral in which there is no driver support in the boot rom. The user is responsible for initializing the peripheral prior and the complete `ADI_ROM_BOOT_CONFIG` object prior to calling this function. Upon return form the function the user application is then responsible for performing a vector to the newly loaded application.

NOTE: Users must ensure that a new application being loaded does not clobber the load function and the part of the software responsible for making the core jump to the start of the newly loaded application.

adi_rom_Crc32Init()

The CRC32 Initcode function in the boot rom that is called to enable CRC32 support of boot stream payloads.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Init(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the `ADI_ROM_BOOT_CONFIG` object containing the complete boot configuration

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_CRC_INITCODE_ERR` when pBootConfig or pBootConfig->pHeader are zero
- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` when the callback is registered and lookup table initialized

Function Description

The boot process supports CRC32 protection of all boot block payloads. In order to enable this feature a global callback must be registered with the boot process via `ADI_ROM_BOOT_CONFIG::pCrcFunction` and the CRC peripherals look up table initialized from the users polynomial.

In order to enable the CRC functionality a init block header in which the `ROM_BFLAG_INIT` flag is set must be included in the boot stream. The_ADI_ROM_BOOT_HEADER::pTargetAddress field must be set to the address of this function and the users polynomial is provided via the blocks `ADI_ROM_BOOT_HEADER::dArgument` member.

When the boot kernel processes the init block described the boot kernel calls this function in the boot rom registering the callback with the kernel and performing the look up table initialization.

The CRC functionality is enabled on `MDMA channel 1` interfaced to the `CRC0` peripheral instance

adi_rom_Crc32Poly()

Initializes the CRC peripheral for use with the user supplied polynomial.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Poly(
                                  uint32_t CrcPoly,
                                  ROM_BOOT_MDMA_REGS const *const pDma
                                 )
```

CrcPoly

None

pDma

None

Function Description

In order to prepare the CRC peripheral for use, the CRC lookup table must be initialized for the CRC polynomial of choice. Users may perform this task via this function.

adi_rom_GetAddress()

Used to find the location of various look-up tables and data objects used during the boot process.

API Details

```
int32_t adi_rom_GetAddress(ROM_GETADDR_VALUE value)
```

value

The `ROM_GETADDR_VALUE` enumeration specifying the object to retrieve the address of in the ROM memory

Returns

The byte address of the object in memory

Function Description

The function returns the address of the object specified by the enumerator provided as an argument to the function. Using this function cane make software more code compatible with future products and silicon revisions.

Table 53-39: ROM_GETADDR_VALUE Members

Enumerator	Description
`ROM_GETADDR_CONSTANTS`	Retrieve the address of the `ROM_CONSTANTS_TYPE` object
`ROM_GETADDR_BMODE`	Retrieve the address of the lookup table sotring the default adi_rom_boot() parameters for each boot mode
`ROM_GETADDR_MDMAREGS`	Retrieve the address of the `ROM_BOOT_MDMA_REGS` object
`ROM_GETADDR_SPILUT`	Retrieve the address of the lookup table in the rom describing the various SPI master boot BCODE configurations
`ROM_GETADDR_ECDSA_DOMAIN`	Retrieve the address of the domain parameteres used for ECDSA

adi_rom_MemCompare()

Verifies that a block of data is filled with a user supplied 32-bit value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCompare(
                                   ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                   ROM_BOOT_MDMA_REGS const *const pDma
                                  )
```

pDmaCfg

None

pDma

None

Function Description

The CRC peripheral used in compare mode and a source MDMA channel is used to read data from a buffer and supply each 32-bit value to the CRC. The CRC peripheral checks the incoming 32-bit value matches the 32-bit value to compare against.

adi_rom_MemCopy()

Performs a Memory to Memory DMA (MDMA) operation using a source and destination pair of DMA channels.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCopy(
                                ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                ROM_BOOT_MDMA_REGS const *const pDma
                              )
```

pDmaCfg

Pointer to the `ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

pDma

Pointer to the `ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channels MMRs and associated CRC peripheral

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation to be performed
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- if a configuration error occurred in the source MDMA channel
- if a configuration error occurred in the destination MDMA channel

Function Description

The memory copy routine performs transfers of blocks of data from one memory location to another. The routine takes a basic descriptor providing configuration details of the operation to be performed via the `ROM_DMA_MDMA_CONFIG` object passed as the first argument. The second argument is a descriptor that provides access the DMA channels MMR registers and associated CRC peripheral. When called from the higher level `adi_rom_MemDma()` routine this object is retrieved from the ROM.

NOTE: Users are expected to make use the `adi_rom_MemDma()` routine for all MDMA operations, there is little additional optional configuration that is supported by using this routine.

adi_rom_MemCrc()

Performs CRC32 verification of a block of block of data by reading the contents and comparing with an expected result.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCrc(
                               ROM_DMA_MDMA_CONFIG * pDmaCfg,
                               ROM_BOOT_MDMA_REGS const *const pDma
                              )
```

pDmaCfg

Pointer to the `ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

pDma

Pointer to the `ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channels MMRs and associated CRC peripheral

Function Description

The routine uses an MDMA channel pairs source DMA channel and the CRC peripheral to calculate a CRC32 result of a data block using a previously supplied polynomial.

The polynomial should be supplied through the `adi_rom_Crc32Poly()` routine in order to ensure consistent CRC peripheral configuration for the look up table initialization that uses the polynomial and the

NOTE: Users are expected to make use the `adi_rom_MemDma()` routine for all MDMA operations, there is little additional optional configuration that is supported by using this routine.

adi_rom_MemDma()

Provides access to all the MDMA operations supported by the boot ROM implementation.

API Details

```
ROM_BOOT_RESULT adi_rom_MemDma(ROM_DMA_MDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the `ROM_DMA_MDMA_CONFIG` object containing the MDMA configuration

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation to be performed

- `ROM_BOOT_RESULT::ROM_BOOT_MDMA_ID_ERR` if an MDMA channel ID is provided that is not supported
- `ROM_BOOT_RESULT::ROM_BOOT_CRC_SUPPORTED_ERR` if a CRC operation was attempted on an MDMA channel not supporting CRC
- `ROM_BOOT_RESULT::ROM_BOOT_MDMA_OPERATION_ERR` if the operation to be performed is not supported
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel and the operation requested involves only a single DMA channel
- if a configuration error occurred in the source MDMA channel
- if a configuration error occurred in the destination MDMA channel
- `ROM_BOOT_RESULT::ROM_BOOT_CRC_COUNT_ERR` if a CRC operation is being requested and the byte count is not a multiple of 4
- `ROM_BOOT_RESULT::ROM_BOOT_CRC_FAILURE` if CRC32 verification fails
- `ROM_BOOT_RESULT::ROM_BOOT_CRC_FAILURE` if 32-bit memory compare fails

Function Description

The MDMA operations supported are:

Table 53-40: ROM_DMA_MDMA_OPERATION Members

Enumerator	Description
`ROM_DMA_MEM_COPY`	Standard MDMA transfer from a source to a destination
`ROM_DMA_MEM_CRC`	Performs a CRC32 MDMA read operation and compares the result with an expected result
`ROM_DMA_MEM_FILL`	Uses the CRC peripheral to perform a fill operation with a 32-bit value
`ROM_DMA_MEM_COMPARE`	Uses the CRC peripheral to compare data with a constant 32-bit value
`ROM_DMA_CRC_LUT_INIT`	Initializes the CRC LUT from the supplied CRC Polynomial

NOTE: While the MDMA and CRC peripherals support on the fly CRC32 calculations during the transfer of data from one location to another, the MDMA functionality of the boot ROM software does not support this. For CRC calculations data is instead read back from its final destination and verified.

While this function is the main entry point for all the MDMA functionality supported by the boot ROM software, the individual functions that are called for each operation type are also exposed via the public API.

NOTE: The recommendation is to use this function for all operations. The lower level functions allow for some basic reconfiguration of default parameters however such modifications should not be required in most cases where these basic MDMA operations are required.

The boot ROM has an MDMA configuration data structure included that is used to specify the overall MDMA configuration of the processor. It provides details on the DMA channel ID associated with each MDMA channels source and destination DMA channels as well as information relating to CRC support and the CRC peripheral instance that is to be used for a given MDMA channel. Please refer to `ROM_BOOT_MDMA` and `ROM_BOOT_MDMA_REGS` for full details of the content stored.

The configuration provided as the only argument to the function is provided below:

Table 53-41: ROM_DMA_MDMA_CONFIG Members

Type	Name	Description
`ROM_DMA_MDMA_OPERATION`	`eOperation`	Type of operation to perform
`ROM_DMA_MDMA_ID`	`eId`	MDMA Channel ID
`void *`	`pSource`	Source Pointer
`void *`	`pDestination`	Destination Pointer
`uint32_t`	`ByteCount`	Byte Count
`ROM_DMA_DONE_DETECT_METHOD`	`eDoneDetect`	DMA Done Detection Method
`uint32_t`	`CrcCtl`	`CRC_CTL` value when CRC operations are required
`uint32_t`	`FillVal`	Fill value for memory fill operations
`uint32_t`	`CrcPoly`	CRC Polynomial for CRC operations
`uint32_t`	`CrcCompare`	Value used for CRC compare operations or for a CRC32 result compare

For a basic MDMA transfer from source to destination the user needs to configure:

- The `ROM_DMA_MDMA_CONFIG::eOperation` type as `ROM_DMA_MDMA_OPERATION::ROM_DMA_MEM_COPY`
- The MDMA channel to use via `ROM_DMA_MDMA_CONFIG::eId` for example
- Set the address of the source data via `ROM_DMA_MDMA_CONFIG::pSource`
- Set the destination address of the source data via `ROM_DMA_MDMA_CONFIG::pDestination`
- Set the byte count via `ROM_DMA_MDMA_CONFIG::ByteCount`
- Set `ROM_DMA_MDMA_CONFIG::eDoneDetect` to `ROM_DMA_DONE_DETECT_METHOD::ROM_DMA_DONE_POLL_IRQDONE` in order to poll for DMA completion

NOTE: The implementation of the MDMA operations does not support interrupt driven data transfers the routines were implemented for the intentions of polling on the DMA status for use during the boot process. Also there are restrictions in regards to the boot stream in regards to byte counts and source and destination address alignment all being a multiple of 4 bytes and as such, compliance to these restrictions must be adhered to when using these MDMA routines.

When wishing to use the CRC32 functionality of the MDMA routines, the user must first of all initialize the CRC lookup table from the user supplied polynomial. This operation can be performed by setting `ROM_DMA_MDMA_CONFIG::eOperation` type as `ROM_DMA_MDMA_OPERATION::ROM_DMA_CRC_LUT_INIT`. If an MDMA channel is specified that does not support CRC functionality an error result is returned.

For further details of the individual operations supported please refer to the following API references:

- `adi_rom_MemCopy()`
- `adi_rom_MemCrc()`
- `adi_rom_MemFill()`
- `adi_rom_MemCompare()`
- `adi_rom_Crc32Poly()`

adi_rom_MemFill()

Fills a block of memory with a 32-bit user supplied value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemFill(
                                ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                ROM_BOOT_MDMA_REGS const *const pDma
                               )
```

pDmaCfg

None

pDma

None

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation to be performed

- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- if a configuration error occurred in the source MDMA channel
- if a configuration error occurred in the destination MDMA channel

Function Description

The CRC peripheralis configured for fill mode and the destination MDMA channel is configured to fill a block of memory with a fixed 32-bit value.

adi_rom_PeriphDma()

Provides access to any peripherals dedicated DMA channel for receive operations only.

API Details

```
ROM_BOOT_RESULT adi_rom_PeriphDma(ROM_DMA_PDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the ROM_DMA_PDMA_CONFIG object containing the peripheral DMA configuration

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS for a successful operation or when byte count is 0 as no operation to be performed
- ROM_BOOT_RESULT::ROM_BOOT_DMA_ACTIVE if the DMA channel is currently running
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel after starting the DMA operation

Function Description

The peripheral DMA routine is used by the load routines of boot peripherals that have dedicated DMA channels and do not support MDMA channel pairs. Examples are the SPI when not configured for memory mapped mode and UART peripherals.

In the boot implementation this routine is called from the peripheral load function to request data from the boot source. The routine supports both polling on DMA completion and non-blocking operation to allow for immediate return after starting the DMA operation and continuing with further processing.

NOTE: The function only supports read operations from the peripheral to memory. Transmit operations from memory to peripheral are not supported

adi_rom_otp_cfg()

Configures the `OTPC` to enable read and program operations to be be performed.

API Details

```
bool adi_rom_otp_cfg(void)
```

Function Description

Users may call this routine to ensure the `OTPC` is configured correctly for read and write access.

NOTE: The preboot process configured the `OTPC` for use and as such there should be no direct requirement to call this function when using the OTP.

adi_rom_otp_get()

Reads the field from `OTP` as defined by the supplied `OTPCMD`.

API Details

```
bool adi_rom_otp_get(
                     OTPCMD cmd,
                     uint32_t data[]
                    )
```

cmd

The `OTPCMD` enumeration describing the OTP content to be read

data[]

Pointer to storage area to store the read OTP contents

Function Description

Users can read the various fields of the OTP via this routine. The supplied `OTPCMD` object is used to specify the object to be read.

Table 53-42: OTPCMD Members

Enumerator	Description
`otpcmd_reserved0`	Reserved
`otpcmd_huk`	Hardware Unique Key
`otpcmd_DTCP_key_ecc`	DTCP Key (ECC Parameters)
`otpcmd_DTCP_key_cont`	DTCP Key (constant for content key)
`otpcmd_DTCP_key_dev`	DTCP Key (device specific keys)

Table 53-42: OTPCMD Members (Continued)

Enumerator	Description
`otpcmd_pvt_128key0`	Customer Private AES Key0
`otpcmd_pvt_128key1`	Customer Private AES Key1
`otpcmd_pvt_128key2`	Customer Private AES Key2
`otpcmd_pvt_128key3`	Customer Private AES Key3
`otpcmd_ek`	Endorsement Key
`otpcmd_secure_emu_key`	Secure Emulation Key
`otpcmd_public_key0`	Customer Public Key0
`otpcmd_public_key1`	Customer Public Key1
`otpcmd_boot_info`	Customer Programmable Boot Information
`otpcmd_otpTiming`	OTP Read timing override
`otpcmd_antiroll_nv_cntr`	AntiRollback NV Counter
`otpcmd_gp1`	General Purpose 1
`otpcmd_bootModeDisable`	Boot Mode Disable Bits
`otpcmd_preboot_ddr_cfg`	User DMC configuration
`otpcmd_stageID`	StageID
`otpcmd_reserved1`	Reserved

adi_rom_otp_lock()

Locks the processor, enabling all security features.

API Details

```
bool adi_rom_otp_lock(void)
```

Function Description

This function si used to lock the processor securing the device from unauthorized access. Once called users must supply a secure debug key in order to gain access to the device with debug tools and the part may only be booted using a secure boot stream.

WARNING: Users must ensure that the OTP secure boot fields are all programmed. Secure boot can be verified prior to locking the processor. Users should also provision a secure debug key.

adi_rom_otp_pgm()

Programs the `OTP` Memory with the contents of the `otp_data` object.

API Details

```
bool adi_rom_otp_pgm(otp_data * data)
```

data

Pointer to the::otp_data object containing the complete OTP contents to program

Function Description

The OTP memory is only programmed with values that are not 0. Any items that are 0 are ignored. Users are expected to use this function for all OTP program operations.

callback()

Callback function for implementing custom callbacks to previously loaded code during boot.

API Details

```
ROM_BOOT_RESULT callback(
                         ADI_ROM_BOOT_CONFIG * pBootConfig,
                         ADI_ROM_BOOT_BUFFER * pBuffer,
                         uint32_t nFlags
                        )
```

pBootConfig

Pointer to the `ADI_ROM_BOOT_CONFIG` object containing the complete context of the boot procedure

pBuffer

Pointer to the `ADI_ROM_BOOT_BUFFER` object containing details of the payload associated with the callback

nFlags

The callback flags as set by the boot kernel

Function Description

A single callback function may be registered with the boot kernel via `ADI_ROM_BOOT_CONFIG::pCallBackFunction`. This functions is then called whenever a block is processed with the `ROM_BFLAG_CALLBACK` flag set. Only a single callback function can be registered for the complete boot process.

Callbacks may typically be used alongside indirect blocks. This would be used if there was a requirement for post processing of the received boot data before sending to the final destination. An example of this would be if compression was applied to block payloads. The compressed payload would be loaded indirectly to the intermediate buffer where it would be decompressed by the callback. The callback can modify the source address and byte count for the final MDMA transfer of the decompressed payload via the supplied pBuffer parameter such that when the callback returns the boot kernel then handles the final transfer of the uncompressed data to the destination.

When dealing with indirect blocks, there are restrictions on the amount of data that can be loaded depending on the size of the intermediate buffer. For this reason the nFlags parameter is used to indicate the status of the callback when handling larger blocks of indirect data. The table below defines the supported flags:

Bit Position	Flag Name	Description
0	`ROM_CBFLAG_DIRECT`	When set indicates the call was from the processing of a block header with the `ROM_BFLAG_CALLBACK` flag set
1	`ROM_CBLAG_PAGESTART`	Indicates the callback was a result of a fetch of a page of data to the intermediate buffers
2	`ROM_CBFLAG_FIRST`	Set if the first fetch of payload data
3	`ROM_CBFLAG_FINAL`	Set if the final fetch of payload data
31:4	Reserved	Reserved

When a callback block header is received by the boot kernel a call to the callback is performed with the `ROM_CBFLAG_DIRECT` flag set. If the `ROM_BFLAG_INDIRECT` flag or the `ROM_BFLAG_PAGEMODE` flags are set indicating the use of indirect or page mode the `ROM_CBFLAG_FIRST` and `ROM_CBFLAG_FINAL` flags are cleared. If the transfer is a direct transfer straight to the final destination and not via the intermediate buffers then the `ROM_CBFLAG_FIRST` and `ROM_CBFLAG_FINAL` flags are also set.

This allows software to identify a callback call based on the processing of a block header with the `ROM_BFLAG_CALLBACK` flag set.

In addition to callbacks being performed on processing of the block header they are also called when processing payloads indirectly or when page mode is enabled. When the callback is called as a result of processing the payload data via the intermediate buffers `ROM_CBFLAG_DIRECT` is cleared. If the callback is being called as a result of fetching the first block of data in the payload the `ROM_CBFLAG_FIRST` flag is set. If the complete block of data fits in the intermediate buffer is also set. If the payload does not fit completely in the intermediate buffers multiple fetches must take place and thus multiple callbacks generated. If no flags are set it indicates a callback on a payload transfer and it is neither the first nor the last block of data in the payload, so there is still further data in the payload to be fetched. If only `ROM_CBFLAG_FINAL` is set then it is the final block in a payload transfer.

The following table provides an overview of the flag states and their meaning for the processing of callbacks.

ROM_CBFLAG_ DIRECT	**ROM_CBFLAG_ PAGESTART**	**ROM_CBFLAG_ FIRST**	**ROM_CBFLAG_ FINAL**	Description
1	0	0	0	Callback as a result of processing a block header with indirect or pagemode enabled
1	0	1	1	Callback as a result of processing a block header with indirect and pagemode disabled
0	1	0	0	Callback as a result of fetching a page of data in pagemode

ROM_CBFLAG_ DIRECT	ROM_CBFLAG_ PAGESTART	ROM_CBFLAG_ FIRST	ROM_CBFLAG_ FINAL	Description
0	1	0	1	Callback as a result of fetching a page of data in pagemode and the final page in the block
0	0	1	0	Callback as a result of fetching the first part of payload in an indirect payload
0	0	0	0	Callback as a result of fetching an indirect payload, not first or last transfer in payload
0	0	0	1	Callback as a result of fetching the final part of payload in an indirect payload
0	0	1	1	Callback as a result of fetching the complete payload in an indirect payload

initcode()

Initcode function for implementing custom callbacks to previously loaded code during boot.

API Details

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the `ADI_ROM_BOOT_CONFIG` object containing the complete context of the boot procedure

Function Description

Initcode functions can be embedded into the boot stream to allow for execution of user defined code during the boot phase. This is typically used to allow for optimal configuration of the CGU or any external memory interfaces that may be required to be initialized in order to be able to boot data to those memories.

A boot stream may have any number of initcodes present. The only requirement is that the code to be executed must be loaded prior to the `BFLAG_INIT` block being processed.

The initcode routine is passed the pointer to the complete boot context allowing for initcodes to provide extensive boot customization tasks is so desired.

Data Structures

The programming model for booting the processor uses the data structures defined in this section.

struct ADI_ROM_BOOT_BUFFER

Structure Type Declaration: `ADI_ROM_BOOT_BUFFER`

Boot Buffer.

A basic buffer type consisting of a pointer to the buffer and its size

Table 53-43: ADI_ROM_BOOT_BUFFER Members

Type	Name	Description
`void *`	`pBuffer`	Pointer to the buffer
`int32_t`	`dByteCount`	Size of the buffer

pBuffer

Pointer to the buffer

dByteCount

Size of the buffer

struct ADI_ROM_BOOT_CONFIG

Structure Type Declaration: `ADI_ROM_BOOT_CONFIG`

The Boot Configuration Object that contains all context for the boot process.

This structure contains the complete context for the boot process. A pointer to this object is passed through many of the routines and is presented to routines that are expected to be customized by users such as initcodes, custom initialization, configuration, load and cleanup routines. The object is passed to error handlers and callbacks giving the end user opportunity to significantly customize and adapt the boot process to their specific needs, especially in regards to multi-stage boot loader development.

Table 53-44: ADI_ROM_BOOT_CONFIG Members

Type	Name	Description
`void *`	`pSource`	Source address from where to fetch the next boot data.
`void *`	`pDestination`	Destination address to store the fetched data.
`int32_t`	`dByteCount`	Number of bytes to fetch from the boot source.
`int32_t`	`dFlags`	Control flags related to the boot kernel processing of blocks.
`uint32_t`	`ulBlockCount`	Limit of blocks to be processed during boot.
`uint32_t`	`ulBlockCurrent`	The number of blocks currently processed by the boot kernel
`void *`	`pNextDxe`	Pointer to the next application in the boot stream or the first free location after the boot stream.
`uint32_t`	`uByteAddress`	The destination address converted to the byte address space.
`uint32_t volatile *`	`pControlRegister`	Pointer to the boot peripherals control register.

Table 53-44: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
`int32_t`	`dControlValue`	Storage for the boot peripheral main control value to enable that peripheral in a required configuration.
`uint32_t volatile *`	`pPeripheralBase`	Pointer to the boot peripherals base MMR address
`uint32_t volatile *`	`pAuxControlRegister`	Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection
`uint32_t volatile *`	`pAuxPeripheralBase`	Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block
`uint32_t volatile *`	`pSecControlRegister`	Base MMR address of the SEC SSI instance associated with the boot peripheral should they be required for advanced second stage boot loader development
`ADI_DMA_TypeDef *`	`pDmaBaseRegister`	Base MMR address of the DMA channel associated with the boot peripheral.
`ROM_DMA_DONE_DETECT_METHOD`	`loadType`	Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA
`ROM_DMA_MDMA_CONFIG`	`MdmaCfg`	An MDMA descriptor that is used by the boot kernel for internal MDMA operations.
`uint16_t`	`uwDataWidth`	The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit
`uint16_t`	`uwSrcModifyMult`	The source modify multiplier used to set `DMA_XMOD` for source MDMA operations or peripheral DMA transmit operations
`uint16_t`	`uwDstModifyMult`	The destination modify multiplier used to set `DMA_XMOD` for destination MDMA operations or peripheral DMA receive operations
`uint16_t`	`uwUserShort`	Free to use by the user
`int32_t`	`dUserLong`	Free to use by the user
`int32_t`	`dReserved0`	Reserved for future use
`void *`	`pModeData`	Pointer to the boot mode specific data structure.
`int32_t`	`dBootCommand`	The boot command value supplied during the call to the `adi_rom_Boot()` routine
`ADI_ROM_BOOT_HEADER *`	`pHeader`	Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel
`void *`	`pTempBuffer`	Pointer to the internal intermediate buffer. Used for processing of indirect blocks
`void`	`dReserved1`	Reserved
`int32_t`	`dTempByteCount`	Size of the internal intermediate buffer in bytes
`void *`	`pTempSource`	Current source address that is being processed in the internal intermediate buffer

Table 53-44: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
int32_t	dPageByteCount	The page size to be used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests
ADI_ROM_BOOT_INTER_BUFFERS	bootBuffers	The internal intermediate buffer descriptors required when using indirect and page mode features
ROM_BOOT_REGISTRY	bootRegistry	The registry object that is used to register a boot peripherals routines with the kernel.
ROM_BOOT_ERROR_FUNC *	pErrorFunction	Pointer to the error handler to be called in the event of an error
ROM_BOOT_CALLBACK_FUNC *	pCallBackFunction	Pointer to the callback function that is called when processing boot blocks with the callback flag set
ROM_BOOT_CALLBACK_FUNC *	pCrcFunction	Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data
ROM_BOOT_CALLBACK_FUNC *	pForwardFunction	Feature not supported on this product
ADI_ROM_BOOT_MODES	bootModes	Access to all boot mode specific resources
void *	pLogBuffer	Pointer to the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
void *	pLogCurrent	The current position within the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
int32_t	dLogByteCount	The size of the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
ADI_ROM_OTP_BOOT_INFO *	pOtpBootInfo	Pointer to the ADI_ROM_OTP_BOOT_INFO boot information block that gets read from OTP and contains boot customization options
ADI_ROM_BOOT_KEY_TYPE	keyType	When set to a specific value allows keys not stored in OTP to be used for secure boot evaluation on an open processor.
ADI_ROM_BOOT_TYPE	bootType	A key to indicate if the boot type is secure or non-secure for open parts
ROM_SB_IMAGE_TYPE	secureBootImageType	The type of secure boot image
SBIF_ECDSA_Header_t *	pSecureHeader	Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase
ADI_SBIF_ECDSA_PublicKey_t	publicKey	The public key used for secure boot image authentication
CRYPTO_DESCRIPTORS	cryptoDescriptors	The descriptor items as required for the PKTE operations
SB_StorageArea_t *	pSB_Storage	Storage area reserved for some crypto operations
int32_t	secureBytesRemaining	The number of bytes remaining to be processed in the secure boot stream
uint32_t[4]	aesKey	The 128-bit AES decryption key

Table 53-44: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
`uint32_t[6]`	`aesWrapKey`	The key wrapped key from the `BLw` secure boot image
`uint32_t[4]`	`IV`	The `IV` as read from the secure boot header as required to initialize the PKTE
`uint8_t *`	`pHash`	Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream
`uint32_t`	`errorReturn`	Storage location for the address of the instruction line following a call to the error handler

pSource

Source address from where to fetch the next boot data.

The source address must be maintained by the boot peripherals load function. The kernel does not update the source pointer automatically after requesting data. This allows for load routines to control and change the source address, especially useful for advanced second stage loaders if they have a requirement to change the source address due to a fragmented boot stream or to reset the address if expanding into a second SPI flash device.

During debug it is useful in identifying the block in the boot stream that is currently being processed.

pDestination

Destination address to store the fetched data.

Used by the boot kernel to indicate the destination address for the data to be fetched. Boot peripherals load function must transfer the data to this location before returning back to the kernel. The boot kernel updates this field depending on whether a block header or payload is being fetched. In normal mode of operation the kernel will load this field with the location of the storage area to store a block header then after processing the block header loads it with the `ADI_ROM_BOOT_HEADER::pTargetAddress` contents red from the fetched block header. When using page mode the destination points to the internal buffers and is then updated for transferring data to the final destination.

dByteCount

Number of bytes to fetch from the boot source.

The kernel sets this parameter to indicate to he load function the number of bytes the kernel is requesting. The kernel is responsible for adjusting the byte count for page mode based accesses. The peripherals load function must return the required number of bytes to the destination address provided.

dFlags

Control flags related to the boot kernel processing of blocks.

When calling a boot mode via the `adi_rom_Boot()` routine the `flags` supplied to that routine are used to initialize this item. These become global flags that remain set through the entire boot process. When a block header is received the lower 16-bits of the block header are OR'ed with the global flags. The boot software may clear some flags if it detects some flags are not compatible with some others and then writes the resulting flags back to this member. The boot kernel then processes the block payload as instructed by the combination of global and boot block specific flags. Upon completion of the block processing original set of global flags are restored and the process repeated.

ulBlockCount

Limit of blocks to be processed during boot.

When calling the boot process the `adi_rom_Boot()` routine can accept a block limit for the number of blocks to process before terminating the boot process. If the block count is set to zero then the boot process will continue until a final block reached indicating end of the boot stream. This member holds the user specified limit for the number of blocks to be processed and is used by the boot kernel after processing of each block and compares it against the `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` value. Boot process terminates when `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` equals `ADI_ROM_BOOT_CONFIG::ulBlockCount`

ulBlockCurrent

The number of blocks currently processed by the boot kernel

pNextDxe

Pointer to the next application in the boot stream or the first free location after the boot stream.

This member is initialized when processing a first block in the boot stream. The `ADI_ROM_BOOT_HEADER::dArgument` field of a first block contains the number of bytes left in the boot stream before we reach the end of that boot stream. This allows for the this pointer to be used to point to the next boot stream or to the first empty location after the boot stream. This allows for a feature when using the `adi_rom_Boot()` routine to find the address of an application in a linked list of boot streams or to find the first empty location after the boot stream.

uByteAddress

The destination address converted to the byte address space.

This member is used to store the byte address space equivalent of SHARC L1 memory addresses, allowing any core to load content to any SHARC cores L1 memory. The SHARC core in which the load is targeted is determined by the block header.

pControlRegister

Pointer to the boot peripherals control register.

This can be used by a boot mode peripherals driver in order to gain efficient access to a control MMR register in the boot peripheral.

NOTE: This is not used in this products boot implementation but may be leveraged by developers of second stage boot loaders if required

dControlValue

Storage for the boot peripheral main control value to enable that peripheral in a required configuration.

This can be used by a boot modes peripheral driver in order to store a control value that can be used to enable the peripheral for a required configuration.

pPeripheralBase

Pointer to the boot peripherals base MMR address

pAuxControlRegister

Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection

pAuxPeripheralBase

Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block

pSecControlRegister

Base MMR address of the SEC SSI instance associated with the boot peripheral should they be required for advanced second stage boot loader development

pDmaBaseRegister

Base MMR address of the DMA channel associated with the boot peripheral.

This is used by the boot kernel to gain access to the DMA channels status when using non blocking DMA operations when page mode or secure boot is required, so it must be set when implementing custom boot loaders in order for the kernel to get access to that peripherals DMA status.

NOTE: If a custom boot peripheral does not support the standard DMA instance then the custom driver will be required to set up a DMA instance in SRAM that this location points to and the load function would need to update the status accordingly to indicate when the DMA was running and when the DMA completed.

loadType

Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA

MdmaCfg

An MDMA descriptor that is used by the boot kernel for internal MDMA operations.

The boot kernel may be required to perform internal MDMA operations outside the control of the boot peripheral driver. Such operations include processing of fill blocks CRC callbacks for CRC verification and MDMA operations from the internal intermediate buffers for indirect block and page mode processing.

NOTE: Users must not use this item or reconfigure this item when developing custom boot drivers, it is intended purely for the internal use by the boot kernel

uwDataWidth

The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit

uwSrcModifyMult

The source modify multiplier used to set `DMA_XMOD` for source MDMA operations or peripheral DMA transmit operations

uwDstModifyMult

The destination modify multiplier used to set `DMA_XMOD` for destination MDMA operations or peripheral DMA receive operations

uwUserShort

Free to use by the user

dUserLong

Free to use by the user

pModeData

Pointer to the boot mode specific data structure.

Can be set by a boot peripheral driver to allow for a single point of access to the boot mode specific object containing control and configuration information specific to that single boot mode.

dBootCommand

The boot command value supplied during the call to the `adi_rom_Boot()` routine

pHeader

Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel

pTempBuffer

Pointer to the internal intermediate buffer. Used for processing of indirect blocks

dTempByteCount

Size of the internal intermediate buffer in bytes

pTempSource

Current source address that is being processed in the internal intermediate buffer

dPageByteCount

The page size to be used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests

bootBuffers

The internal intermediate buffer descriptors required when using indirect and page mode features

bootRegistry

The registry object that is used to register a boot peripherals routines with the kernel.

When using the `adi_rom_Boot()` function the boot software calls a peripherals initialization, configuration, load and cleanup routines from the pointers stored in this object. The kernel itself only makes calls to the load function for the peripheral so when using the `adi_rom_BootKernel()` function the load function that is called by the boot kernel to fetch data from the boot source must be registered via `ADI_ROM_BOOT_REGISTRY::pLoadFunction`.

pErrorFunction

Pointer to the error handler to be called in the event of an error

pCallBackFunction

Pointer to the callback function that is called when processing boot blocks with the callback flag set

pCrcFunction

Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data

pForwardFunction

Feature not supported on this product

bootModes

Access to all boot mode specific resources

pLogBuffer

Pointer to the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pLogCurrent

The current position within the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

dLogByteCount

The size of the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pOtpBootInfo

Pointer to the `ADI_ROM_OTP_BOOT_INFO` boot information block that gets read from `OTP` and contains boot customization options

keyType

When set to a specific value allows keys not stored in `OTP` to be used for secure boot evaluation on an open processor.

By default when performing boot on an open part the decryption keys and the public key are fetched from `OTP`. By setting this field to `ADI_ROM_BOOT_KEY_TYPE::ADI_ROM_CUSTOM_SECURITY` users can disable the fetching of the keys from `OTP` and instead provision the keys directly in the `ADI_ROM_BOOT_CONFIG::publicKey` and `ADI_ROM_BOOT_CONFIG::aesKey` members via hook routines when using the `adi_rom_Boot()` function.

bootType

A key to indicate if the boot type is secure or non-secure for open parts

secureBootImageType

The type of secure boot image

pSecureHeader

Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase

publicKey

The public key used for secure boot image authentication

cryptoDescriptors

The descriptor items as required for the PKTE operations

pSB_Storage

Storage area reserved for some crypto operations

secureBytesRemaining

The number of bytes remaining to be processed in the secure boot stream

aesKey

The 128-bit AES decryption key

aesWrapKey

The key wrapped key from the `BLw` secure boot image

IV

The `IV` as read from the secure boot header as required to initialize the PKTE

pHash

Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream

errorReturn

Storage location for the address of the instruction line following a call to the error handler

struct ADI_ROM_BOOT_CUSTOM

Structure Type Declaration: `ADI_ROM_BOOT_CUSTOM`

A custom boot structure for storing information relating to a custom boot mode.

This structure is not used by the boot rom software at all but it is included in `ADI_ROM_BOOT_CONFIG::bootModes` in the event storage is required for custom boot loaders. Users can make use of this storage are to register DMA channels and define flags that may be used by a custom load routine.

Table 53-45: ADI_ROM_BOOT_CUSTOM Members

Type	Name	Description
`uint32_t`	`nFlags`	Flags related to the custom boot mode
`void *`	`pRegisters`	Used to store a pointer to the MMR registers for the peripheral
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the peripherals Tx DMA channel
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the peripherals Rx DMA channel
`ADI_SEC_Sysblock_TypeDef *`	`pSecSsi`	Pointer to the peripherals SEC SSI block
`void *`	`pModeData`	Void pointer that can be used to access additional boot mode specific resources not originally accessible via this data object
`ADI_ROM_BOOT_REGISTRY`	`registry`	For storage of the custom boot mode registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry` to register custom functions

nFlags

Flags related to the custom boot mode

pRegisters

Used to store a pointer to the MMR registers for the peripheral

pRxDmaRegisters

Pointer to the peripherals Tx DMA channel

pTxDmaRegisters

Pointer to the peripherals Rx DMA channel

pSecSsi

Pointer to the peripherals SEC SSI block

pModeData

Void pointer that can be used to access additional boot mode specific resources not originally accessible via this data object

registry

For storage of the custom boot mode registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry` to register custom functions

struct ADI_ROM_BOOT_HEADER

Structure Type Declaration: `ADI_ROM_BOOT_HEADER`

Boot Block Header.

Boot block headers control the loading process of the boot stream, For full details on the contents of the block header and supported flags see Boot Loader Stream .

Table 53-46: ADI_ROM_BOOT_HEADER Members

Type	Name	Description
`int32_t`	`dBlockCode`	Instructs the boot kernel how to process the block.
`void *`	`pTargetAddress`	Destination Address of Payload
`int32_t`	`dByteCount`	Byte Count of the Payload
`int32_t`	`dArgument`	Argument functionality varies depending on operation

dBlockCode

Instructs the boot kernel how to process the block.

Contains a number of fields for verification of the block header and flags to indicate the type of block allowing the kernel to process the block accordingly.

pTargetAddress

Destination Address of Payload

dByteCount

Byte Count of the Payload

dArgument

Argument functionality varies depending on operation

struct ADI_ROM_BOOT_INTER_BUFFER

Structure Type Declaration: `ADI_ROM_BOOT_INTER_BUFFER`

The buffer object for the internal intermediate buffers used for indirect and page mode operations.

Table 53-47: ADI_ROM_BOOT_INTER_BUFFER Members

Type	Name	Description
`uint8_t *`	`pBuffer`	Pointer to the buffer
`uint32_t`	`size`	Size of the buffer
`uint32_t`	`pageSize`	Page size for block based devices.

pBuffer

Pointer to the buffer

size

Size of the buffer

pageSize

Page size for block based devices.

NOTE: On this product this field is not used. A fixed page size of 1024 bytes is used in this product.

struct ADI_ROM_BOOT_INTER_BUFFERS

Structure Type Declaration: `ADI_ROM_BOOT_INTER_BUFFERS`

The boot kernels internal buffer object used to access the intermediate buffers and obtain buffer status.

Table 53-48: ADI_ROM_BOOT_INTER_BUFFERS Members

Type	Name	Description
`ADI_ROM_BOOT_INTER_BUFFER[2]`	`buffer`	The two buffer descriptors
`ADI_ROM_BOOT_BUFFER_STATE`	`state`	Buffer Status Information
`void *`	`pSource`	Original source address pointer of data loaded to active buffer. Not used on this product
`ADI_DMA_TypeDef *`	`pDma`	

buffer

The two buffer descriptors

state

Buffer Status Information

pSource

Original source address pointer of data loaded to active buffer. Not used on this product

struct ADI_ROM_BOOT_LINKPORT

Structure Type Declaration: `ADI_ROM_BOOT_LINKPORT`

The LINKPORT Slave Boot Mode Specific Structure.

This structure contains all the boot context information that is specific to the use for the LINKPORT slave boot mode.

Table 53-49: ADI_ROM_BOOT_LINKPORT Members

Type	Name	Description
`uint32_t`	`nFlags`	Flags related to linkport boot mode
`ADI_LP_TypeDef *`	`pRegisters`	Pointer to the LP peripherals base MMR address, not used on this product
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product
`ADI_SEC_Sysblock_TypeDef *`	`pSecSsi`	Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product
`ADI_ROM_BOOT_REGISTRY`	`registry`	For storage of the LP Slave boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

nFlags

Flags related to linkport boot mode

pRegisters

Pointer to the LP peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

pSecSsi

Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product

registry

For storage of the LP Slave boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

struct ADI_ROM_BOOT_MODES

Structure Type Declaration: `ADI_ROM_BOOT_MODES`

Holds all boot mode specific configuration items.

A boot mode may have requirements for some dedicated storage. This object is used to collect together all those storage items for all the boot modes supported by the boot rom.

Table 53-50: ADI_ROM_BOOT_MODES Members

Type	Name	Description
`ADI_ROM_BOOT_SPI`	`spi`	Access to all SPI boot mode resources
`ADI_ROM_BOOT_UART`	`uart`	Access to all UART boot mode resources
`ADI_ROM_BOOT_LINKPORT`	`linkport`	Access to all LINKPORT boot mode resources
`ADI_ROM_BOOT_CUSTOM`	`custom`	Access to all custom boot mode resources

spi

Access to all SPI boot mode resources

uart

Access to all UART boot mode resources

linkport

Access to all LINKPORT boot mode resources

custom

Access to all custom boot mode resources

struct ADI_ROM_BOOT_REGISTRY

Structure Type Declaration: `ADI_ROM_BOOT_REGISTRY`

Boot Mode Registration.

Used to hold pointers for the boot modes initialization, configuration, load and cleanup functions. By using pointer users can customize the registered content via hook routines or install their own load functions or cleanup functions from within init codes.

When using `adi_rom_Boot()` the boot process will make a call to the initialization function and the configuration function before calling the kernel. The kernel then runs making calls to the load function. Upon reaching the end of the boot stream the cleanup function is then called.

When using the `adi_rom_BootKernel()` function only the load function is called during execution of the software in the boot rom. All the functions here must return a `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` result in order for the boot process to continue. All functions expect a single argument that is the pointer to the boot structure object `ADI_ROM_BOOT_CONFIG`.

Table 53-51: ADI_ROM_BOOT_REGISTRY Members

Type	Name	Description
`ROM_BOOT_MODE_INIT_FUNC *`	`pInitFunction`	Pointer to the boot modes Initialization function
`ROM_BOOT_MODE_CONFIG_FUNC *`	`pConfigFunction`	Pointer to the boot modes Configuration function
`ROM_BOOT_MODE_LOAD_FUNC *`	`pLoadFunction`	Pointer to the boot modes Load function
`ROM_BOOT_MODE_CLEANUP_FUNC *`	`pCleanUpFunction`	Pointer to the boot modes Cleanup function
`void *`	`pReserved`	Reserved for future use
`int32_t`	`dReserved`	Reserved for future use

pInitFunction

Pointer to the boot modes Initialization function

pConfigFunction

Pointer to the boot modes Configuration function

pLoadFunction

Pointer to the boot modes Load function

pCleanUpFunction

Pointer to the boot modes Cleanup function

struct ADI_ROM_BOOT_SPI

Structure Type Declaration: `ADI_ROM_BOOT_SPI`

The SPI Master Boot Mode Specific Structure.

This structure contains all the boot context information that is specific to the use for the SPI master boot mode. During auto-detection information is copied from the required ::ROM_SPI_LUTENTRY item into this structure and used to configure the SPI peripheral for the mode of operation.

Table 53-52: ADI_ROM_BOOT_SPI Members

Type	Name	Description
`uint8_t`	`ubReadCommand`	Read command to use to read data from the SPI device
`uint8_t`	`ubDummyBytes`	Number of dummy bytes to issue after the read command
`uint8_t`	`ubAddressBytes`	Number of address bytes required to access the device
`uint8_t`	`ubDataBits`	The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad
`uint16_t`	`uwClkLower`	The SPI clock divider value to be used
`uint16_t`	`uReserved0`	Reserved
`uint32_t`	`nTxCtl`	The value to be written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles
`uint32_t`	`nRxCtl`	The value to be written to the `SPI_RXCTL` register that is used for all receive operations
`uint32_t`	`nCmdCtl`	The value to be written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash
`ROM_BOOT_SPIM_IO_ENABLE_FUNC *`	`pMIOEnFunction`	Pointer to the function used to enable quad mode on the SPI flash
`uint8_t`	`nDummy`	The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated
`uint8_t`	`nFlags`	Flags used for some additional SPI configuration processing.
`uint16_t`	`uReserved2`	Reserved
`void *`	`pXIPAddress`	The memory mapped SPI address to boot from
`ADI_SPI_TypeDef *`	`pRegisters`	Pointer to the SPI peripherals base MMR address, not used on this product
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product
`ADI_SEC_Sysblock_TypeDef *`	`pSecSsi`	Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product
`ADI_ROM_BOOT_REGISTRY`	`registry`	For storage of the SPI Master boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

ubReadCommand

Read command to use to read data from the SPI device

ubDummyBytes

Number of dummy bytes to issue after the read command

ubAddressBytes

Number of address bytes required to access the device

ubDataBits

The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad

uwClkLower

The SPI clock divider value to be used

nTxCtl

The value to be written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles

nRxCtl

The value to be written to the `SPI_RXCTL` register that is used for all receive operations

nCmdCtl

The value to be written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash

pMIOEnFunction

Pointer to the function used to enable quad mode on the SPI flash

nDummy

The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated

nFlags

Flags used for some additional SPI configuration processing.

The flags supported are defined as follows:

Bit Position	Name	Description
0	`ROM_SPI_FLAGS_CMDSKIP_EN`	When set will result in the configuration routine enabling command skip mode where the SPI will not issue a read command for read operations
1	`ROM_SPI_FLAGS_MULTICMD_EN`	Instructs the configuration routine to enable sending of command cycles over dual or quad bit bus

pXIPAddress

The memory mapped SPI address to boot from

pRegisters

Pointer to the SPI peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

pSecSsi

Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product

registry

For storage of the SPI Master boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

struct ADI_ROM_BOOT_UART

Structure Type Declaration: `ADI_ROM_BOOT_UART`

The UART Slave Boot Mode Specific Structure.

This structure contains all the boot context information that is specific to the use for the UART slave boot mode.

Table 53-53: ADI_ROM_BOOT_UART Members

Type	Name	Description
`uint32_t`	`nFlags`	Flags related to UART Boot mode
`ADI_UART_TypeDef *`	`pRegisters`	Pointer to the UART peripherals base MMR address, not used on this product
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product
`ADI_SEC_Sysblock_TypeDef *`	`pSecSsi`	Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product
`ADI_ROM_BOOT_REGISTRY`	`registry`	For storage of the UART Slave boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

nFlags

Flags related to UART Boot mode

pRegisters

Pointer to the UART peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

pSecSsi

Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product

registry

For storage of the UART Slave boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

struct ADI_ROM_OTP_BOOT_CFG

Structure Type Declaration: `ADI_ROM_OTP_BOOT_CFG`

The boot configuration object for storing further boot customization objects.

This is a 160-bit structure that is allocated to one contiguous region in the `OTP` memory array. The functionality provided allows for individual flags to be set to enable or disable specific features of the boot process. Each flag is allocated in a separate 16-bit word so that each flag can be set at different times and the ECC information will not impact the setting of another flag.

Table 53-54: ADI_ROM_OTP_BOOT_CFG Members

Type	Name	Description
`uint32_t`	`cacheDis:1` (bitfield)	Cache Disable.
`uint32_t`	`reserved0:15` (bitfield)	Reserved
`uint32_t`	`decryptOnlyEn:1` (bitfield)	Decrypt Only Enable.
`uint32_t`	`reserved1:15` (bitfield)	Reserved
`uint32_t`	`cacheDisInv:1` (bitfield)	Cache Disable Invalidate.
`uint32_t`	`reserved2:15` (bitfield)	Reserved
`uint32_t`	`decryptOnlyInv:1` (bitfield)	Decrypt Only Invalidate.
`uint32_t`	`reserved3:15` (bitfield)	Reserved
`uint32_t`	`pubkey0Inv:1` (bitfield)	Invalidate Public Key 0, use next public key for secure boot
`uint32_t`	`reserved4:15` (bitfield)	Reserved
`uint32_t`	`pubkey1Inv:1` (bitfield)	Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys
`uint32_t`	`reserved5:15` (bitfield)	Reserved

Table 53-54: ADI_ROM_OTP_BOOT_CFG Members (Continued)

Type	Name	Description
`uint32_t`	`privkey0Inv:1` (bitfield)	Invalidate Decryption Key 0, use next Decryption key for secure boot
`uint32_t`	`reserved6:15` (bitfield)	Reserved
`uint32_t`	`privkey1Inv:1` (bitfield)	Invalidate Decryption Key 1, use next Decryption key for secure boot
`uint32_t`	`reserved7:15` (bitfield)	Reserved
`uint32_t`	`privkey2Inv:1` (bitfield)	Invalidate Decryption Key 2, use next Decryption key for secure boot
`uint32_t`	`reserved8:15` (bitfield)	Reserved
`uint32_t`	`privkey3Inv:1` (bitfield)	Invalidate Decryption Key 3, once invalidated part will no longer be bootable
`uint32_t`	`reserved9:15` (bitfield)	Reserved
`uint32_t`	`dmcEn:1` (bitfield)	Enables configuration of the DMC from values in `OTP` stored in the format of the `ADI_ROM_OTP_DMC_CONFIG` object
`uint32_t`	`reserved10:15` (bitfield)	Reserved
`uint32_t`	`dmcInv:1` (bitfield)	Invalidates the DMC values in the `OTP` resulting is bypassing of DMC configuration
`uint32_t`	`reserved11:15` (bitfield)	Reserved

cacheDis

Cache Disable.

By default the cache is enabled by boot software during preboot. Setting this bit will prevent the boot software from enabling the cache which will result in slower boot performance but allows for the memory regions that are used for cache space to be bootable memory regions for larger application images.

decryptOnlyEn

Decrypt Only Enable.

By default secure boot images that are also encrypted are decrypted then authenticated before the application is executed by the core. Authentication is a time consuming task and in situations where boot time is critical and the security of the device is less important, bypassing the authentication stage is possible by setting this bit. This will compromise security and is not recommended for use without additional security measures being implemented to verify the integrity of the loader software.

cacheDisInv

Cache Disable Invalidate.

Invalidates the Cache Disable bit to allow the boot rom to enable the cache again during preboot

decryptOnlyInv

Decrypt Only Invalidate.

Invalidates the Decrypt Only bit to allow the boot rom to perform full decrypt and authentication of secure boot streams

pubkey0Inv

Invalidate Public Key 0, use next public key for secure boot

pubkey1Inv

Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys

privkey0Inv

Invalidate Decryption Key 0, use next Decryption key for secure boot

privkey1Inv

Invalidate Decryption Key 1, use next Decryption key for secure boot

privkey2Inv

Invalidate Decryption Key 2, use next Decryption key for secure boot

privkey3Inv

Invalidate Decryption Key 3, once invalidated part will no longer be bootable

dmcEn

Enables configuration of the DMC from values in `OTP` stored in the format of the `ADI_ROM_OTP_DMC_CONFIG` object

dmcInv

Invalidates the DMC values in the `OTP` resulting is bypassing of DMC configuration

struct ADI_ROM_OTP_BOOT_CGU_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_CGU_INFO`

The CGU configuration object located in `OTP` for configuration of the CGU by the boot software.

This is a 128-bit structure that is allocated to one contiguous region in the `OTP` memory array. The functionality provided allows the boot software to configure the CGU to allow for a more efficient boot process.

Table 53-55: ADI_ROM_OTP_BOOT_CGU_INFO Members

Type	Name	Description
`uint32_t`	`ctl_WEN:1` (bitfield)	Enable write to the `CGU_CTL` register
`uint32_t`	`div_WEN:1` (bitfield)	Enable write to the `CGU_DIV` register
`uint32_t`	`reserved0:1` (bitfield)	Reserved
`uint32_t`	`div_DSEL:5` (bitfield)	`CGU_DIV.DSEL` value
`uint32_t`	`div_CSEL:5` (bitfield)	`CGU_DIV.CSEL` value
`uint32_t`	`div_S0SEL:3` (bitfield)	`CGU_DIV.S0SEL` value
`uint32_t`	`div_SYSSEL:5` (bitfield)	`CGU_DIV.SYSSEL` value
`uint32_t`	`div_S1SEL:3` (bitfield)	`CGU_DIV.S1SEL` value
`uint32_t`	`div_OSEL:7` (bitfield)	`CGU_DIV.OSEL` value
`uint32_t`	`ctl_DF:1` (bitfield)	`CGU_CTL.DF` value
`uint32_t`	`ctl_MSEL:7` (bitfield)	`CGU_CTL.MSEL` value
`uint32_t`	`auto_disable:1` (bitfield)	disable polling on auto-alignment of clocks, NOT RECOMMENDED!
`uint32_t`	`clkoutsel_USBCLKSEL:6` (bitfield)	`CGU_CLKOUTSEL.USBCLKSEL` value
`uint32_t`	`clkoutsel_CLKOUTSEL:5` (bitfield)	`CGU_CLKOUTSEL.CLKOUTSEL` value
`uint32_t`	`clkoutsel_WEN:1` (bitfield)	Enable write to the `CGU_CLKOUTSEL` register
`uint32_t`	`oscwctl0_WEN:1` (bitfield)	Enable write to the `CGU_OSCWDCTL` instance 0 register
`uint32_t`	`oscwctl0_HODF:6` (bitfield)	`CGU_OSCWDCTL.HODF` value
`uint32_t`	`oscwctl0_HODEN:1` (bitfield)	`CGU_OSCWDCTL.HODEN` value
`uint32_t`	`oscwctl0_CNGEN:1` (bitfield)	`CGU_OSCWDCTL.CNGEN` value
`uint32_t`	`oscwctl0_BOUF:5` (bitfield)	`CGU_OSCWDCTL.BOUF` value
`uint32_t`	`oscwctl0_BOUEN:1` (bitfield)	`CGU_OSCWDCTL.BOUEN` value
`uint32_t`	`oscwctl0_FAULTEN:1` (bitfield)	`CGU_OSCWDCTL.FAULTEN` value
`uint32_t`	`oscwctl0_MONDIS:1` (bitfield)	`CGU_OSCWDCTL.MONDIS` value

Table 53-55: ADI_ROM_OTP_BOOT_CGU_INFO Members (Continued)

Type	Name	Description
`uint32_t`	`oscwctl0_ FAULTPINDIS:1` (bitfield)	`CGU_OSCWDCTL.FAULTPINDIS` value
`uint32_t`	`oscwctl1_WEN:1` (bitfield)	Enable write to the `CGU_OSCWDCTL` instance 0 register
`uint32_t`	`oscwctl1_HODF:6` (bitfield)	`CGU_OSCWDCTL.HODF` value
`uint32_t`	`oscwctl1_HODEN:1` (bitfield)	`CGU_OSCWDCTL.HODEN` value
`uint32_t`	`oscwctl1_CNGEN:1` (bitfield)	`CGU_OSCWDCTL.CNGEN` value
`uint32_t`	`oscwctl1_BOUF:5` (bitfield)	`CGU_OSCWDCTL.BOUF` value
`uint32_t`	`oscwctl1_BOUEN:1` (bitfield)	`CGU_OSCWDCTL.BOUEN` value
`uint32_t`	`oscwctl1_FAULTEN:1` (bitfield)	`CGU_OSCWDCTL.FAULTEN` value
`uint32_t`	`oscwctl1_MONDIS:1` (bitfield)	`CGU_OSCWDCTL.MONDIS` value
`uint32_t`	`oscwctl1_ FAULTPINDIS:1` (bitfield)	`CGU_OSCWDCTL.FAULTPINDIS` value
`uint32_t`	`reserved2:28` (bitfield)	Reserved

ctl_WEN

Enable write to the `CGU_CTL` register

div_WEN

Enable write to the `CGU_DIV` register

div_DSEL

`CGU_DIV.DSEL` value

div_CSEL

`CGU_DIV.CSEL` value

div_S0SEL

`CGU_DIV.S0SEL` value

div_SYSSEL

`CGU_DIV.SYSSEL` value

div_S1SEL

`CGU_DIV.S1SEL` value

div_OSEL

`CGU_DIV.OSEL` value

ctl_DF

`CGU_CTL.DF` value

ctl_MSEL

`CGU_CTL.MSEL` value

auto_disable

disable polling on auto-alignment of clocks, NOT RECOMMENDED!

clkoutsel_USBCLKSEL

`CGU_CLKOUTSEL.USBCLKSEL` value

clkoutsel_CLKOUTSEL

`CGU_CLKOUTSEL.CLKOUTSEL` value

clkoutsel_WEN

Enable write to the `CGU_CLKOUTSEL` register

oscwctl0_WEN

Enable write to the `CGU_OSCWDCTL` instance 0 register

oscwctl0_HODF

`CGU_OSCWDCTL.HODF` value

oscwctl0_HODEN

`CGU_OSCWDCTL.HODEN` value

oscwctl0_CNGEN

`CGU_OSCWDCTL.CNGEN` value

oscwctl0_BOUF

`CGU_OSCWDCTL.BOUF` value

oscwctl0_BOUEN

`CGU_OSCWDCTL.BOUEN` value

oscwctl0_FAULTEN

`CGU_OSCWDCTL.FAULTEN` value

oscwctl0_MONDIS

`CGU_OSCWDCTL.MONDIS` value

oscwctl0_FAULTPINDIS

`CGU_OSCWDCTL.FAULTPINDIS` value

oscwctl1_WEN

Enable write to the `CGU_OSCWDCTL` instance 0 register

oscwctl1_HODF

`CGU_OSCWDCTL.HODF` value

oscwctl1_HODEN

`CGU_OSCWDCTL.HODEN` value

oscwctl1_CNGEN

`CGU_OSCWDCTL.CNGEN` value

oscwctl1_BOUF

`CGU_OSCWDCTL.BOUF` value

oscwctl1_BOUEN

`CGU_OSCWDCTL.BOUEN` value

oscwctl1_FAULTEN

`CGU_OSCWDCTL.FAULTEN` value

oscwctl1_MONDIS

`CGU_OSCWDCTL.MONDIS` value

oscwctl1_FAULTPINDIS

`CGU_OSCWDCTL.FAULTPINDIS` value

struct ADI_ROM_OTP_BOOT_CMD_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_CMD_INFO`

The boot command object for storing a customized boot command for each boot mode within the `OTP` memory array.

This is a 128-bit structure that is allocated to one contiguous region in the `OTP` memory array. The functionality provided allows the boot rom software to pass a custom boot command to a specific boot mode changing the default boot behavior on startup. This can be used for example to change the default UART instance used for a UART boot operation.

Table 53-56: ADI_ROM_OTP_BOOT_CMD_INFO Members

Type	Name	Description
`uint32_t`	`spiMasterBootCmd`	SPI Master Boot Mode
`uint32_t`	`spiSlaveBootCmd`	SPI Slave Boot Mode
`uint32_t`	`lpBootCmd`	Link Port Slave Boot Mode
`uint32_t`	`uartBootCmd`	UART Slave Boot Mode

spiMasterBootCmd

SPI Master Boot Mode

spiSlaveBootCmd

SPI Slave Boot Mode

lpBootCmd

Link Port Slave Boot Mode

uartBootCmd

UART Slave Boot Mode

struct ADI_ROM_OTP_BOOT_INFO

Structure Type Declaration: `ADI_ROM_OTP_BOOT_INFO`

The 512-bit boot info object located in `OTP` for boot customization.

This is a 512 bit structure that is allocated to one contiguous region in the `OTP` memory array. The contented in `OTP` is stored in the format of this structure allowing boot to read the contents directly into this object.

Users can read this object using the `adi_rom_otp_get()` routine supplying the `OTPCMD::otpcmd_boot_info` enumeration

Table 53-57: ADI_ROM_OTP_BOOT_INFO Members

Type	Name	Description
`ADI_ROM_OTP_BOOT_CGU_INFO`	`cgu`	CGU Configuration information
`ADI_ROM_OTP_BOOT_CMD_INFO`	`bcmd`	Boot Command customization for each boot mode allowing for a change of default boot peripheral instance and configuration

Table 53-57: ADI_ROM_OTP_BOOT_INFO Members (Continued)

Type	Name	Description
ADI_ROM_OTP_BOOT_CFG	bcfg	Additional boot configuration flags for key invalidation and for enabling DMC configuration
uint32_t	reserved0	Reserved
uint16_t	reserved1	Reserved
uint16_t	otpReadTiming	Reserved, Must always be zero

cgu

CGU Configuration information

bcmd

Boot Command customization for each boot mode allowing for a change of default boot peripheral instance and configuration

bcfg

Additional boot configuration flags for key invalidation and for enabling DMC configuration

otpReadTiming

Reserved, Must always be zero

struct ADI_ROM_OTP_DMC_CONFIG

Structure Type Declaration: `ADI_ROM_OTP_DMC_CONFIG`

The 384-bit configuration object located in `OTP` for configuration of the DMC during preboot.

If the user wishes to make use of the memories connected to the DMC peripheral during boot without the use of init codes in the boot stream then the settings can be applied to this object in the `OTP` memory. During preboot the boot software reads the object from `OTP` and will configure the peripheral accordingly.

NOTE: When the device is open it advisable to avoid the use of `OTP` and instead use initcodes for DMC initialization as the initcode method is highly customizable. If users wish to lock the device to enable secure boot then in order to boot to memories interfaced to the DMC then the configuration must be provisioned to `OTP` as initcodes are not supported in secure boot.

Table 53-58: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
uint32_t	reserved0:10 (bitfield)	Reserved

Table 53-58: ADI_ROM_OTP_DMC_CONFIG Members (Continued)

Type	Name	Description
uint32_t	ulDDR_DLLCTL:12 (bitfield)	Contents of DMC_DLLCTL [11:0]
uint32_t	ulDDR_EMR2:8 (bitfield)	Contents of DMC_EMR2 [7:0]
uint32_t	reserved1:2 (bitfield)	Reserved
uint32_t	ulDDR_CFGCTL	Packed content of DMC_CTL Register, DMC_CFG registers.
uint32_t	ulDDR_MREMR1	Packed content of DMC_EMR1 Register, DMC_MR registers.
uint32_t	ulDDR_TR0	Content of DMC_TR0
uint32_t	ulDDR_TR1	Content of DMC_TR1
uint32_t	ulDDR_TR2	Content of DMC_TR2
uint32_t	ulDDR_PHYCTL0	Content of DMC_PHY_CTL0
uint32_t	ulDDR_PHYCTL145	Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers.
uint32_t	ulDDR_PHYCTL2	Content of DMC_PHY_CTL2
uint32_t	ulDDR_PHYCTL3	Content of DMC_PHY_CTL3
uint32_t	ulDDR_CAL_PADCTL0_PHY_STAT3_0	Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , and DMC_PHY_STAT3 registers.
uint32_t	ulDDR_CAL_PADCTL2	Content of DMC_CAL_PADCTL2

ulDDR_DLLCTL

Contents of DMC_DLLCTL [11:0]

ulDDR_EMR2

Contents of DMC_EMR2 [7:0]

ulDDR_CFGCTL

Packed content of DMC_CTL Register, DMC_CFG registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_CFGCTL	Register
[15:0]	DMC_CFG [15:0]
[31:16]	DMC_CTL [15:0]

ulDDR_MREMR1

Packed content of DMC_EMR1 Register, DMC_MR registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_MREMR1	Register
[15:0]	DMC_EMR1 [15:0]
[31:16]	DMC_MR [15:0]

ulDDR_TR0

Content of DMC_TR0

ulDDR_TR1

Content of DMC_TR1

ulDDR_TR2

Content of DMC_TR2

ulDDR_PHYCTL0

Content of DMC_PHY_CTL0

ulDDR_PHYCTL145

Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_PHYCTL145	Register
[7:0]	DMC_PHY_CTL5 [7:0]
[15:8]	DMC_PHY_CTL4 [7:0]
[31:16]	DMC_PHY_CTL1 [31:16]

ulDDR_PHYCTL2

Content of DMC_PHY_CTL2

ulDDR_PHYCTL3

Content of DMC_PHY_CTL3

ulDDR_CAL_PADCTL0_PHY_STAT3_0

Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , and DMC_PHY_STAT3 registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_CAL_PADCTL0_PHY_STAT3_0	Register
[8:0]	DMC_PHY_STAT0[8:0]
[11:9]	DMC_PHY_STAT3[2:0]
[31:12]	DMC_CAL_PADCTL0[31:12]

ulDDR_CAL_PADCTL2

Content of `DMC_CAL_PADCTL2`

struct ROM_BOOT_DMA_INSTANCE

Structure Type Declaration: `ROM_BOOT_DMA_INSTANCE`

DMA Channel Instance.

Specifies the base MMR address of the DMA channel as well as trigger and interrupt IDs

Table 53-59: ROM_BOOT_DMA_INSTANCE Members

Type	Name	Description
ADI_DMA_TypeDef *	pReg	Pointer to the base address of the DMA channel MMR registers
DMA_CHANn_TypeDef	eDmaChannelId	The actual DMA channel ID in the system
uint8_t	TriggerId	The trigger ID associated with the DMA channel
uint8_t	InterruptId	The interrupt ID associated with the DMA channel

pReg

Pointer to the base address of the DMA channel MMR registers

eDmaChannelId

The actual DMA channel ID in the system

TriggerId

The trigger ID associated with the DMA channel

InterruptId

The interrupt ID associated with the DMA channel

struct ROM_BOOT_MDMA

Structure Type Declaration: `ROM_BOOT_MDMA`

MDMA Channels.

Provides access to all the MDMA channels and CRC peripherals supported by the processor

Table 53-60: ROM_BOOT_MDMA Members

Type	Name	Description
`ROM_BOOT_MDMA_REGS[PARAM_SYS0_NUM_MDMA_STREAMS]`	`Stream`	Array of MDMA channel configurations supported by the processor

Stream

Array of MDMA channel configurations supported by the processor

struct ROM_BOOT_MDMA_REGS

Structure Type Declaration: `ROM_BOOT_MDMA_REGS`

MDMA Channel Registers.

Contains the Source and Destination MDMA channel instances for access to the MMRs and interrupt and trigger information. Information is also provided on the CRC support of the MDMA channel and access is provided to the corresponding CRC peripheral.

Table 53-61: ROM_BOOT_MDMA_REGS Members

Type	Name	Description
`ROM_BOOT_DMA_INSTANCE`	`Src`	The source DMA Channel in the MDMA pair
`ROM_BOOT_DMA_INSTANCE`	`Dst`	The destination DMA Channel in the MDMA pair
`ADI_CRC_TypeDef *`	`pCrc`	The base MMR address of the associated CRC peripheral if one exists
`ROM_BOOT_MDMA_CRC_SUPPORT`	`eCrcSupport`	Indicates if the MDMA channel supports CRC or not

Src

The source DMA Channel in the MDMA pair

Dst

The destination DMA Channel in the MDMA pair

pCrc

The base MMR address of the associated CRC peripheral if one exists

eCrcSupport

Indicates if the MDMA channel supports CRC or not

struct ROM_DMA_MDMA_CONFIG

Structure Type Declaration: `ROM_DMA_MDMA_CONFIG`

MDMA Configuration Object.

The user configurable structure for controlling the MDMA operation to be supplied to the `adi_rom_MemDma()` routine.

Table 53-62: ROM_DMA_MDMA_CONFIG Members

Type	Name	Description
`ROM_DMA_MDMA_OPERATION`	`eOperation`	Type of operation to perform
`ROM_DMA_MDMA_ID`	`eId`	MDMA Channel ID
`void *`	`pSource`	Source Pointer
`void *`	`pDestination`	Destination Pointer
`uint32_t`	`ByteCount`	Byte Count
`ROM_DMA_DONE_DETECT_METHOD`	`eDoneDetect`	DMA Done Detection Method
`uint32_t`	`CrcCtl`	`CRC_CTL` value when CRC operations are required
`uint32_t`	`FillVal`	Fill value for memory fill operations
`uint32_t`	`CrcPoly`	CRC Polynomial for CRC operations
`uint32_t`	`CrcCompare`	Value used for CRC compare operations or for a CRC32 result compare

eOperation

Type of operation to perform

eId

MDMA Channel ID

pSource

Source Pointer

pDestination

Destination Pointer

ByteCount

Byte Count

eDoneDetect

DMA Done Detection Method

CrcCtl

`CRC_CTL` value when CRC operations are required

FillVal

Fill value for memory fill operations

CrcPoly

CRC Polynomial for CRC operations

CrcCompare

Value used for CRC compare operations or for a CRC32 result compare

struct ROM_DMA_PDMA_CONFIG

Structure Type Declaration: `ROM_DMA_PDMA_CONFIG`

PDMA Configuration Object.

The user configurable structure for controlling the PDMA operation via the `adi_rom_PeriphDma()` function.

Table 53-63: ROM_DMA_PDMA_CONFIG Members

Type	Name	Description
`ROM_DMA_PDMA_OPERATION`	`eOperation`	Type of operation to perform
`ADI_DMA_TypeDef volatile *`	`pRegs`	Pointer to the base address of the DMA channel MMR registers
`uint16_t`	`dataWidth`	The maximum supported data width of the DMA channel. Used to configure the `DMA_CFG.PSIZE PSIZE` field in `DMA_CFG`
`uint16_t`	`dstModifyMult`	The modify multiplier to be applied, usually set to 1
`void *`	`pSource`	Source Pointer used for transmit operations
`void *`	`pDestination`	Destination Pointer used for receive operations
`uint32_t`	`byteCount`	Number of bytes to transfer
`ROM_DMA_DONE_DETECT_METHOD`	`eDoneDetect`	DMA Done Detection method to be used for the transfer
`uint32_t`	`Reserved`	Reserved

eOperation

Type of operation to perform

pRegs

Pointer to the base address of the DMA channel MMR registers

dataWidth

The maximum supported data width of the DMA channel. Used to configure the `DMA_CFG.PSIZE` `PSIZE` field in `DMA_CFG`

dstModifyMult

The modify multiplier to be applied, usually set to 1

pSource

Source Pointer used for transmit operations

pDestination

Destination Pointer used for receive operations

byteCount

Number of bytes to transfer

eDoneDetect

DMA Done Detection method to be used for the transfer

struct otp_data

Structure Type Declaration: `otp_data`

Container for accessing data to be written to `OTP` via the `adi_rom_otp_pgm()` routine.

Any pointers that are `NULL` will result in the object not being written. Any data values of 0 will not be written.

Table 53-64: otp_data Members

Type	Name	Description
`uint32_t(*) [ROM_OTP_SZ_huk]`	`huk`	Pointer to 256-bit Hardware Unique Key
`uint32_t(*) [ROM_OTP_SZ_DTCP_key_ecc]`	`DTCP_key_ecc`	Pointer to 1280-bit DTCP Key (ECC Parameters)
`uint32_t(*) [ROM_OTP_SZ_DTCP_key_cont]`	`DTCP_key_cont`	Pointer to 320-bit DTCP Key (constant for content key)
`uint32_t(*) [ROM_OTP_SZ_DTCP_key_dev]`	`DTCP_key_dev`	Pointer to 1024-bit DTCP Key (device specific keys)
`uint32_t(*) [ROM_OTP_SZ_pvt_128key0]`	`pvt_128key0`	Pointer to 128-bit AES Key

Table 53-64: otp_data Members (Continued)

Type	Name	Description
`uint32_t(*)[ROM_OTP_SZ_pvt_128key1]`	`pvt_128key1`	Pointer to 128-bit AES Key
`uint32_t(*)[ROM_OTP_SZ_pvt_128key2]`	`pvt_128key2`	Pointer to 128-bit AES Key
`uint32_t(*)[ROM_OTP_SZ_pvt_128key3]`	`pvt_128key3`	Pointer to 128-bit AES Key
`uint32_t(*)[ROM_OTP_SZ_ek]`	`ek`	Pointer to 256-bit endorsement key
`uint32_t(*)[ROM_OTP_SZ_secure_emu_key]`	`secure_emu_key`	Pointer to 128-bit Secure Debug Key
`uint32_t(*)[ROM_OTP_SZ_public_key0]`	`public_key0`	Pointer to 512-bit public key used for boot stream authentication
`uint32_t(*)[ROM_OTP_SZ_public_key1]`	`public_key1`	Pointer to 512-bit public key used for boot stream authentication
`uint32_t(*)[ROM_OTP_SZ_boot_info]`	`boot_info`	Pointer to 512-bit boot customization structure, see also `ADI_ROM_OTP_BOOT_INFO`
`uint8_t`	`antiroll_nv_cntr`	Anti-rollback counter to prevent loading of older firmware during secure boot.
`uint32_t(*)[ROM_OTP_SZ_gp1]`	`gp1`	Pointer to 512-bit General purpose user space
`uint32_t`	`bootModeDisable:8` (bitfield)	Boot mode disable for permanently disabling specific boot modes
`uint32_t(*)[ROM_OTP_SZ_preboot_ddr_cfg]`	`preboot_ddr_cfg`	Pointer to 384-bit DMC Configuration. See also `ADI_ROM_OTP_DMC_CONFIG`
`uint32_t(*)[ROM_OTP_SZ_stageID]`	`stageID`	Pointer to 64-bit staging ID

huk

Pointer to 256-bit Hardware Unique Key

DTCP_key_ecc

Pointer to 1280-bit DTCP Key (ECC Parameters)

DTCP_key_cont

Pointer to 320-bit DTCP Key (constant for content key)

DTCP_key_dev

Pointer to 1024-bit DTCP Key (device specific keys)

pvt_128key0

Pointer to 128-bit AES Key

pvt_128key1

Pointer to 128-bit AES Key

pvt_128key2

Pointer to 128-bit AES Key

pvt_128key3

Pointer to 128-bit AES Key

ek

Pointer to 256-bit endorsement key

secure_emu_key

Pointer to 128-bit Secure Debug Key

public_key0

Pointer to 512-bit public key used for boot stream authentication

public_key1

Pointer to 512-bit public key used for boot stream authentication

boot_info

Pointer to 512-bit boot customization structure, see also `ADI_ROM_OTP_BOOT_INFO`

antiroll_nv_cntr

Anti-rollback counter to prevent loading of older firmware during secure boot.

The counter supports values of 0 through 31. The counter feature is disabled as long as the counter is set initially to 0.

gp1

Pointer to 512-bit General purpose user space

bootModeDisable

Boot mode disable for permanently disabling specific boot modes

preboot_ddr_cfg

Pointer to 384-bit DMC Configuration. See also `ADI_ROM_OTP_DMC_CONFIG`

stageID

Pointer to 64-bit staging ID

Enumerations

The programming model for booting the processor uses the enumerations defined in this section.

enum ADI_ROM_BOOT_KEY_TYPE

Enumeration Type Declaration: `ADI_ROM_BOOT_KEY_TYPE`

Indicates if custom security keys are to be used for evaluation of secure boot.

By default the boot process will fetch all security keys from `OTP` for use during secure boot. Enabling custom security allows for the user to set their own keys in the `ADI_ROM_BOOT_CONFIG` item and not have them taken from `OTP`. Allowing evaluation of secure boot through the use of the `adi_rom_Boot()` function without provisioning keys in `OTP`.

Table 53-65: ADI_ROM_BOOT_KEY_TYPE Members

Enumerator	Description
`ADI_ROM_CUSTOM_SECURITY`	Enable use of custom security keys for authentication and decryption

ADI_ROM_CUSTOM_SECURITY

Enable use of custom security keys for authentication and decryption

enum ADI_ROM_BOOT_TYPE

Enumeration Type Declaration: `ADI_ROM_BOOT_TYPE`

Used to indicate to the boot kernel in an open processor if secure or non secure boot is required.

The boot kernel defaults to a secure boot unless the boot structure has been configured to indicate Non-Secure Boot

Table 53-66: ADI_ROM_BOOT_TYPE Members

Enumerator	Description
`ADI_ROM_SECURE_BOOT_DIS`	Non-Secure Boot
`ADI_ROM_SECURE_BOOT`	Secure Boot

ADI_ROM_SECURE_BOOT_DIS

Non-Secure Boot

ADI_ROM_SECURE_BOOT

Secure Boot

enum OTPCMD

Enumeration Type Declaration: `OTPCMD`

Commands required by the `adi_rom_otp_get()` routine to retrieve specific fields from the `OTP` memory.

Table 53-67: OTPCMD Members

Enumerator	Description
`otpcmd_reserved0`	Reserved
`otpcmd_huk`	Hardware Unique Key
`otpcmd_DTCP_key_ecc`	DTCP Key (ECC Parameters)
`otpcmd_DTCP_key_cont`	DTCP Key (constant for content key)
`otpcmd_DTCP_key_dev`	DTCP Key (device specific keys)
`otpcmd_pvt_128key0`	Customer Private AES Key0
`otpcmd_pvt_128key1`	Customer Private AES Key1
`otpcmd_pvt_128key2`	Customer Private AES Key2
`otpcmd_pvt_128key3`	Customer Private AES Key3
`otpcmd_ek`	Endorsement Key
`otpcmd_secure_emu_key`	Secure Emulation Key
`otpcmd_public_key0`	Customer Public Key0
`otpcmd_public_key1`	Customer Public Key1
`otpcmd_boot_info`	Customer Programmable Boot Information
`otpcmd_otpTiming`	OTP Read timing override
`otpcmd_antiroll_nv_cntr`	AntiRollback NV Counter
`otpcmd_gp1`	General Purpose 1
`otpcmd_bootModeDisable`	Boot Mode Disable Bits
`otpcmd_preboot_ddr_cfg`	User DMC configuration
`otpcmd_stageID`	StageID
`otpcmd_reserved1`	Reserved

otpcmd_reserved0

Reserved

otpcmd_huk

Hardware Unique Key

otpcmd_DTCP_key_ecc

DTCP Key (ECC Parameters)

otpcmd_DTCP_key_cont

DTCP Key (constant for content key)

otpcmd_DTCP_key_dev

DTCP Key (device specific keys)

otpcmd_pvt_128key0

Customer Private AES Key0

otpcmd_pvt_128key1

Customer Private AES Key1

otpcmd_pvt_128key2

Customer Private AES Key2

otpcmd_pvt_128key3

Customer Private AES Key3

otpcmd_ek

Endorsement Key

otpcmd_secure_emu_key

Secure Emulation Key

otpcmd_public_key0

Customer Public Key0

otpcmd_public_key1

Customer Public Key1

otpcmd_boot_info

Customer Programmable Boot Information

otpcmd_otpTiming

OTP Read timing override

otpcmd_antiroll_nv_cntr

AntiRollback NV Counter

otpcmd_gp1

General Purpose 1

otpcmd_bootModeDisable

Boot Mode Disable Bits

otpcmd_preboot_ddr_cfg

User DMC configuration

otpcmd_stageID

StageID

enum ROM_BOOT_MDMA_CRC_SUPPORT

Enumeration Type Declaration: `ROM_BOOT_MDMA_CRC_SUPPORT`

MDMA Channel CRC support.

Specifies whether the MDMA channel supports CRC operations or not

Table 53-68: ROM_BOOT_MDMA_CRC_SUPPORT Members

Enumerator	Description
`ROM_BOOT_DMA_CRC_SUPPORTED`	MDMA Channel supports CRC
`ROM_BOOT_DMA_CRC_NOT_SUPPORTED`	MDMA Channel does not support CRC

ROM_BOOT_DMA_CRC_SUPPORTED

MDMA Channel supports CRC

ROM_BOOT_DMA_CRC_NOT_SUPPORTED

MDMA Channel does not support CRC

enum ROM_BOOT_RESULT

Enumeration Type Declaration: `ROM_BOOT_RESULT`

General Boot ROM Return Types.

Used throughout the boot software to indicate various success and failure events that can occur during boot. General success and failure are used for most operation in which the user likely has no particular general interest in knowing further details.

Table 53-69: ROM_BOOT_RESULT Members

Enumerator	Description
ROM_BOOT_FAILURE	Failure.
ROM_BOOT_SUCCESS	Success.
ROM_BOOT_HDR_CHKSUM_ERR	Boot Stream Block Header Checksum Error.
ROM_BOOT_HDR_SIGN_ERR	Boot Stream Block Header Sign Failure.
ROM_BOOT_HDR_DEST_ERR	Boot Stream Block Payload Destination Error.
RESERVED0	Reserved.
RESERVED1	Reserved.
RESERVED2	Reserved.
RESERVED3	Reserved.
ROM_BOOT_CGU_WRITE_ERR	CGU Write Error
ROM_BOOT_DMA_SUCCESS	DMA operation was successful
ROM_BOOT_DMA_FAILURE	DMA Failure.
ROM_BOOT_DMA_ACTIVE	DMA Channel is Active
ROM_BOOT_DMA_CONFIG_ERR	DMA configuration error
ROM_BOOT_MDMA_ID_ERR	Illegal MDMA Channel ID.
ROM_BOOT_MDMA_OPERATION_ERR	Illegal MDMA operation Specified.
ROM_BOOT_MDMA_CONFIG_ERR	Memory DMA configuration error
ROM_BOOT_MDMA_SRC_ERR	MDMA Source Channel Configuration Error.
ROM_BOOT_MDMA_DST_ERR	MDMA Destination Channel Configuration Error.
ROM_BOOT_MDMA_DONE_DETECT_ERR	Memory DMA Completed with errors
ROM_BOOT_MDMA_SUCCESS	Memory DMA Completed successfully
ROM_BOOT_PDMA_CONFIG_ERR	Peripheral DMA configuration invalid
ROM_BOOT_PDMA_SUCCESS	Peripheral DMA completed successfully
ROM_BOOT_CRC_FAILURE	MDMA CRC32 Failure.
ROM_BOOT_CRC_COUNT_ERR	CRC Byte Count was not a multiple 4.
ROM_BOOT_CRC_SUPPORTED_ERR	CRC Not Supported Error.
ROM_BOOT_CRC_INITCODE_ERR	CRC32 Enable Failure During Boot.
ROM_BOOT_CRC_CALLBACK_ERR	Error in Execution of the CRC Callback.
ROM_BOOT_SB_IMAGE_VERSION_ERR	Secure Boot Header Version Error.
ROM_BOOT_SB_IMAGE_TYPE_ERR	Secure Boot Header Type Error.

Table 53-69: ROM_BOOT_RESULT Members (Continued)

Enumerator	Description
`ROM_BOOT_SB_CERT_COUNT_ERR`	Secure Boot Header Certificate Count Error.
`ROM_BOOT_SB_KEY_UNWRAP_ERR`	Decryption Key Unwrap Error.
`ROM_BOOT_ROLLBACK_ID_ERR`	Secure Boot Anti-Rollback Protection Error.
`ROM_BOOT_OTP_NVCNTR_READ_ERR`	Non-Volatile Counter Read Error.
`ROM_BOOT_OTP_NVCNTR_PGM_ERR`	Non-Volatile Counter Program Error.
`ROM_BOOT_WAKEUP_NO_CGU_INIT`	Wakeup Actions CGU Programming Status.
`ROM_BOOT_WAKEUP_NO_DMC_INIT`	Wakeup Actions DMC Programming Status.
`ROM_BMODE_SPI_SWITCH`	Need to boot from the alternate SPI port
`ROM_BMODE_ILLEGAL_DEVNUM`	Illegal Device Enumeration Error.
`ROM_BMODE_EXIT`	Boot Stream Block Payload Destination Error.
`ROM_BMODE_FAILURE`	Boot Mode Error.
`ROM_DMA_SUCCESS`	The API call succeeded.
`ROM_DMA_FAILURE`	The API call failed.

ROM_BOOT_FAILURE

Failure.

General failure can be used to indicate any general failure throughout the boot process

ROM_BOOT_SUCCESS

Success.

General success can be used to indicate any general functional success for an operation during the boot process.

NOTE: This must be the return result for a boot mode drivers initialization, configuration, load and cleanup routines when overriding their functionality in second stage boot loaders to use custom functions.

ROM_BOOT_HDR_CHKSUM_ERR

Boot Stream Block Header Checksum Error.

Indicates that the 8-bit XOR checksum of the 16-byte block header failed to generated the expected result.

ROM_BOOT_HDR_SIGN_ERR

Boot Stream Block Header Sign Failure.

The 0xAD block required as byte 4 of the boot block header was not found.

ROM_BOOT_HDR_DEST_ERR

Boot Stream Block Payload Destination Error.

The target address field of the block header indicates that the payload for the block is destined towards an address that is not supported. This would typically indicate an attempt to load data to the reserved 8KB region of memory reserved by the boot process as non-bootable.

ROM_BOOT_CGU_WRITE_ERR

CGU Write Error

Returned by the CGU Configuration routine if a CGU error is set during the initialization of the CGU from settings provisioned in the OTP.

ROM_BOOT_DMA_SUCCESS

DMA operation was successful

ROM_BOOT_DMA_FAILURE

DMA Failure.

Returned by the DMA routines if an error was detected in the `DMA_STAT.IRQERR` prior to setting up a new DMA operation with the newly supplied configuration.

ROM_BOOT_DMA_ACTIVE

DMA Channel is Active

Returned only by the peripheral DMA routine when an attempt to run another peripheral DMA operation is attempted and the DMA channel is already running.

NOTE: This is not currently implemented for MDMA operations.

ROM_BOOT_DMA_CONFIG_ERR

DMA configuration error

ROM_BOOT_MDMA_ID_ERR

Illegal MDMA Channel ID.

Returned by `adi_rom_MemDma()` if the MDMA channel ID is not supported. For supported channel IDs, please refer to `ROM_DMA_MDMA_ID`

ROM_BOOT_MDMA_OPERATION_ERR

Illegal MDMA operation Specified.

Returned by `adi_rom_MemDma()` if the MDMA operation to be performed is not supported. For supported operations, please refer to `ROM_DMA_MDMA_OPERATION`

ROM_BOOT_MDMA_CONFIG_ERR

Memory DMA configuration error

ROM_BOOT_MDMA_SRC_ERR

MDMA Source Channel Configuration Error.

Set by the MDMA routines if after configuring the MDMA source channel to start a DMA operation, an error is generated in the source channels `DMA_STAT.IRQERR`

ROM_BOOT_MDMA_DST_ERR

MDMA Destination Channel Configuration Error.

Set by the MDMA routines if after configuring the MDMA source channel to start a DMA operation, an error is generated in the destination channels `DMA_STAT.IRQERR`.

ROM_BOOT_MDMA_DONE_DETECT_ERR

Memory DMA Completed with errors

ROM_BOOT_MDMA_SUCCESS

Memory DMA Completed successfully

ROM_BOOT_PDMA_CONFIG_ERR

Peripheral DMA configuration invalid

ROM_BOOT_PDMA_SUCCESS

Peripheral DMA completed successfully

ROM_BOOT_CRC_FAILURE

MDMA CRC32 Failure.

Returned by the higher level `adi_rom_MemDma()` routine and the underlying `adi_rom_MemCompare()` routine if the CRC32 result of the block of data did not match the expected result.

ROM_BOOT_CRC_COUNT_ERR

CRC Byte Count was not a multiple 4.

The CRC peripheral operates on 32-bit data only and as such all CRC operations must have a byte count that is a multiple of 4. This result is returned by the higher level `adi_rom_MemDma()` routine and the underlying `adi_rom_MemCompare()` and `adi_rom_MemFill()` routines if the byte count is not a multiple of 4 bytes.

ROM_BOOT_CRC_SUPPORTED_ERR

CRC Not Supported Error.

Returned by `adi_rom_MemDma()`, `adi_rom_MemFill()`, `adi_rom_MemCompare()` and `adi_rom_Crc32Poly()` if the supplied DMA configuration specified an MDMA channel that does not support CRC operations.

ROM_BOOT_CRC_INITCODE_ERR

CRC32 Enable Failure During Boot.

Returned by `adi_rom_Crc32Init()` if the boot process cannot enable the CRC32 functionality due to a `NULL` `ADI_ROM_BOOT_CONFIG` pointer or `NULL` `ADI_ROM_BOOT_HEADER` pointer located in `ADI_ROM_BOOT_CONFIG::pHeader`

ROM_BOOT_CRC_CALLBACK_ERR

Error in Execution of the CRC Callback.

Returned by the default CRC callback function located in the boot rom if any of the following conditions are met:

- The `ADI_ROM_BOOT_CONFIG` pointer passed to the callback is a `NULL` pointer
- The `ADI_ROM_BOOT_BUFFER` pointer pointing to the buffer to run CRC validation on is a `NULL` pointer
- The `ROM_CBFLAG_DIRECT` flag is not set in the supplied flags parameter indicating it was a direct callback
- The `ADI_ROM_BOOT_HEADER` pointer located in `ADI_ROM_BOOT_CONFIG::pHeader` is a `NULL` pointer

ROM_BOOT_SB_IMAGE_VERSION_ERR

Secure Boot Header Version Error.

Returned by the routine responsible for verifying the secure boot header if the version field is not the version supported by the processor. The version field is automatically set by the `signtool` utility.

ROM_BOOT_SB_IMAGE_TYPE_ERR

Secure Boot Header Type Error.

Returned by the routine responsible for verifying the secure boot header if the secure boot image type field is not one of the types supported by the processor. The type field is used to indicate if the image is a `BLp`, `BLx`, or `BLw` image secure boot image.

ROM_BOOT_SB_CERT_COUNT_ERR

Secure Boot Header Certificate Count Error.

Returned by the routine responsible for verifying the secure boot header if the number of certificate count is greater than 0 as the processor does not support the use of certificates in the secure boot implementation.

ROM_BOOT_SB_KEY_UNWRAP_ERR

Decryption Key Unwrap Error.

Indicates failure to unwrap the decryption key in `BLw` secure boot images.

ROM_BOOT_ROLLBACK_ID_ERR

Secure Boot Anti-Rollback Protection Error.

Indicates an attempt to boot a secure boot image with a lower firmware version than is currently stored in `OTP` or if the rollback ID read from `OTP` is out of bounds of the supported firmware update limit.

ROM_BOOT_OTP_NVCNTR_READ_ERR

Non-Volatile Counter Read Error.

A failure occurred in the reading of the non-volatile counter in `OTP` that contains the current Anti-Rollback protection value

ROM_BOOT_OTP_NVCNTR_PGM_ERR

Non-Volatile Counter Program Error.

A failure occurred in the programming of the new Anti-Rollback firmware version to the Non-Volatile Counter in `OTP`.

ROM_BOOT_WAKEUP_NO_CGU_INIT

Wakeup Actions CGU Programming Status.

Indicates that the routine responsible for programming the CGU during wakeup events did not perform any CGU configuration as there was no wakeup action request to do so.

ROM_BOOT_WAKEUP_NO_DMC_INIT

Wakeup Actions DMC Programming Status.

Indicates that the routine responsible for programming the DMC during wakeup events did not perform any DMC configuration as there was no wakeup action request to do so.

ROM_BMODE_SPI_SWITCH

Need to boot from the alternate SPI port

ROM_BMODE_ILLEGAL_DEVNUM

Illegal Device Enumeration Error.

Set when the boot process attempts to boot from a peripheral enumeration that is not supported or does not exist on the product. The peripheral enumeration is checked multiple times by all boot mode drivers in the ROM to ensure that peripheral instance to boot from is supported.

ROM_BMODE_EXIT

Boot Stream Block Payload Destination Error.

The target address field of the block header indicates that the payload for the block is destined towards an address that is not supported. This would typically indicate an attempt to load data to the reserved 8KB region of memory reserved by the boot process as non-bootable.

ROM_BMODE_FAILURE

Boot Mode Error.

A general error that can be returned by any of the boot mode drivers functions to indicate an error occurred. If an error occurs during the initialization, configuration or loading of boot data from the boot source then this error result may be used in the event a more concise error is is not available.

ROM_DMA_SUCCESS

The API call succeeded.

ROM_DMA_FAILURE

The API call failed.

enum ROM_CORE_ID

Enumeration Type Declaration: `ROM_CORE_ID`

Core ID.

An enumeration for referencing a particular core

Table 53-70: ROM_CORE_ID Members

Enumerator	Description
`ROM_CORE_ID0`	Core 0
`ROM_CORE_ID1`	Core 1
`ROM_CORE_ID2`	Core 2
`ROM_CORE_NUM_CORES`	Number of Cores

ROM_CORE_ID0

Core 0

ROM_CORE_ID1

Core 1

ROM_CORE_ID2

Core 2

ROM_CORE_NUM_CORES

Number of Cores

enum ROM_DMA_DONE_DETECT_METHOD

Enumeration Type Declaration: `ROM_DMA_DONE_DETECT_METHOD`

DMA Done Detection Method.

Specifies the method to be used for detecting the completion of the requested DMA operation.

When a user requests a non-blocking DMA operation then separate software is required to check the status of the DMA channel. The boot rom does not provide an API for use for this operation.

NOTE: The trigger mode is not supported on this product

Table 53-71: ROM_DMA_DONE_DETECT_METHOD Members

Enumerator	Description
`ROM_DMA_DONE_NON_BLOCKING`	Return without waiting for the DMA to complete
`ROM_DMA_DONE_POLL_IRQDONE`	Poll on the IRQDONE bit in the DMA Status register
`ROM_DMA_DONE_WAKEUP_TRIGGER`	Configure a trigger to wakeup the core on completion

ROM_DMA_DONE_NON_BLOCKING

Return without waiting for the DMA to complete

ROM_DMA_DONE_POLL_IRQDONE

Poll on the IRQDONE bit in the DMA Status register

ROM_DMA_DONE_WAKEUP_TRIGGER

Configure a trigger to wakeup the core on completion

enum ROM_DMA_MDMA_ID

Enumeration Type Declaration: `ROM_DMA_MDMA_ID`

MDMA Channel ID.

The ID of the Memory DMA channel to be used. This item is used in the `ROM_DMA_MDMA_CONFIG` configuration to specify the Memory DMA channel to use for operations accessible via the `adi_rom_MemDma()` routine

Table 53-72: ROM_DMA_MDMA_ID Members

Enumerator	Description
`ROM_DMA_MDMA0`	Memory DMA Stream 0
`ROM_DMA_MDMA1`	Memory DMA Stream 1
`ROM_DMA_MDMA2`	Memory DMA Stream 2
`ROM_DMA_MDMA3`	Memory DMA Stream 3

Table 53-72: ROM_DMA_MDMA_ID Members (Continued)

Enumerator	Description
`ROM_DMA_MEMDMA_END_COUNT`	Number of Memory DMA Streams

ROM_DMA_MDMA0

Memory DMA Stream 0

ROM_DMA_MDMA1

Memory DMA Stream 1

ROM_DMA_MDMA2

Memory DMA Stream 2

ROM_DMA_MDMA3

Memory DMA Stream 3

ROM_DMA_MEMDMA_END_COUNT

Number of Memory DMA Streams

enum ROM_DMA_MDMA_OPERATION

Enumeration Type Declaration: `ROM_DMA_MDMA_OPERATION`

MDMA Operation to be performed.

The operation determines if only an MDMA is required to be configured, or whether a CRC operation must be used in conjunction with the MDMA.

Table 53-73: ROM_DMA_MDMA_OPERATION Members

Enumerator	Description
`ROM_DMA_MEM_COPY`	Standard MDMA transfer from a source to a destination
`ROM_DMA_MEM_CRC`	Performs a CRC32 MDMA read operation and compares the result with an expected result
`ROM_DMA_MEM_FILL`	Uses the CRC peripheral to perform a fill operation with a 32-bit value
`ROM_DMA_MEM_COMPARE`	Uses the CRC peripheral to compare data with a constant 32-bit value
`ROM_DMA_CRC_LUT_INIT`	Initializes the CRC LUT from the supplied CRC Polynomial

ROM_DMA_MEM_COPY

Standard MDMA transfer from a source to a destination

ROM_DMA_MEM_CRC

Performs a CRC32 MDMA read operation and compares the result with an expected result

ROM_DMA_MEM_FILL

Uses the CRC peripheral to perform a fill operation with a 32-bit value

ROM_DMA_MEM_COMPARE

Uses the CRC peripheral to compare data with a constant 32-bit value

ROM_DMA_CRC_LUT_INIT

Initializes the CRC LUT from the supplied CRC Polynomial

enum ROM_DMA_PDMA_OPERATION

Enumeration Type Declaration: `ROM_DMA_PDMA_OPERATION`

Table 53-74: ROM_DMA_PDMA_OPERATION Members

Enumerator	Description
`ROM_DMA_PERI_TX`	Peripheral Transmit Operation
`ROM_DMA_PERI_RX`	Peripheral Receive Operation

ROM_DMA_PERI_TX

Peripheral Transmit Operation

ROM_DMA_PERI_RX

Peripheral Receive Operation

enum ROM_GETADDR_VALUE

Enumeration Type Declaration: `ROM_GETADDR_VALUE`

Parameter for `adi_rom_GetAddress()` function t oretireve the address of a data object stored in the boot rom.

Table 53-75: ROM_GETADDR_VALUE Members

Enumerator	Description
`ROM_GETADDR_CONSTANTS`	Retrieve the address of the `ROM_CONSTANTS_TYPE` object
`ROM_GETADDR_BMODE`	Retrieve the address of the lookup table sotring the default adi_rom_boot() parameters for each boot mode
`ROM_GETADDR_MDMAREGS`	Retrieve the address of the `ROM_BOOT_MDMA_REGS` object
`ROM_GETADDR_SPILUT`	Retrieve the address of the lookup table in the rom describing the various SPI master boot BCODE configurations
`ROM_GETADDR_ECDSA_DOMAIN`	Retrieve the address of the domain parameteres used for ECDSA

ROM_GETADDR_CONSTANTS

Retrieve the address of the `ROM_CONSTANTS_TYPE` object

ROM_GETADDR_BMODE

Retrieve the address of the lookup table sotring the default adi_rom_boot() parameters for each boot mode

ROM_GETADDR_MDMAREGS

Retrieve the address of the `ROM_BOOT_MDMA_REGS` object

ROM_GETADDR_SPILUT

Retrieve the address of the lookup table in the rom describing the various SPI master boot BCODE configurations

ROM_GETADDR_ECDSA_DOMAIN

Retrieve the address of the domain parameteres used for ECDSA

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

When calling a boot mode via adi_rom_Boot, the user may provide an optional hook routine as a callback. This hook routine is called by the boot software firstly after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 53-76: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
`ROM_HOOK_CALL_INIT_COMPLETE`	Call was as a result of completion of the boot modes initialization function
`ROM_HOOK_CALL_CONFIG_COMPLETE`	Call was as a result of the completion of the boot modes configuration function

ROM_HOOK_CALL_INIT_COMPLETE

Call was as a result of completion of the boot modes initialization function

ROM_HOOK_CALL_CONFIG_COMPLETE

Call was as a result of the completion of the boot modes configuration function

enum ROM_SB_IMAGE_TYPE

Enumeration Type Declaration: `ROM_SB_IMAGE_TYPE`

Secure Boot Image Types.

The secure boot header contains a type field for the secure boot image type, this enumeration provides a complete list of all image types.

NOTE: The secure boot process does not necessarily support all image types defined.

Table 53-77: ROM_SB_IMAGE_TYPE Members

Enumerator	Description
`ROM_SB_IMAGE_UNKNOWN`	Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place
`ROM_SB_IMAGE_BLP`	Plain text `BLp` secure boot image supporting authentication only with no decryption
`ROM_SB_IMAGE_BLW`	Keywrapped `BLw` secure boot image supporting authentication and decryption, boot stream decryption key wrapped in the secure header
`ROM_SB_IMAGE_BLE`	Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header
`ROM_SB_IMAGE_BLX`	`BLx` Secure boot image supporting authentication and decryption, boot stream decryption key wrapped located in `OTP`
`ROM_SB_IMAGE_UNSUPPORTED`	May be used by software to indicate any other unsupported image type

ROM_SB_IMAGE_UNKNOWN

Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place

ROM_SB_IMAGE_BLP

Plain text `BLp` secure boot image supporting authentication only with no decryption

ROM_SB_IMAGE_BLW

Keywrapped `BLw` secure boot image supporting authentication and decryption, boot stream decryption key wrapped in the secure header

ROM_SB_IMAGE_BLE

Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header

ROM_SB_IMAGE_BLX

`BLx` Secure boot image supporting authentication and decryption, boot stream decryption key wrapped located in `OTP`

ROM_SB_IMAGE_UNSUPPORTED

May be used by software to indicate any other unsupported image type

enum ROM_SPI_PROTOCOL

Enumeration Type Declaration: `ROM_SPI_PROTOCOL`

The SPI Protocol to use.

SPI Flash devices are now capable of supporting multiple protocols for the sending of the command to the SPI flash. Typically the command would be sent over the single bit bus, however a number of newer devices also support the sending of the command over the dual or quad bit bus.

WARNING: Enabling of DUALIO or QUADIO protocol for the command cycles runs the risk of the boot process being unable to communicate with the SPI flash, especially in system reset type events where the processor will attempt to reboot and the flash may not have been reset. It is not recommended to enable such features on SPI Flash devices if they are also the primary boot source used for booting from hardware reset and system reset events.

Table 53-78: ROM_SPI_PROTOCOL Members

Enumerator	Description
`ROM_SPI_EXT_PROTOCOL`	Extended protocol where the command cycle is sent on the single bit bus
`ROM_SPI_DUALIO_PROTOCOL`	DualIO protocol where the command cycle is sent on the dual bit bus
`ROM_SPI_QUADIO_PROTOCOL`	QuadIO protocol where the command cycle is sent on the quad bit bus

ROM_SPI_EXT_PROTOCOL

Extended protocol where the command cycle is sent on the single bit bus

ROM_SPI_DUALIO_PROTOCOL

DualIO protocol where the command cycle is sent on the dual bit bus

ROM_SPI_QUADIO_PROTOCOL

QuadIO protocol where the command cycle is sent on the quad bit bus

54 System Crossbars (SCB)

A modern system on a chip (SoCs) contains multi-cores, memory controllers, security blocks, and other high speed peripherals. As system integration increases, SoCs need to provide bus connectivity that allows better throughput to reduce performance bottlenecks. While traditional point-to-point connection buses have performed well in smaller systems, there is a need to use advanced switching based bus architectures for efficient handling of data transfer between multiplicity of data sources and sinks in the system. Additionally, mixing various traffic types in the same SoC (for example control, communication over peripherals and computing) while sharing the same bus resources, create different requirements from the Quality of Service (QoS) perspective.

The system crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB architecture addresses the challenges described above. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable quality of service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.

SCB Features

The SCBs provide the following features:

- Efficient, pipelined bus transfer protocol for sustained throughput
- Full-duplex bus operation for flexibility and reduced latency
- Concurrent bus transfer support to allow multiple bus masters to access bus slaves simultaneously
- Protection model (privileged or secure) support for selective bus interconnect protection
- Simple priority-based QoS based arbitration model
- Programmable quality of service

SCB Functional Description

The following sections provide a functional description of the SCB.

- SCB Architectural Concepts

ADSP-SC58x SCB0 Register List

This describes the System Crossbar fabric.

Table 54-1: ADSP-SC58x SCB0 Register List

Name	Description
SCB0_MST[n]_RQOS	Read Quality of Service for Master n
SCB0_MST[n]_WQOS	Write Quality of Service for Master n

ADSP-SC58x SCB1 Register List

System Crossbar for DMC Memory Space

Table 54-2: ADSP-SC58x SCB1 Register List

Name	Description
SCB1_MST00_SYNC	Mst00 Interface Block Sync Mode

ADSP-SC58x SCB3 Register List

System MMR Fabric Registers

Table 54-3: ADSP-SC58x SCB3 Register List

Name	Description
SCB3_DCLK0_WR_TIDE	DCLK0 Interface Block APB WR Tidemark
SCB3_MST00_SYNC	SYNC Mode

SCB Architectural Concepts

This section describes the components of the SCB and the modules connected to it. The basic elements in the SCB are SCB masters, slaves master interfaces, and slave interfaces.

Masters

The controllers in the system that raise the data request in the form of a read/write transaction on the SCB are called masters. The system bus masters include peripheral Direct Memory Access (DMA) channels. These include the Serial Port (SPORT) DMA, and SPI DMAs, among others. Also included are the Memory-to-Memory DMA channels (MDMA), the L1 code fill block, and the processor cores.

Slaves

Slaves are SCB connections that are responding to transfer requests. Slaves include MMR registers, memory units, and various peripherals depending upon individual configurations. Each system slave has its own latencies and response times.

SCB Block Diagrams

The SCB architectural model is illustrated in the ***SCB Overview*** figure. This figure shows a high level representation of a basic SCB connecting n Slaves to x Masters. A variable number of masters connect to a variable number of slaves in each SCB. In this example, all SIs connect to all MIs as indicated by the lines connecting them.

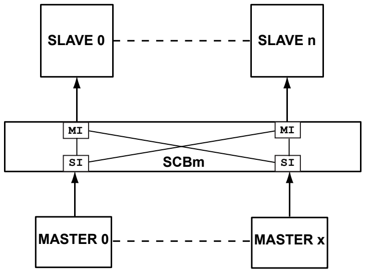


Figure 54-1: SCB Overview

Hierarchy Block Diagram

A system interconnect built from multiple SCBs in a hierarchical model is illustrated in the ***SCB Hierarchy Overview*** figure. The system master node level SCBs master connects multiple SIs to a single MI, which in turn connects to an SI of the system slave level node SCB.

As discussed above, all the masters in the system are distributed across different SCBs. A given SCB at system master node level connects directly to the system masters. These SCBs connect to SCB0 through its SIs forming a hierarchal structure. While a master has to access any slave, its first access goes through the SCB it is connected to, and then through SCB0, to reach its intended slave. This simplifies the connecting hardware in the basic SCB block by limiting the masters. Care must be taken when sharing masters to allow adequate throughput for their individual data transfer requirements.

In this example, all SIs are connected to all MIs.

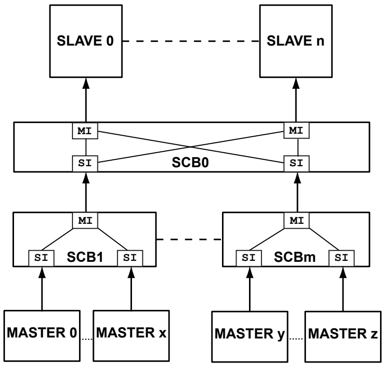


Figure 54-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the SCB Block Diagram.

SCB Block Diagram

The following figures show the SCB block diagram. For DMA channel assignments for the peripherals shown in the *SCB Block Diagram* figure, see the *DMA Channel Peripherals Controlled by SCBs* table following the figure.

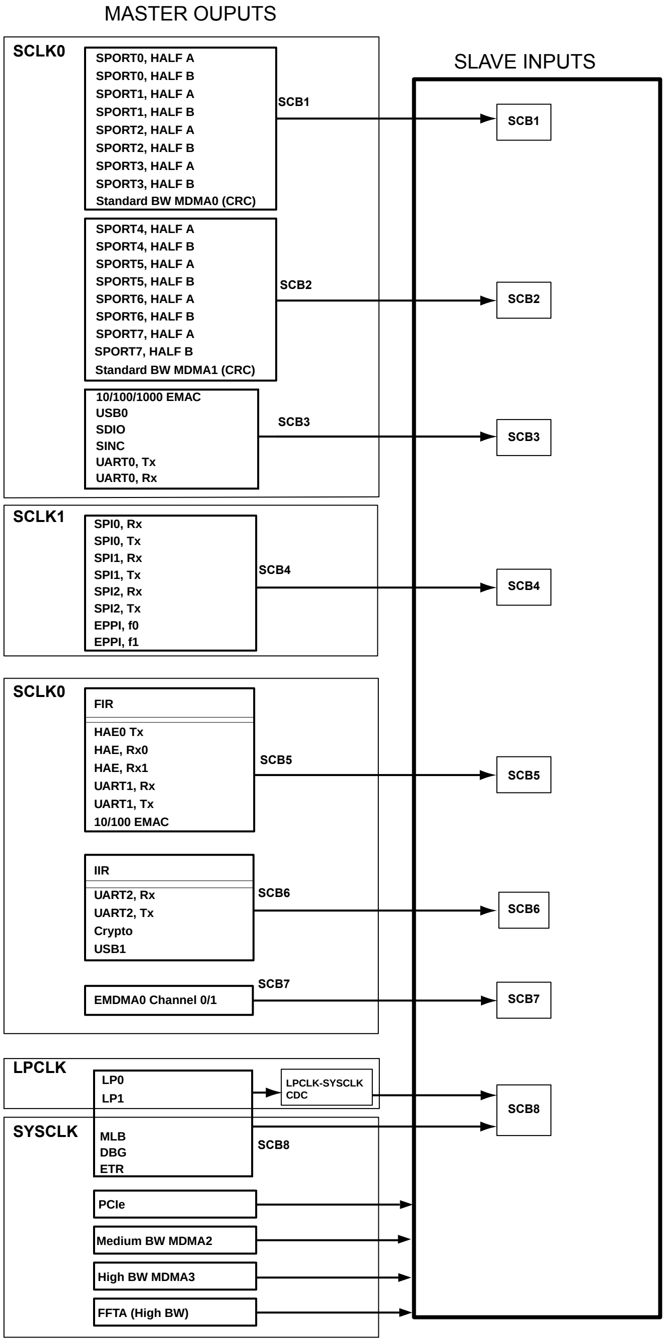

MASTER OUPUTS
SLAVE INPUTS
SCLK0
SPORT0, HALF A
SPORT0, HALF B
SPORT1, HALF A
SPORT1, HALF B
SPORT2, HALF A
SPORT2, HALF B
SPORT3, HALF A
SPORT3, HALF B
Standard BW MDMA0 (CRC)
SCB1
SCB1
SPORT4, HALF A
SPORT4, HALF B
SPORT5, HALF A
SPORT5, HALF B
SPORT6, HALF A
SPORT6, HALF B
SPORT7, HALF A
SPORT7, HALF B
Standard BW MDMA1 (CRC)
SCB2
SCB2
10/100/1000 EMAC
USB0
SDIO
SINC
UART0, Tx
UART0, Rx
SCB3
SCB3
SCLK1
SPI0, Rx
SPI0, Tx
SPI1, Rx
SPI1, Tx
SPI2, Rx
SPI2, Tx
EPPI, f0
EPPI, f1
SCB4
SCB4
SCLK0
FIR
HAE0 Tx
HAE, Rx0
HAE, Rx1
UART1, Rx
UART1, Tx
10/100 EMAC
SCB5
SCB5
IIR
UART2, Rx
UART2, Tx
Crypto
USB1
SCB6
SCB6
EMDMA0 Channel 0/1
SCB7
SCB7
LPCLK
LP0
LP1
LPCLK-SYSCLK CDC
SYSCLK
MLB
DBG
ETR
SCB8
SCB8
PCIe
Medium BW MDMA2
High BW MDMA3
FFTA (High BW)

Figure 54-3: SCB Block Diagram

There are two types of peripherals that use DMA. The first have dedicated DMA channels controlled by the Dedicated DMA Engine (DDE) and have the same operating modes (see DMA Operating Modes) and use the same programming model (DMA Channel Programming Model). The second type is not controlled by the DDE. These peripherals have their own operating modes and programming models (see the peripheral chapter for this information). The peripheral types are shown in the following table.

Table 54-4: DMA Channel Peripherals Controlled by SCBs

	Masters	DDE DMA Channels	Non DDE DMA Channels
SCB1	SPORT0, HALF A	DMA0	
	SPORT0, HALF B	DMA1	
	SPORT1, HALF A	DMA2	
	SPORT1, HALF B	DMA3	
	SPORT2, HALF A	DMA4	
	SPORT2, HALF B	DMA5	
	SPORT3, HALF A	DMA6	
	SPORT3, HALF B	DMA7	
	Std BW MDMA0 CRC, CH0	DMA8	
	Std BW MDMA0 CRC, CH1	DMA9	
SCB2	SPORT4, HALF A	DMA10	
	SPORT4, HALF B	DMA11	
	SPORT5, HALF A	DMA12	
	SPORT5, HALF B	DMA13	
	SPORT6, HALF A	DMA14	
	SPORT6, HALF B	DMA15	
	SPORT7, HALF B	DMA16	
	SPORT7, HALF B	DMA17	
	Std BW MDMA1 CRC, CH0	DMA18	
	Std BW MDMA1 CRC, CH1	DMA19	
SCB3	UART0, TX	DMA20	
	UART0, RX	DMA21	
	EMAC0 (10/100/1000)	n/a	6 channels (3 receive and 3 transmit)
	USB0	n/a	8 channels
	MSI	n/a	1 channel
	SINC	n/a	1 channel
SCB4	SPI0, TX	DMA22	
	SPI0, RX	DMA23	
	SPI1, TX	DMA24	

Table 54-4: DMA Channel Peripherals Controlled by SCBs (Continued)

	Masters	DDE DMA Channels	Non DDE DMA Channels
	SPI1, RX	DMA25	
	SPI2, TX	DMA26	
	SPI2, RX	DMA27	
	PPI, F0	DMA28	
	PPI, F1	DMA29	
SCB5	HAE, OUT	DMA31	
	HAE, IN0	DMA32	
	HAE, IN1	DMA33	
	UART1, TX	DMA34	
	UART1, RX	DMA35	
	FIR0	n/a	1 channel
	EMAC1, 10/100	n/a	2 channels (TX + RX)
SCB6	UART2, TX	DMA37	
	UART2, RX	DMA38	
	IIR0	n/a	1 channel
	CRYPTO	n/a	1 Channel
	USB1	n/a	8 channels
SCB7	EMDMA0	n/a	2 channels
SCB8	LP0	DMA30	
	LP1	DMA36	
	MLB0	n/a	1 channel (4 TDM slots)
SCB0 Masters only	Enh BW MDMA2, CH0	DMA39	
	Enh BW MDMA2, CH0	DMA40	
	FFTA (Max BW, MDMA, CH0)	DMA41	
	FFTA (Max BW, MDMA, CH1)	DMA42	
	Max BW (MDMA3, CH0)	DMA43	
	Max BW (MDMA3, CH0)	DMA44	
	PCIe	n/a	2 Channels (1 read, 1 write)

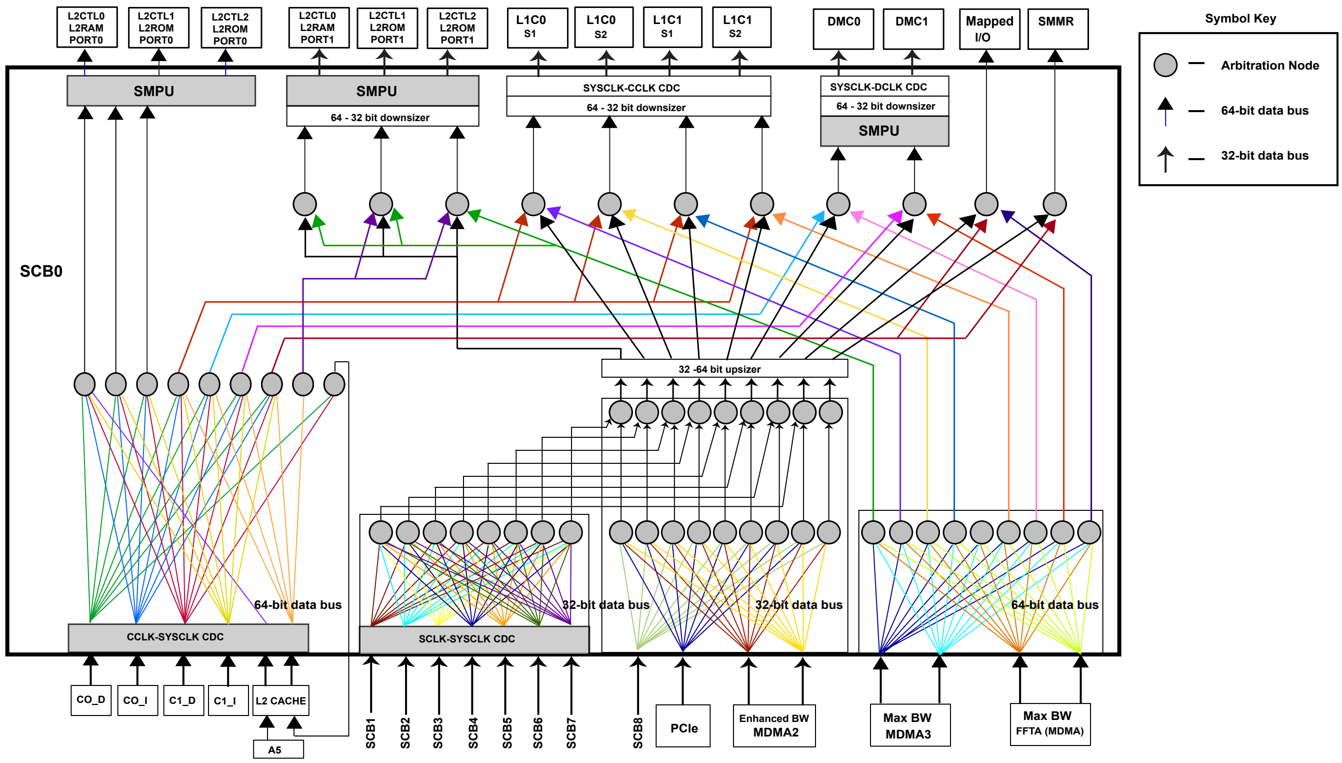

L2CTL0 L2RAM PORT0
L2CTL1 L2ROM PORT0
L2CTL2 L2ROM PORT0
L2CTL0 L2RAM PORT1
L2CTL1 L2ROM PORT1
L2CTL2 L2ROM PORT1
L1C0 S1
L1C0 S2
L1C1 S1
L1C1 S2
DMC0
DMC1
Mapped I/O
SMMR
Symbol Key
Arbitration Node
64-bit data bus
32-bit data bus
SMPU
SMPU
64 - 32 bit downsizer
SYSCLK-CCLK CDC
64 - 32 bit downsizer
SYSCLK-DCLK CDC
64 - 32 bit downsizer
SMPU
SCB0
32 -64 bit upsizer
64-bit data bus
32-bit data bus
32-bit data bus
64-bit data bus
CCLK-SYSCLK CDC
SCLK-SYSCLK CDC
CO_D
CO_I
C1_D
C1_I
L2 CACHE
A5
SCB1
SCB2
SCB3
SCB4
SCB5
SCB6
SCB7
SCB8
PCIe
Enhanced BW MDMA2
Max BW MDMA3
Max BW FFTA (MDMA)

Figure 54-4: SCB Block Diagram - Interconnections

While this figure is useful just for the overview it provides, it is also useful to observe the following relationships that are highlighted.

- The hierarchy of SCBs manages system bus interconnections, multiplexing, and arbitration among the peripherals on the processor.
- The SCBs connections support DMA channels for some peripherals and support dedicated connections for others (such as USB). The connections also support memory-mapped register access for internal memory (L1 and L2) and for external memory.

  The slave interface of the crossbar (where the masters such as DMA connect to) performs two functions. The first function is arbitration. SCBx handles arbitration. The second function is clock conversion. The programmable QoS registers can be viewed as being associated with the SCBx. For example, the programmable QoS registers for DMA16-17, USB0, GIG-E, SDIO, ACM and SINC can be viewed as residing in SCB3. Whenever a transaction is received at DMA16, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB3.
- Most of the peripherals and their SCBs are in the SCLK0_0 domain. Crypto is also in the SCLK0_0 domain. Enhanced bandwidth MDMA, MDMA0 and MDMA1 streams, MLB, DBG, ETR, PCIe and their SCB are in the SYSCLK_0domain.
- Each peripheral has a latency for access across the SCB. The latency varies with the nature of the peripheral. Also, the number of active peripherals (especially for cases where multiple peripherals are active on a shared

SCB) affects SCB performance. Refer to the L2 System Memory chapter for more details on the L2 memory organization.

MDMA channels are unidirectional. For example, MDMA0 SRC is read-only and MDMA1 is write-only. Access to the MMR space of SCB0 is allowed only in secure mode. The MMR space of SCB0 has registers for programming the QoS of various masters and controlling the clock domain crossing.

The following are definitions of acronyms that appear in the figure:

DMA0-DMA35

Indicate DMA channels for peripherals supporting DMA transfers.

SCB0-8

Indicate SCB interfaces, connecting the system bus masters and slaves.

SCLK0, SCLK1, SYSCLK, LPCLK

Indicate clock domains in which the specific SCBs operate. For more information on clock domains, see the Clock Generation Unit (CGU) chapter and the product data sheet.

CDC

Indicates the clock domain crossing.

C0_D, C1_D

Indicates the SHARC processor (0 or 1) data bus (DM/PM).

C0_I, C1_I

Indicates the SHARC processor (0 or 1) instruction bus (PM).

L2CC

Indicates the system L2 cache.

A5

Indicates the ARM Cortex A5 processor.

L1C0 S1, L1C0 S2, L1C1 S1, L1C1 S2

Indicates the L1 memory for SHARC+ processor (0 or 1) slave port (1 or 2).

DMC0, DMC1

Indicates the dynamic memory controller (DMC) interfaces.

PCIe

Indicates the PCI Express interface.

SP0 A/B, SP1 A/B

Indicates the serial port interfaces and their full-duplex halves.

SPI0, SPI1 - RX/TX

Indicates the serial peripheral interfaces ports with receive or transmit paths.

DBG, ETR

Indicates the Debug Port (DBG) and Embedded Trace Router (ETR) which provide access to debug and trace capabilities.

SMPU

Indicates the system memory protection unit (SMPU).

MDMA0, MDMA1

Indicates the memory DMA 0 through 1 interfaces.

USB

Indicates the universal serial bus (USB) interface.

SMMR

Indicates the system memory-mapped register interface.

System Crossbars

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. The SCB (often referred to as the system interconnect fabric), is a collection of inter-connection units connecting system masters to slave memory spaces. The SCB connects one or more master devices to one or more memory-mapped slave devices. Each connected master can be a core that originates an SCB transaction, or a master interface of an upstream SCB cascaded interconnect. Each connected slave can be the final target of an SCB transaction or a slave interface of a downstream cascaded SCB interconnect (forming a hierarchy of SCBs).

Each SCB that has multiple masters and slaves share the total bandwidth of the SCB. (In a M:N configuration where M masters are connected to N slaves through the SCBx.)

The SCB provides separate channels for reads and writes. Read and write accesses through a given SCB do not share bandwidth. All the SCBs are 32 bits wide and run at SCLK speed, and can provide a bandwidth of up to 400 Mbytes per second for reads and writes separately (when SCLK = 100 MHz). Only SCB0, which is the major SCB in the SCB hierarchy, has the multiple paths between multiple master and slave interfaces.

See the SCB Block Diagram.

All other SCBs in the chip connect to SCB0 through different slave interfaces. Other primary masters (DMAs, cores, and so on) in the system are distributed across these small SCBs. For a given SCB, all the master and slaves share the total bandwidth of the SCB. (Only SCB0 is the exception). Since different DMA channels are scattered across different SCBs (SCB1, SCB2 SCB3, and so on), they do not conflict for the bandwidth as long as they are in different SCB and are accessing different slaves. SCB0 allows for concurrent data transfer between multiple bus masters and multiple bus slaves, providing flexibility, and full-duplex operation. For example, the data transfer between SCB4 (one of the MDMA channels), and SMC controller (accessing SRAM memory) can happen in parallel to SCB2 (SPI RX/TX DMA) accessing memory mapped SPI memory with both the transfers occurring at up to 400 MBPS. If system accesses are carefully architected, SCB has a potential of providing sufficient sustained bandwidth in the end system.

Since the SCBs support burst transfers, it is important to configure the requesting master appropriately to make best use of available SCB bandwidth. For a DMA master, choosing the appropriate `DMA_CFG.MSIZE` value, is important from both a functional and a performance perspective. The value in the `DMA_CFG.PSIZE` bit field determines the width of the peripheral bus in use. It can be configured to 1-byte, 2-bytes, or 4-bytes. The `DMA_CFG.MSIZE` value determines the actual size of the SCB bus in use. It also determines the minimum number of bytes which are transferred from or to memory corresponding to a single DMA request or grant. The transfer can be 1-, 2-, 4-, 8-, 16-, or 32-bytes. If the `DMA_CFG.MSIZE` value is greater than the SCB bus width, the SCB performs burst transfers according to the width defined in `DMA_CFG.MSIZE`. When `DMA_CFG.MSIZE` is less than the SCB bus width, bursting is not supported and partial bus use results.

Each of the SCB unit in the fabric consists of N Slave interfaces (MSTn). Each of these interfaces has controls for read quality of service, write quality of service, and functional mode. A subset of these matrices includes controls for IB (Interface Block) sync mode, and bus functional mode. For more details on IB, see the clock domain synchronization section.

Clock Domain Synchronization

Most of the masters in the system operate at the same clock as the SCB, which is SCLK0_0. There is no need to synchronize. The M4 core and L1 code fill blocks (used for cache fill from SPI flash memory or SMC memory) operate in the CCLK domain. The SCB provides the option to program the different synchronization schemes for these masters through the sync mode registers (`SCB3_MST00_SYNC`, and `SCB1_MST00_SYNC`).

These registers perform clock domain crossing synchronization from CCLK to SCLK. The configuration of these registers depends on the CCLK and SCLK configuration.

SCB Bus Master IDs

The SCB bus master ID tables indicate which masters are connected to each of the slave ports of SCB0. The tables also indicate the precise value of the ID as seen by the slave. These values are useful for SWU programming.

NOTE: For an overall diagram of all SCB interconnections, see the SCB Block Diagram.

Table 54-5: Bus Master IDs

SCB	Masters	Hex ID Values
SCB1	DMA0 (SPORT0, HALF A)	0x0, 0x10 (descriptor fetch)
	DMA1 (SPORT0, HALF B)	0x1, 0x11 (descriptor fetch)
	DMA2 (SPORT1, HALF A)	0x2, 0x12 (descriptor fetch)
	DMA3 (SPORT1, HALF B)	0x3, 0x13 (descriptor fetch)
	DMA4 (SPORT2, HALF A)	0x4, 0x14 (descriptor fetch)
	DMA5 (SPORT2, HALF B)	0x5, 0x15 (descriptor fetch)
	DMA6 (SPORT3, HALF A)	0x6, 0x16 (descriptor fetch)
	DMA7 (SPORT3, HALF B)	0x7, 0x17 (descriptor fetch)
	DMA8 (Std BW MDMA0 CRC, CH0)	0x8, 0x18 (descriptor fetch)
	DMA9 (Std BW MDMA0 CRC, CH1)	0x9, 0x19 (descriptor fetch)
SCB2	DMA10 (SPORT4, HALF A)	0x200, 0x210 (descriptor fetch)
	DMA11 (SPORT4, HALF B)	0x201, 0x211 (descriptor fetch)
	DMA12 (SPORT5, HALF A)	0x202, 0x212 (descriptor fetch)
	DMA13 (SPORT5, HALF B)	0x203, 0x213 (descriptor fetch)
	DMA14 (SPORT6, HALF A)	0x204, 0x214 (descriptor fetch)
	DMA15 (SPORT6, HALF B)	0x205, 0x215 (descriptor fetch)
	DMA16 (SPORT7, HALF A)	0x206, 0x216 (descriptor fetch)
	DMA17 (SPORT7, HALF B)	0x207, 0x217 (descriptor fetch)
	DMA18 (Std BW MDMA1 CRC, CH0)	0x208, 0x218 (descriptor fetch)
	DMA19 (Std BW MDMA1 CRC, CH1)	0x209, 0x219 (descriptor fetch)
SCB3	DMA20 (UART0 TX) EMAC0	0x401,0x411 (descriptor fetch)
	DMA21 (UART0 RX) USB0	0x402, 0x412 (descriptor fetch)
	EMAC0	0x400,0x410,0x420,0x430,0x440,0x450,0x460,0x470,0x480,0x490,0x4A0,0x4B0,0x4C0,0x4D0,0x4E0,0x4F0 (descriptor fetch)
	SINC	0x404, 0x40C (descriptor fetch)
	SDIO	0x403

Table 54-5: Bus Master IDs (Continued)

SCB	Masters	Hex ID Values
	USB0	0x405
SCB4	DMA22 (SPI0, TX)	0x800, 0x810 (descriptor fetch)
	DMA23 (SPI0, RX)	0x801, 0x811 (descriptor fetch)
	DMA24 (SPI1, RX)	0x803, 0x813 (descriptor fetch)
	DMA25 (SPI1, TX)	0x802, 0x812 (descriptor fetch)
	DMA26 (SPI2, RX)	0x805, 0x815 (descriptor fetch)
	DMA27 (SPI2, TX)	0x804, 0x814 (descriptor fetch)
	DMA28 (PPI, F0)	0x806, 0x816 (descriptor fetch)
	DMA29 (PPI, F1)	0x807, 0x817 (descriptor fetch)
SCB5	FIR (CH0)	0xA06
	FIR (CH1)	0xA07
	DMA31 (HAE, IN0)	0xA00, 0xA10 (descriptor fetch)
	DMA32 (HAE, IN1)	0xA01, 0xA11 (descriptor fetch)
	DMA33 (HAE, OUT)	0xA02, 0xA12 (descriptor fetch)
	DMA29 (UART1, RX)	0xA04, 0xA14 (descriptor fetch)
	DMA30 (UART1, TX)	0xA03, 0xA13 (descriptor fetch)
	EMAC1	0xA05,0xA15,oxA25,oxA35,0xA45,0xA55,0xA65,0xA75, 0xA85,0xA95,0xAA5,0xAB5,0xAC5,0xAd5,0xAE5,0xAF5 (descriptor fetch)
SCB6	IIR (CH0)	0xC06
	IIR (CH1)	0xC07
	DMA34 (UART2, RX)	0xC04, 0xC14 (descriptor fetch)
	DMA35 (UART2, TX)	0xC05, 0xC15 (descriptor fetch)
	CRYPTO	0xC03
	USB1	0xC00
SCB7	EMDMA0 (CH0)	0xE00
	EMDMA0 (CH1)	0xE01
	EMDMA1 (CH0)	0xE02
	EMDMA1 (CH1)	0xE03

Table 54-5: Bus Master IDs (Continued)

SCB	Masters	Hex ID Values
SCB8	DMA30 LP0	0x601, 0x611 (descriptor fetch)
	DMA36 LP1	0x602, 0x612 (descriptor fetch)
	MLB	0x600
	DBG	0x603
	ETR	0x604
DSP MEM	DMA39 (MSMDMA CH1)	0x1001, 0x1011
	DMA40 (MSMDMA CH0)	0x1000, 0x1010
	PCIe	There are 32 values. "x" is either a 0 or 1.
	DMA43 (HSMDMA CH0)	0x1200, 0x1210
	DMA44 (HSMDMA CH1)	0x1201, 0x1211
	DMA41 (FFTA CH0)	0x1202, 0x1212
	DMA42 (FFTA CH1)	0x1203, 0x1213
	SH0 (DPORT)	0x1400
	SH0 (IPORT)	0x1401
	SH1 (DPORT)	0x1402
	SH1 (IPORT)	0x1403
	PL310 (M0)	There are 32 values. "x" is either a 0 or 1.
	PL310 (M1)	There are 32 values. "x" is either a 0 or 1.

SCB Programming Model

The following sections provide information for programming the SCB properly.

Programming SCB Arbitration

Each slave interface has a QoS value (priority) associated with both read and write channels. These values are 4 bits and are located in the `SCB0_MST[n]_RQOS` and `SCB0_MST[n]_WQOS` registers. At the entry point to the infrastructure, all transactions are allocated this programmable local QoS value. The arbitration of the transaction throughout the infrastructure uses this QoS. At any arbitration node, a fixed priority exists for transactions with a different QoS. The highest value has the highest priority.

If there are coincident transactions at an arbitration node with the same QoS that require arbitration, then the network uses a Least Recently Granted (LRG) algorithm. At each switch, the master with the highest QoS acquires access and that switch output takes the QoS value of the winner for that transaction. At the next switch slave interface, the master uses the QoS value of the winner. QoS can have values from 0 (lowest priority) to 15 (highest priority).

For example in the following figure, ***SCB Arbitration***:

1. At SCB1, masters (1, 2, 3) have RQOS values of (6, 4, 2)
2. At SCB2, masters (4, 5, 6) have RQOS values of (12, 13, 1)

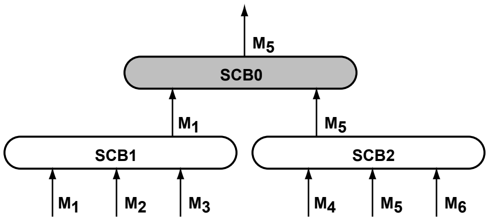


Figure 54-5: SCB Arbitration

In this case, master 1 wins at SCB1, and master 5 wins at SCB2. However, in a perfect competition at SCB0, masters 4 and 5 had the highest overall RQOS values. Masters 4 and 5 would have fought for arbitration directly at SCB0. However, because of the mini-SCBs, master 1, at a much lower RQOS value, is able to win against master 4 and make it all the way to SCB0.

Programming Clock Domain Crossing Registers

In addition to the QoS registers, the interconnect has a set of registers to program the clock domain crossing (CDC) in the interface blocks. The clock domain crossing can be programmed to be one of the following:

- SYNC 1:1
- SYNC 1:n
- SYNC n:1
- ASYNC
- SYNC m:n

The CDC can be programmed at the interface between DCLK and the SYSCLK by writing appropriate bits in the `SCB1_MST00_SYNC` register (address 0x30200020) for DMC0 and the `SCB3_MST00_SYNC` register (address 0x30300020) for DMC1. Refer to the Register Description section for more details.

To change the clock domain crossing mode, follow the actions described in the ***Changing Clock Domain Crossing Modes*** table.

Table 54-6: Changing Clock Domain Crossing Modes

Original Mode	Required Mode	Action
ASYNC	Any other mode	Change the clocks then change the MMR register
Any mode	ASYNC	Change the MMR register then change clocks to ASYNC.
m:n	1:1	Change the clocks, then change the register.

Table 54-6: Changing Clock Domain Crossing Modes (Continued)

Original Mode	Required Mode	Action
1:1	m:n	Change the register, then change the clocks.

The other CDCs are not programmable. The default values are:

- SCLK : SYSCLK CDC – 1: n
- CCLK : SYSCLK CDC – m : 1
- SYSCLK : CCLK CDC – 1 : n
- LPCLK : SYSCLK CDC – ASYNC

SCB Programming Concepts

The SCB arbitration model among master or slave SCBs of the processor is fixed (not programmable). But, each slave does have a quality of service (QoS) programmable feature that affects arbitration.

The arbitration of transactions in SCB is based on the QoS value or the priority of the transaction. All masters with the same priority form a priority group. Arbitration is granted to the highest priority group from which a member is trying to win access, and within that group, to the highest master at that time. When a master wins arbitration, it is relegated to the bottom of its group to ensure that it cannot prevent other masters in its group from accessing the slave.

If you configure all master priorities to different levels, the arbiter implements a fixed priority scheme. This scheme occurs because each master is in a group of its own, and therefore, masters maintain their ordering.

The LRG and fixed priority modes concurrently exist when the master priority value registers are programmed with a combination of identical and unique values.

NOTE: The SCB arbitration hierarchy is fixed (for example, SCB1 master to SCB1 slave). However, multiple master inputs to the same slave permit QoS programming.

The *LRG Arbitration Example* figure shows three groups with different QoS values. Masters in the same group share a QoS value. The arbitration occurs using an LRG scheme.

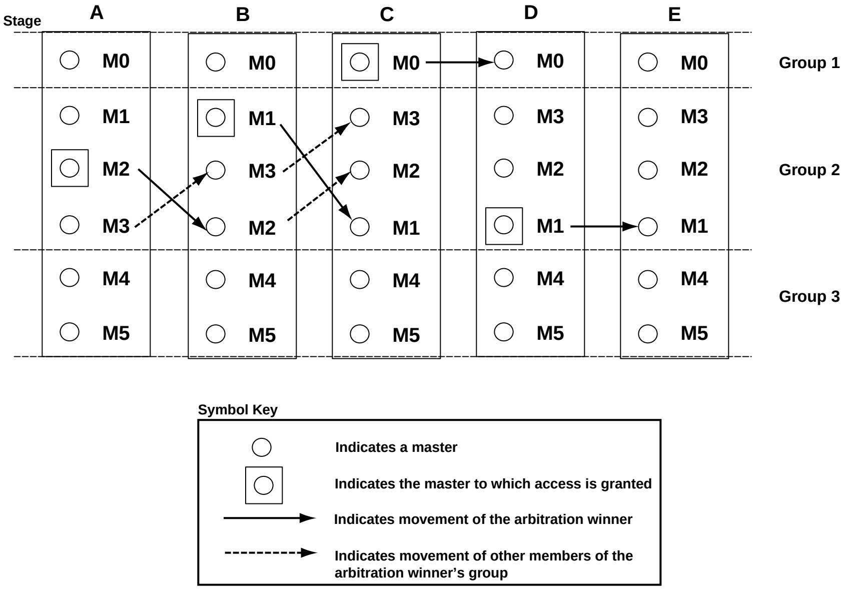

Stage
A
B
C
D
E
M0
M1
M2
M3
M4
M5
Group 1
Group 2
Group 3
Symbol Key
Indicates a master
Indicates the master to which access is granted
Indicates movement of the arbitration winner
Indicates movement of other members of the arbitration winner's group

Figure 54-6: LRG Arbitration Example

The QoS value assigned to a transaction at entry point is carried forward by the transaction as it passes through all arbitration stages in the SCB. QoS for all masters is configured as programmable in system fabric interconnect in ADSP-SC58x.

The priority of the masters fits into three groups:

- Group A – Peripherals without flow control
- Group B – Cores and debug ports
- Group C – Peripherals with flow control or low bandwidth and MDMAs
- Group D – Peripherals with flow control but without bandwidth control

Group A: Peripherals without Flow Control

These masters are assigned the highest priority. Long delays in servicing requests due to losing arbitration could result in underrun or overrun of FIFOs and loss of data. All masters in this group are assigned the QoS reset value of 13. The Group A masters include the following peripherals:

- 10/100/1000 EMAC
- EPPI
- MLB (synchronous mode does not support flow control)
- Serial ports
- 10/100 EMAC

Group B: Cores and Debug Ports

Typically, software running on the cores has the highest priority after the peripherals in Group A. These masters are assigned the QoS reset value of 9.

Group C: Peripherals with Flow Control or Low Bandwidth and MDMAs

This group includes MDMAs and peripherals with flow control or low bandwidth. These masters are assigned the QoS reset value of 5. The Group C masters include the following:

- USB
- SDIO
- Link Ports
- PCIe
- HAE
- SINC
- Standard-BW MDMA
- Enhanced-BW MDMA
- Max-BW MDMA
- FFT-MDMA (Shared Max-BW MDMA)
- SPI
- UART

Although HAE and SINC do not have any flow control, their bandwidth requirement is much less, which places them in this group. MDMAs are given the same QoS as other peripherals with flow control since they can be made to consume lesser bandwidth by using the bandwidth control feature in the DDE.

Group D: Peripherals with Flow Control but without Bandwidth Control

Peripherals with flow control but without bandwidth control are given the lowest priority. This priority prevents them from flooding the system bandwidth. These peripherals can withstand longer latency times without any data loss, but can potentially keep issuing requests continuously. These masters are assigned the reset value of QoS as 1. The Group D masters include the following peripherals:

- EMDMA (delay line DMA)
- FIR
- IIR
- Crypto

NOTE: Although USB, SDIO, SINC, and other peripherals in Group C do not always have bandwidth control mechanisms like MDMAs, their rate of issuing requests is limited by their low bandwidth requirement. The QoS reset values are spaced out as 1, 5, 9 and 13 instead of 0, 1, 2 and 3. These values allows the software to change the priority of a few masters in the system without having to reprogram the QoS of all the masters.

Table 54-7: QoS Register Table

Master IDs	Master	read_qos Reset Value	write_qos Reset Value
0	SPORT0_A_DMA	13	13
1	SPORT0_B_DMA	13	13
2	SPORT1_A_DMA	13	13
3	SPORT1_B_DMA	13	13
4	SPORT2_A_DMA	13	13
5	SPORT2_B_DMA	13	13
6	SPORT3_A_DMA	13	13
7	SPORT3_B_DMA	13	13
8	STD BW MDMA0_SRC	5	5
9	STD BW MDMA0_DST	5	5
10	SPORT4_A_DMA	13	13
11	SPORT4_B_DMA	13	13
12	SPORT5_A_DMA	13	13
13	SPORT5_B_DMA	13	13
14	SPORT6_A_DMA	13	13
15	SPORT6_B_DMA	13	13
16	SPORT7_A_DMA	13	13
17	SPORT7_B_DMA	13	13
18	STD BW MDMA1_SRC	5	5
19	STD BW MDMA1_DST	5	5
20	EMAC0	13	13
21	USB0	5	5
22	UART0_TXDMA	5	5
23	UART0_RXDMA	5	5
24	MSI0	5	5
25	SINC0	5	5
26	SPI0_TXDMA	5	5

Table 54-7: QoS Register Table (Continued)

Master IDs	Master	read_qos Reset Value	write_qos Reset Value
27	SPI0_RXDMA	5	5
28	SPI1_TXDMA	5	5
29	SPI1_RXDMA	5	5
30	SPI2_TXDMA	5	5
31	SPI2_RXDMA	5	5
32	EPPI0_DMA_CH0	13	13
33	EPPI0_DMA_CH1	13	13
34	FIR0_CH0	1	1
35	FIR0_CH1	1	1
36	HAE0_RXDMA_CH0	5	5
37	HAE0_RXDMA_CH1	5	5
38	HAE0_TXDMA	5	5
39	UART1_TXDMA	5	5
40	UART1_RXDMA	5	5
41	EMAC1	13	13
42	IIR0_CH0	1	1
43	IIR0_CH1	1	1
44	USB1	5	5
47	PKTE0 (Crypto)	1	1
48	UART2_TXDMA	5	5
49	UART2_RXDMA	5	5
50	EMDMA0_CH0	1	1
51	EMDMA0_CH1	1	1
52	EMDMA1_CH0	1	1
53	EMDMA1_CH1	1	1
54	PCIE0	5	5
55	ENH BW MDMA2_SRC	5	5
56	ENH BW MDMA2_DST	5	5
57	MLB0	13	13
58	LP0_DMA	5	5
59	LP1_DMA	5	5

Table 54-7: QoS Register Table (Continued)

Master IDs	Master	read_qos Reset Value	write_qos Reset Value
60	CS_DAP	9	9
61	CS_ETR	9	9
62	MAX BW MDMA3_SRC	5	5
63	MAX BW MDMA3_DST	5	5
64	MAX BW MDMA (Shared Max-BW MDMA) FFTA0_TXDMA	5	5
65	MAX BW MDMA (Shared Max-BW MDMA) FFTA0_RXDMA	5	5
66	C1_DPORT	9	9
67	C1_IPORT	9	9
68	C2_DPORT	9	9
69	C2_IPORT	9	9
70	C0_L2CC_M0	9	9
71	C0_L2CC_M1	9	9

ADSP-SC58x SCB0 Register Descriptions

System Interconnect Fabric (SCB0) contains the following registers.

Table 54-8: ADSP-SC58x SCB0 Register List

Name	Description
`SCB0_MST[n]_RQOS`	Read Quality of Service for Master n
`SCB0_MST[n]_WQOS`	Write Quality of Service for Master n

Read Quality of Service for Master n

The SCB0_MST[n]_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

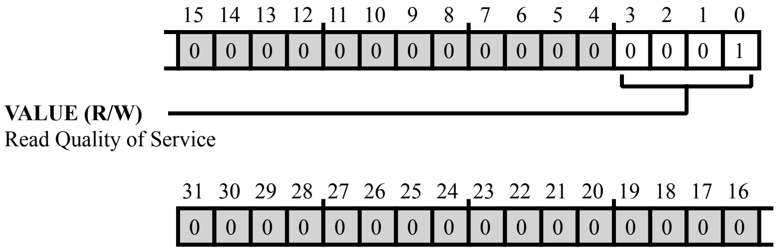

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
VALUE (R/W)
Read Quality of Service
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

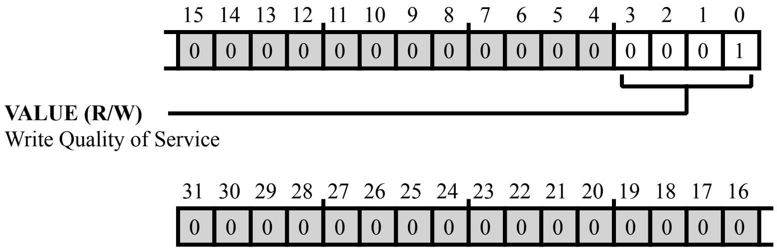
Figure 54-7: SCB0_MST[n]_RQOS Register Diagram

Table 54-9: SCB0_MST[n]_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The `SCB0_MST[n]_RQOS.VALUE` bit field indicates the read QOS or priority value for the indicated master.

Write Quality of Service for Master n

The `SCB0_MST[n]_WQOS` register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
VALUE (R/W)
Write Quality of Service
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 54-8: SCB0_MST[n]_WQOS Register Diagram

Table 54-10: SCB0_MST[n]_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The `SCB0_MST[n]_WQOS.VALUE` bit field indicates the write QOS or priority value for the indicated master.

ADSP-SC58x SCB1 Register Descriptions

System Crossbar for DMC Memory Space (SCB1) contains the following registers.

Table 54-11: ADSP-SC58x SCB1 Register List

Name	Description
`SCB1_MST00_SYNC`	Mst00 Interface Block Sync Mode

Mst00 Interface Block Sync Mode

The SCB1_MST00_SYNC register is used to program the clock domain crossing in the interface blocks.

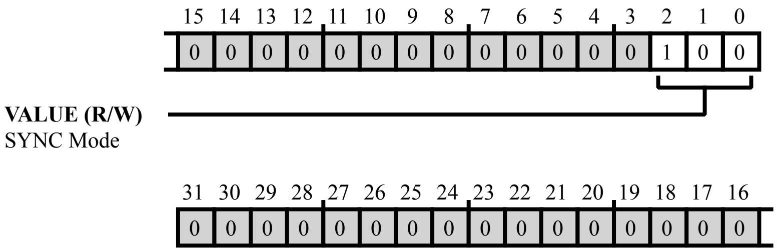

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
VALUE (R/W)
SYNC Mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 54-9: SCB1_MST00_SYNC Register Diagram

Table 54-12: SCB1_MST00_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	SYNC Mode. The `SCB1_MST00_SYNC.VALUE` bit field is used to configure the SYSCLK-DCLK CDC for DMC for the following clock domain boundaries.	
		0	Sync 1:1
		1	Sync n:1
		2	Sync 1:n
		3	Sync m:n
		4	Async
		5	Reserved
		6	Reserved
		7	Reserved

ADSP-SC58x SCB3 Register Descriptions

SMMR Fabric (SCB3) contains the following registers.

Table 54-13: ADSP-SC58x SCB3 Register List

Name	Description
SCB3_DCLK0_WR_TIDE	DCLK0 Interface Block APB WR Tidemark
SCB3_MST00_SYNC	SYNC Mode

DCLK0 Interface Block APB WR Tidemark

The `SCB3_DCLK0_WR_TIDE` register indicates the maximum permitted number of active transactions before the QoS mechanism is activated.

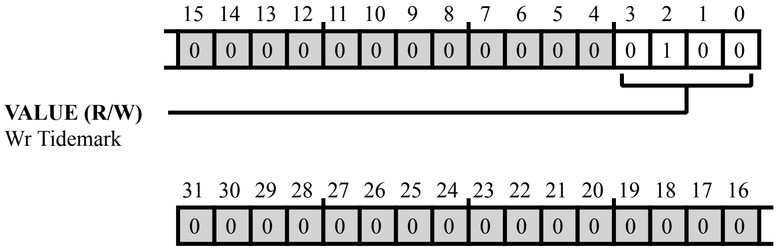

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
VALUE (R/W)
Wr Tidemark
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 54-10: SCB3_DCLK0_WR_TIDE Register Diagram

Table 54-14: SCB3_DCLK0_WR_TIDE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Wr Tidemark. The `SCB3_DCLK0_WR_TIDE.VALUE` bit field indicates the maximum permitted number of active transactions before the QoS mechanism is activated.

SYNC Mode

The SCB3_MST00_SYNC register is used to program the clock domain crossing in the interface blocks.

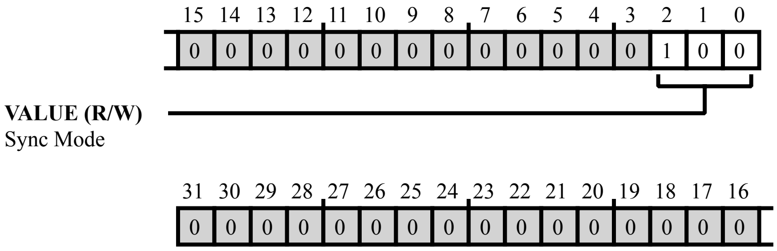

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
VALUE (R/W)
Sync Mode
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 54-11: SCB3_MST00_SYNC Register Diagram

Table 54-15: SCB3_MST00_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	Sync Mode. The `SCB3_MST00_SYNC.VALUE` bit field is used to program the clock domain crossing in the interface blocks.	
		0	Sync 1:1
		1	Sync n:1
		2	Sync 1:n
		3	Sync m:n
		4	Async
		5	Reserved
		6	Reserved
		7	Reserved

55 System Watchpoint Unit (SWU)

The system watchpoint unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

Each SWU contains four match groups of registers with associated hardware. These four SWU match groups operate independently, but share common event (interrupt and trigger) outputs. Each match group can monitor either the write or read address channel and can operate in either watchpoint mode or bandwidth mode.

SWU Features

The system watchpoint unit has the following features.

- Four independent match groups for each SWU
- Each match group can operate in either bandwidth mode or watchpoint mode
- In addition to the read and write address channels, the SWU monitors read and write ID, VALID and READY signals for data channels.

SWU Functional Description

This section describes the function of the SWU match block, interface block, and MMR block.

ADSP-SC58x SWU Register List

The System Watchpoint Unit (SWU) provides debug and development support through flexible transaction level and bandwidth monitoring and associated event triggering. The SWU can generate events based on monitoring transactions at the system slaves through watchpoint-match groups. The SWU also provides watchpoint event status reporting, a global lock, and processor reset capability. A set of registers governs SWU operations. For more information on SWU functionality, see the SWU register descriptions.

Table 55-1: ADSP-SC58x SWU Register List

Name	Description
`SWU_CNT[n]`	Count Register n
`SWU_CTL[n]`	Control Register n
`SWU_CUR[n]`	Current Register n
`SWU_GCTL`	Global Control Register
`SWU_GSTAT`	Global Status Register
`SWU_HIST[n]`	Bandwidth History Register n
`SWU_ID[n]`	ID Register n
`SWU_LA[n]`	Lower Address Register n
`SWU_TARG[n]`	Target Register n
`SWU_UA[n]`	Upper Address Register n

ADSP-SC58x SWU Interrupt List

Table 55-2: ADSP-SC58x SWU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
225	SWU0_EVT	SWU0 Event	None	
226	SWU2_EVT	SWU2 Event	None	
227	SWU1_EVT	SWU1 Event	None	
228	SWU4_EVT	SWU4 Event	None	
229	SWU3_EVT	SWU3 Event	None	
230	SWU6_EVT	SWU6 Event	None	
231	SWU5_EVT	SWU5 Event	None	
232	SWU7_EVT	SWU7 Event	None	
233	SWU8_EVT	SWU8 Event	None	
234	SWU9_EVT	SWU9 Event	None	
235	SWU10_EVT	SWU10 Event	None	
236	SWU11_EVT	SWU11 Event	None	
237	SWU12_EVT	SWU12 Event	None	
238	SWU13_EVT	SWU13 Event	None	
239	SWU14_EVT	SWU14 Event	None	
240	SWU15_EVT	SWU15 Event	None	

ADSP-SC58x SWU Trigger List

Table 55-3: ADSP-SC58x SWU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
99	`SWU0_EVT`	SWU0 Event	None
100	`SWU2_EVT`	SWU2 Event	None
101	`SWU1_EVT`	SWU1 Event	None
102	`SWU4_EVT`	SWU4 Event	None
103	`SWU3_EVT`	SWU3 Event	None
104	`SWU6_EVT`	SWU6 Event	None
105	`SWU5_EVT`	SWU5 Event	None
106	`SWU7_EVT`	SWU7 Event	None
107	`SWU8_EVT`	SWU8 Event	None
108	`SWU9_EVT`	SWU9 Event	None
109	`SWU10_EVT`	SWU10 Event	None
110	`SWU11_EVT`	SWU11 Event	None
111	`SWU12_EVT`	SWU12 Event	None
112	`SWU13_EVT`	SWU13 Event	None
113	`SWU14_EVT`	SWU14 Event	None
114	`SWU15_EVT`	SWU15 Event	None
115	`SWU0_DBG`	SWU0 Debug	Edge
116	`SWU2_DBG`	SWU2 Debug	Edge
117	`SWU1_DBG`	SWU1 Debug	Edge
118	`SWU4_DBG`	SWU4 Debug	Edge
119	`SWU3_DBG`	SWU3 Debug	Edge
120	`SWU6_DBG`	SWU6 Debug	Edge
121	`SWU5_DBG`	SWU5 Debug	Edge
122	`SWU7_DBG`	SWU7 Debug	Edge
123	`SWU8_DBG`	SWU8 Debug	Edge
124	`SWU9_DBG`	SWU9 Debug	Edge
125	`SWU10_DBG`	SWU10 Debug	Edge
126	`SWU11_DBG`	SWU11 Debug	Edge
127	`SWU12_DBG`	SWU12 Debug	Edge
128	`SWU13_DBG`	SWU13 Debug	Edge

Table 55-3: ADSP-SC58x SWU Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
129	`SWU14_DBG`	SWU14 Debug	Edge
130	`SWU15_DBG`	SWU15 Debug	Edge

Table 55-4: ADSP-SC58x SWU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
119	`SWU0_EN`	SWU0 Enable	Pulse
120	`SWU2_EN`	SWU2 Enable	Pulse
121	`SWU1_EN`	SWU1 Enable	Pulse
122	`SWU4_EN`	SWU4 Enable	Pulse
123	`SWU3_EN`	SWU3 Enable	Pulse
124	`SWU6_EN`	SWU6 Enable	Pulse
125	`SWU5_EN`	SWU5 Enable	Pulse
126	`SWU7_EN`	SWU7 Enable	Pulse
127	`SWU8_EN`	SWU8 Enable	Pulse
128	`SWU9_EN`	SWU9 Enable	Pulse
129	`SWU10_EN`	SWU10 Enable	Pulse
130	`SWU11_EN`	SWU11 Enable	Pulse
131	`SWU12_EN`	SWU12 Enable	Pulse
132	`SWU13_EN`	SWU13 Enable	Pulse
133	`SWU14_EN`	SWU14 Enable	Pulse
134	`SWU15_EN`	SWU15 Enable	Pulse

SWU Definitions

The following definitions are helpful when using the SWU module.

Watchpoint Mode

Mode in which transactions are recognized on an exact match. Actions can be configured to be taken after a specified number of matches have occurred.

Bandwidth Mode

Mode in which transactions are recognized and counted inside sampling window.

SWU Architectural Concepts

The information in this section provides basic module design concepts.

SWU-to-SCB Interface

The SWU system crossbar interface block latches all transactions on the system crossbar read and write address channels when the `SWU_GCTL.EN` register enable bit is set.

SWU Block Diagram

The *System Watchpoint Unit Top-Level Block Diagram* figure shows the SWU block diagram.

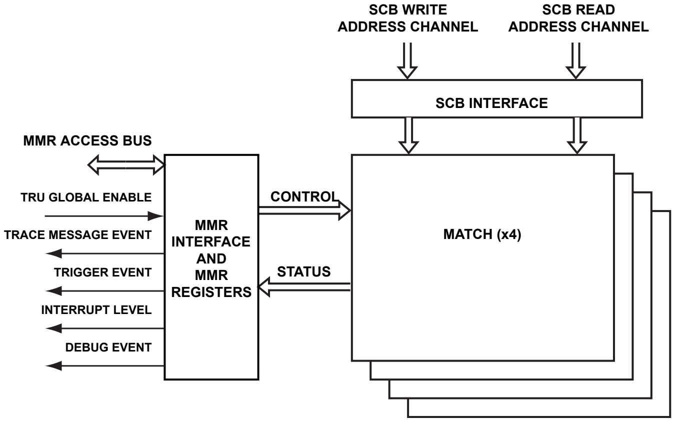

SCB WRITE ADDRESS CHANNEL
SCB READ ADDRESS CHANNEL
SCB INTERFACE
MMR ACCESS BUS
TRU GLOBAL ENABLE
TRACE MESSAGE EVENT
TRIGGER EVENT
INTERRUPT LEVEL
DEBUG EVENT
MMR INTERFACE AND MMR REGISTERS
CONTROL
STATUS
MATCH (x4)

Figure 55-1: System Watchpoint Unit Top-Level Block Diagram

SCB Interface Block

The SWU system crossbar (SCB) latches all transactions on the SCB read and write address channels when the `SWU_GCTL.EN` bit is set.

MMR Interface Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupt requests and events from each match block into common outputs.

SWU Operating Modes

There are two operating modes supported by the SWU: bandwidth mode and watchpoint mode.

Bandwidth Mode

In bandwidth mode, the SWU module counts transactions which match the properties specified in the `SWU_CTL[n]` register during a sampling window determined by the respective `SWU_CNT[n]` register. At the end of the sampling window, the SWU stores results in the `SWU_HIST[n]` register. If the sampled bandwidth falls outside a programmed range, then the programmed action occurs.

Watchpoint Mode

In watchpoint mode, if the `SWU_CTL[n].CNTEN` bit is set, the SWU module decrements the `SWU_CUR[n]` register for each match, until it equals zero, at which point any programmed actions occur. The `SWU_CUR[n]` register is then reloaded from the `SWU_CNT[n]` register (if the `SWU_CTL[n].CNTRPTEN` bit is set), and the cycle repeats. If the `SWU_CTL[n].CNTRPTEN` bit is not set, any programmed actions happen on every match.

Match Block

There are four match blocks for each SWU. Each SWU match block can monitor either the read or write address channel, selected by the `SWU_CTL[n].DIR` bit. The SWU match block can operate in either watchpoint or bandwidth mode, as selected by the `SWU_CTL[n].BWEN` bit.

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive or exclusive range), ID (with masking), security, and lock type. All enabled matches are AND'ed together to determine a match.

Scaling

Scaling allows the SWU to count more transactions by scaling the number of transactions and the number of clock cycles in bandwidth window (CNTn register) by 10,100 or 1000. This functionality is applicable only in bandwidth mode (`SWU_CTL[n].BWEN`==1).

Consider a case where the `SWU_TARG[n].BWMAX` bit field is programmed to 2. In the absence of any scaling, bandwidth overflow occurs when the `SWU_CUR[n].CURBW` bit field value > the `SWU_TARG[n].BWMAX` bit field value (2). With scaling set to 1:100 and after 275 transactions, the `SWU_CUR[n].CURBW` value is still 2 and equal to `SWU_TARG[n].BWMAX`. This event triggers a bandwidth overflow as the actual number of transactions is greater than 200 (2 × 100). The code can be rerun with a smaller scaling selected to analyze the cause for the overflow.

The counter increments after every group of N transactions with scaling enabled, where N is either 10,100 or 1000. (For N = 10, 0–9 transactions == 0 scaled transaction, 10–19 transactions == 1 scaled transaction, 20–29 transactions == 2 scaled transactions, and so on).

Fractional counts with scaling enabled are discarded and not rolled over from one bandwidth window to the next. For example, consider a case where scaling by 1000 is configured and the first window has 1200 transactions. The second window has 2800 transactions. The bandwidth for the first window is read as 1 and the second as 2. The 200 transactions from the first window do not get carried forward to the next window.

Do not use scaling by 10 with the `SWU_CTL[n].BLENINC` bit enabled if any of the masters accessing the slave can launch a transaction of burst length 16.

SWU Event Control

The SWU can generate the following events when a match occurs and when the event is enabled by configuring the proper bits in the control register.

1. Trace Message

2. Trigger
3. Interrupt request
4. Debug

SWU Interrupts

All interrupt requests and events from each match block are merged into common outputs.

SWU Status and Errors

SWU status and errors are reported in the `SWU_GSTAT` register. The SWU records an address error when a write or read attempt is made to the MMR address space of the SWU and the register does not exist. This error is the only one the SWU records. The register contains bits that perform the following functions.

- Indicate whether a particular match group sampled a transaction that is below a minimum target or above a maximum target in bandwidth mode.
- Indicate whether a watchpoint match occurred for each match group.
- Indicate whether an interrupt request was triggered due to a match event from one of the match groups.

Triggers

The SWU can be either a trigger master or a trigger slave depending on the trigger routing unit (TRU) configuration. As a trigger master, programs must set the `SWU_CTL[n].TRGEN` bit so that when a match condition is met, a trigger event is generated. Each SWU in the system can also be a trigger slave when mapped as one in the TRU.

When the SWU is a slave, a trigger event activates the SWU by automatically setting the `SWU_GCTL.EN` bit. Since the SWU can be automatically enabled through a trigger event, programs must pre-configure the SWU before enabling the TRU. Furthermore, although a trigger event can enable the SWU as a slave, to disable the SWU, programs must manually clear the `SWU_GCTL.EN` bit.

SWU Programming Model

Program the appropriate registers to use the SWU. Each control register configures aspects such as:

- The direction of monitoring (reads or writes)
- Whether SWU uses bandwidth mode or watchpoint mode
- The setup of events that are triggered when a condition is met while monitoring using the SWU

Configure supplemental registers such as the lower (`SWU_LA[n]`) and upper (`SWU_UA[n]`) address boundaries before enabling the SWU.

Once the SWU has been enabled and the monitoring conditions are met, events are generated when configured.

The global status register (`SWU_GSTAT`) can be read to observe the status of the units.

The *SWU Logical Flow* diagram shows the logical program flow of the SWU.

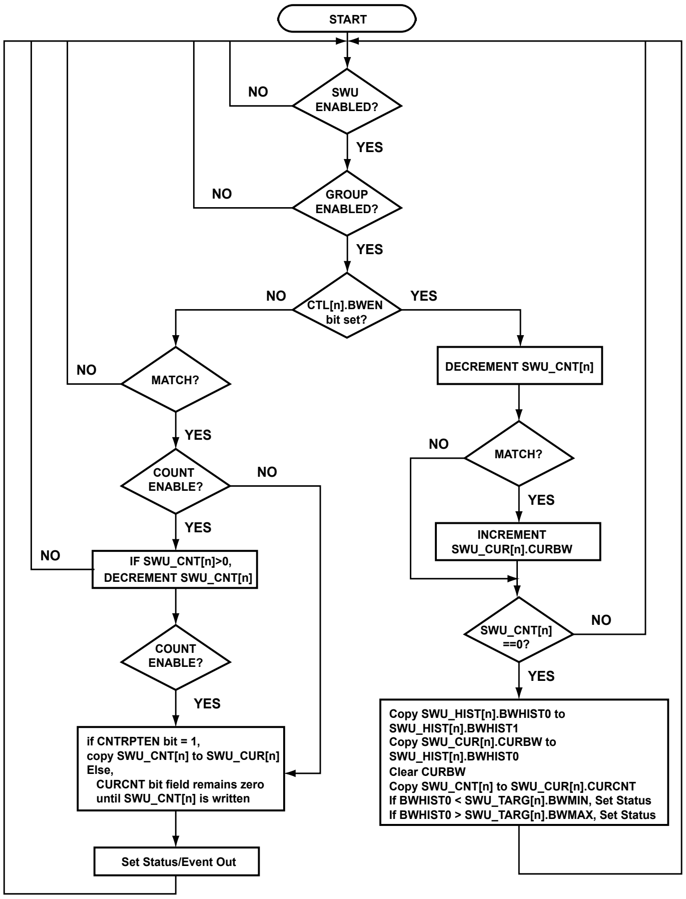

START
NO
SWU ENABLED?
YES
NO
GROUP ENABLED?
YES
NO
CTL[n].BWEN bit set?
YES
NO
MATCH?
YES
COUNT ENABLE?
NO
YES
NO
IF SWU_CNT[n]>0, DECREMENT SWU_CNT[n]
COUNT ENABLE?
YES
if CNTRPTEN bit = 1, copy SWU_CNT[n] to SWU_CUR[n] Else, CURCNT bit field remains zero until SWU_CNT[n] is written
Set Status/Event Out
DECREMENT SWU_CNT[n]
NO
MATCH?
YES
INCREMENT SWU_CUR[n].CURBW
SWU_CNT[n] ==0?
NO
YES
Copy SWU_HIST[n].BWHIST0 to SWU_HIST[n].BWHIST1
Copy SWU_CUR[n].CURBW to SWU_HIST[n].BWHIST0
Clear CURBW
Copy SWU_CNT[n] to SWU_CUR[n].CURCNT
If BWHIST0 < SWU_TARG[n].BWMIN, Set Status
If BWHIST0 > SWU_TARG[n].BWMAX, Set Status

Figure 55-2: SWU Logical Flow

SWU Mode Configuration

The following sections show the steps for configuring SWU bandwidth mode and watchpoint mode.

Configuring the SWU for Bandwidth Mode

In bandwidth mode, the SWU counts transactions which match during a sampling window. At the end of the sampling window, the SWU stores the results. An action can be taken if the sampled bandwidth goes above or falls below a programmed range.

1. Configure the `SWU_CTL[n].DIR` bit to test the match on writes or reads.
2. Configure the `SWU_CTL[n].ACMPM` bits to address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the `SWU_CTL[n].IDCMPEN` bit.
4. Set the `SWU_CTL[n].BLENINC` bit to increment by burst length or clear it to increment by 1.

5. Configure the `SWU_CTL[n].MAXACT` and `SWU_CTL[n].MINACT` bits to enable actions taken when the bandwidth goes above the maximum, or falls below the minimum, respectively.
6. Set the `SWU_CTL[n].BWEN`=1 to enable bandwidth mode.
7. Program the lower address register, `SWU_LA[n]`, and upper address register, `SWU_UA[n]`, to define the memory range for comparison.
8. If ID comparison is enabled, program the ID register, `SWU_ID[n]`.
9. Program the count register, `SWU_CNT[n]`, with the number of clock cycles for which the SWU counts the number of matches.
10. If the SWU is set to respond when the bandwidth measurement underflows or overflows, program the min and max values into the `SWU_TARG[n]` register.
11. Enable the SWU

The SWU counts the number of matches in a pre-defined number of clock cycles as programmed. As an option, it can define lower and upper limits. If the matches fall outside the limits, an action can be taken.

Configuring the SWU for Watchpoint Mode

In watchpoint mode, the SWU can trigger a programmed action after every match or after a number of matches. This sequence can be automatically reset.

1. Set the `SWU_CTL[n].DIR` bit to test the match on writes or reads.
2. Configure the `SWU_CTL[n].ACMPM` bits for address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the `SWU_CTL[n].IDCMPEN`.
4. Set the `SWU_CTL[n].CNTEN` bit to enable the events to be triggered when the count decrements to zero.
5. If needed, set the `SWU_CTL[n].CNTRPTEN` bit to automatically reload the counter after it has decremented to zero to start another match sequence.
6. Clear the `SWU_CTL[n].BWEN` = 0 to configure watchpoint mode.
7. Configure the lower address register, `SWU_LA[n]`, and upper address register, `SWU_UA[n]`, to define the memory range for comparison.
8. If ID comparison is enabled, configure the ID register, `SWU_ID[n]`.
9. Configure the count register, `SWU_CNT[n]`, to determine how many matches occur before the watchpoint group responds.
10. Enable the SWU.

The SWU detects and counts down the number of match occurrences. When the counter expires, an action is taken.

ADSP-SC58x SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 55-5: ADSP-SC58x SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

Count Register n

The SWU count registers (`SWU_CNT[n]`) contain a 16-bit count field (`SWU_CNT[n].COUNT`) whose usage differs depending on the mode of the watchpoint group. In bandwidth mode, the `SWU_CNT[n].COUNT` field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the `SWU_CNT[n].COUNT` field value determines how many matches occur before the watchpoint group takes action.

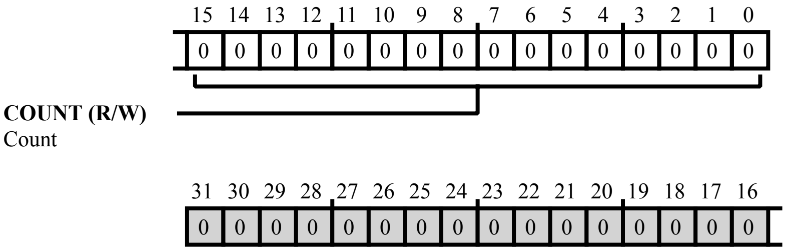

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
COUNT (R/W)
Count
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 55-3: SWU_CNT[n] Register Diagram

Table 55-6: SWU_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count. The `SWU_CNT[n].COUNT` field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the `SWU_CNT[n].COUNT` field value determines how many matches occur before the watchpoint group takes action.

Control Register n

The SWU control registers (`SWU_CTL[n]`) contain watchpoint attribute controls for all four watchpoint groups. These controls include enabling watchpoints, selecting the transaction direction for match, selecting address comparison mode, enabling ID comparison, enabling security comparison, enabling locked comparison, enabling cycle count, enabling count repeat, enabling debug events, enabling interrupts, enabling triggers, enabling trace messages, enabling bandwidth mode, selecting the burst length increment, and enabling bandwidth underflow and overflow detection.

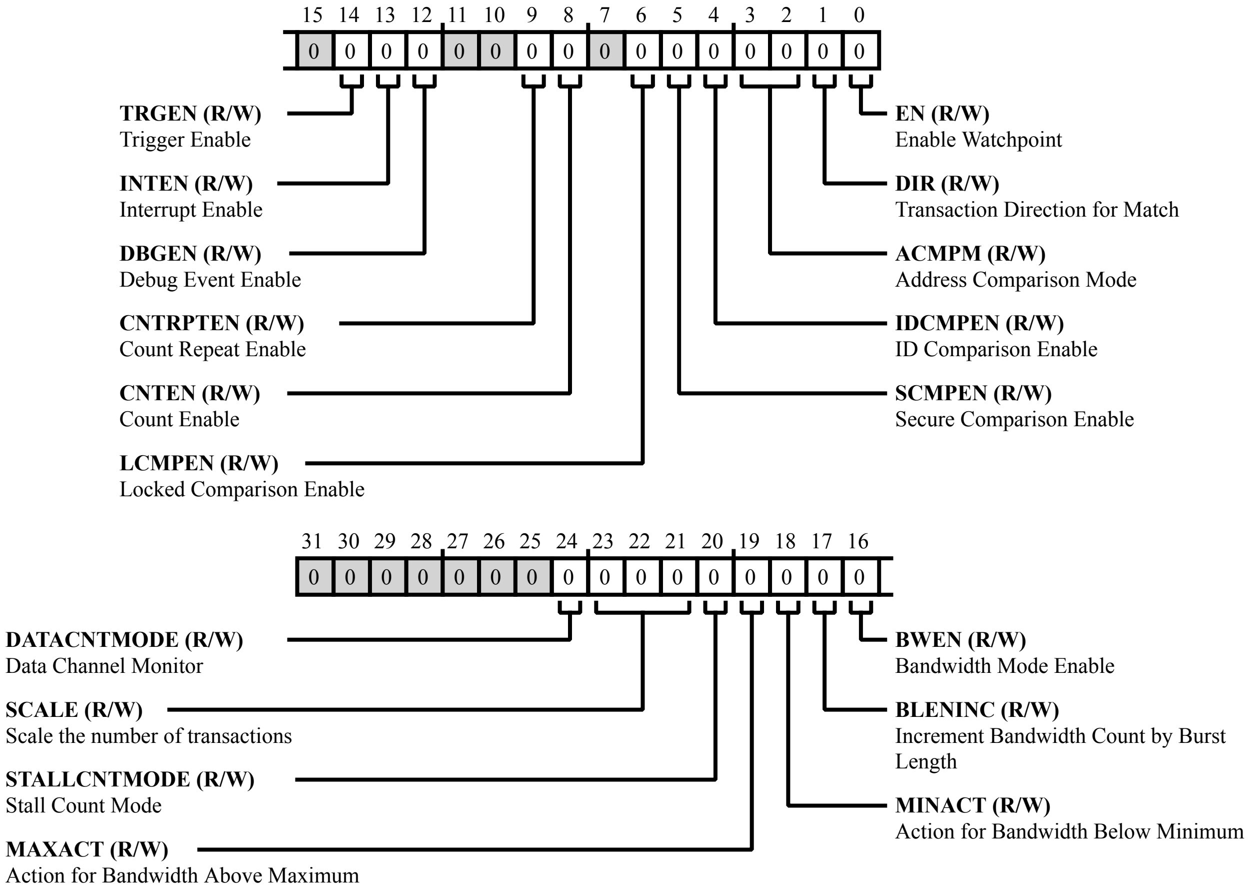

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TRGEN (R/W)
Trigger Enable
INTEN (R/W)
Interrupt Enable
DBGEN (R/W)
Debug Event Enable
CNTRPTEN (R/W)
Count Repeat Enable
CNTEN (R/W)
Count Enable
LCMPEN (R/W)
Locked Comparison Enable
EN (R/W)
Enable Watchpoint
DIR (R/W)
Transaction Direction for Match
ACMPM (R/W)
Address Comparison Mode
IDCMPEN (R/W)
ID Comparison Enable
SCMPEN (R/W)
Secure Comparison Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
DATACNTMODE (R/W)
Data Channel Monitor
SCALE (R/W)
Scale the number of transactions
STALLCNTMODE (R/W)
Stall Count Mode
MAXACT (R/W)
Action for Bandwidth Above Maximum
BWEN (R/W)
Bandwidth Mode Enable
BLENINC (R/W)
Increment Bandwidth Count by Burst Length
MINACT (R/W)
Action for Bandwidth Below Minimum

Figure 55-4: SWU_CTL[n] Register Diagram

Table 55-7: SWU_CTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	DATACNTMODE	Data Channel Monitor. The `SWU_CTL[n].DATACNTMODE` bit determines whether an address channel or a data channel is monitored. Note that in data channel only ID,READY and VALID signals are monitored and hence other comparisons (Address, Lock, Secure) will be ignored even if enabled. `SWU_CTL[n].STALLCNTMODE` and `SWU_CTL[n].DIR` can be used in conjunction with this bit	
		0	Monitor address channel
		1	Monitor data channel
23:21 (R/W)	SCALE	Scale the number of transactions. The `SWU_CTL[n].SCALE` bit field allows a program to count more transactions by scaling the number of transactions and also the number of clock cycles in the bandwidth window (`SWU_CNT[n]` register) by 10,100 or 1000. This is applicable only in bandwidth mode (`SWU_CTL[n].BWEN`==1).	
		0	No scaling
		1	1:10
		2	1:100
		3	Reserved
		4	1:1000
		5-7	Reserved
20 (R/W)	STALLCNTMODE	Stall Count Mode. The `SWU_CTL[n].STALLCNTMODE` bit determines whether the number of stalls are counted or whether the number of transactions are counted. This feature is only valid in bandwidth mode.	
		0	Count number of transactions
		1	Count number of stalls
19 (R/W)	MAXACT	Action for Bandwidth Above Maximum. Each `SWU_CTL[n].MAXACT` bit determines whether a watchpoint group takes action on bandwidth overflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action
18 (R/W)	MINACT	Action for Bandwidth Below Minimum. Each `SWU_CTL[n].MINACT` bit determines whether a watchpoint group takes action on bandwidth underflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action

Table 55-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	BLENINC	Increment Bandwidth Count by Burst Length. Each `SWU_CTL[n].BLENINC` bit controls how a watchpoint group's bandwidth count is incremented in the `SWU_CUR[n]` register's `SWU_CUR[n].CURBW` field. If the `SWU_CTL[n].BLENINC` bit is cleared (= 0), the SWU increments the bandwidth count by 1 for each matching transaction. If the `SWU_CTL[n].BLENINC` bit is set (=1), the SWU increments the bandwidth count by the burst length of the transaction for each matching transaction. This feature is only valid for bandwidth mode (`SWU_CTL[n].BWEN` bit == 1). Note that if the address range match is enabled (`SWU_CTL[n].ACMPM` bits) and if any address of a burst falls within the address range, the `SWU_CUR[n].CURBW` field is incremented by the burst length even if some of the burst address fall outside of the range. Also, note that the burst size of the transaction is not included in the increment, only the burst length of the transaction. This increment operation provides an approximate (not exact) number of bus cycles consumed during the bandwidth.	
		0	Increment by 1
		1	Burst Length Increment for Bandwidth Count
16 (R/W)	BWEN	Bandwidth Mode Enable. Each `SWU_CTL[n].BWEN` bit controls whether a watchpoint group operates in watchpoint mode or bandwidth mode. In watchpoint mode, the `SWU_CTL[n].CNTEN` and (optionally) `SWU_CTL[n].CNTRPTEN` registers control usage of the cycle count for watchpoint group operations. In bandwidth mode, the `SWU_CTL[n].BLENINC`, `SWU_TARG[n]`, and `SWU_HIST[n]` registers control usage of watchpoint matches for watchpoint group operations.	
		0	Watchpoint Mode
		1	Bandwidth Mode
14 (R/W)	TRGEN	Trigger Enable. Each `SWU_CTL[n].TRGEN` bit controls whether a match for a watchpoint group generates a trigger event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
13 (R/W)	INTEN	Interrupt Enable. Each `SWU_CTL[n].INTEN` bit controls whether a match for a watchpoint group generates an interrupt. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable

Table 55-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DBGEN	Debug Event Enable. Each `SWU_CTL[n].DBGEN` bit controls debug event comparison for a watchpoint group, permitting matches based on debug status.	
		0	Disable
		1	Enable
9 (R/W)	CNTRPTEN	Count Repeat Enable. Each `SWU_CTL[n].CNTRPTEN` bit controls whether the watchpoint group's cycle count is reloaded and repeated after cycle countdown. If the `SWU_CTL[n]` register's `SWU_CTL[n].CNTRPTEN` bit is set, the `SWU_CUR[n]` register's `SWU_CUR[n].CURCNT` field is reloaded from `SWU_CNT[n]` register's `SWU_CNT[n].COUNT` field, and the countdown starts again. If `SWU_CTL[n].CNTRPTEN` bit is cleared, the expired count remains zero, and no further events are signalled. (See the `SWU_CTL[n].CNTEN` bit description for information regarding the countdown setup.)	
		0	Disable
		1	Enable
8 (R/W)	CNTEN	Count Enable. Each `SWU_CTL[n].CNTEN` bit controls whether the cycle count in the watchpoint group's `SWU_CNT[n]` register is decremented each cycle until it reaches zero. This feature is only valid in watchpoint mode (`SWU_CTL[n].BWEN` bit == 0).When the count reaches zero, any enabled watchpoint events are triggered. (See the `SWU_CTL[n].CNTRPTEN` bit description for optional actions at that may occur at the end of the countdown.)	
		0	Disable
		1	Enable
6 (R/W)	LCMPEN	Locked Comparison Enable. Each `SWU_CTL[n].LCMPEN` bit controls locked comparison operation of an SWU watchpoint group, permitting matches based on lock status.	
		0	Match on all transaction
		1	Match only locked transactions
5 (R/W)	SCMPEN	Secure Comparison Enable. Each `SWU_CTL[n].SCMPEN` bit controls secure transaction comparison operation of an SWU watchpoint group, permitting matches based on transaction security.	
		0	Match on all transaction
		1	Match only secure transactions

Table 55-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	IDCMPEN	ID Comparison Enable. Each `SWU_CTL[n].IDCMPEN` bit controls the ID comparison operation of an SWU watchpoint group. The ID match is based on comparison with the value in the `SWU_ID[n]` register.
3:2 (R/W)	ACMPM	Address Comparison Mode. Each set of `SWU_CTL[n].ACMPM` bits control the address comparison operation of an SWU watchpoint group. The address within range for comparison is defined as (`SWU_LA[n]` register <= address < `SWU_UA[n]` register). The address outside range for comparison is defined as (address < `SWU_LA[n]`) or (`SWU_UA[n]` <= address). 0 No address comparison 1 Exact match on LAn 2 Match on address within range 3 Match on address outside range
1 (R/W)	DIR	Transaction Direction for Match. Each `SWU_CTL[n].DIR` bit determines whether the SWU check reads or writes for watchpoint matches. 0 Match on reads only 1 Match on writes only
0 (R/W)	EN	Enable Watchpoint. Each `SWU_CTL[n].EN` bit controls the operation of one SWU watchpoint group. Clearing the `SWU_CTL[n].EN` bit halts the execution of watchpoint or bandwidth tracking operations in the watchpoint group without resetting status or configuration registers. Setting the `SWU_CTL[n].EN` bit enables the SWU watchpoint group to begin or resume operation with the current configuration and status. 0 Disable 1 Enable

Current Register n

The SWU current register (`SWU_CUR[n]`) operation varies depending whether the watchpoint group is in bandwidth mode or watchpoint mode. In both modes, the watchpoint count begins when the SWU loads the register's `SWU_CUR[n].CURCNT` field from the `SWU_CNT[n]` register's `SWU_CNT[n].COUNT` field when the watchpoint count is enabled (`SWU_CTL[n]` register, `SWU_CTL[n].CNTEN` bit =1).

In bandwidth mode, the current count field (`SWU_CUR[n].CURCNT`) contains the cycle count remaining within the current watchpoint period. The SWU decrements this value every cycle until the count reaches zero. At that point, the SWU reloads the `SWU_CUR[n].CURCNT` field from `SWU_CNT[n]` register's `SWU_CNT[n].COUNT` field. In bandwidth mode, the current bandwidth field (`SWU_CUR[n].CURBW`) contains the count of watchpoint matches (bandwidth) accumulated in the current watchpoint period.

In watchpoint mode, the current count field (`SWU_CUR[n].CURCNT`) contains the watchpoint match count remaining within the current watchpoint period. The SWU decrements this value with every watchpoint match until the count reaches zero. At that point, the SWU reloads the `SWU_CUR[n].CURCNT` field from `SWU_CNT[n]` register's `SWU_CNT[n].COUNT` field if the `SWU_CTL[n]` register's `SWU_CTL[n].CNTRPTEN` bit is set (=1). In watchpoint mode, the current bandwidth field (`SWU_CUR[n].CURBW`) is undefined.

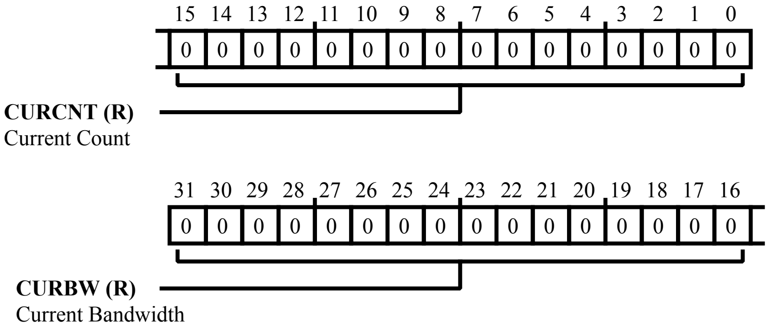


Figure 55-5: SWU_CUR[n] Register Diagram

Table 55-8: SWU_CUR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	CURBW	Current Bandwidth.
15:0 (R/NW)	CURCNT	Current Count.

Global Control Register

The SWU global control register (`SWU_GCTL`) provides SWU reset and enable.

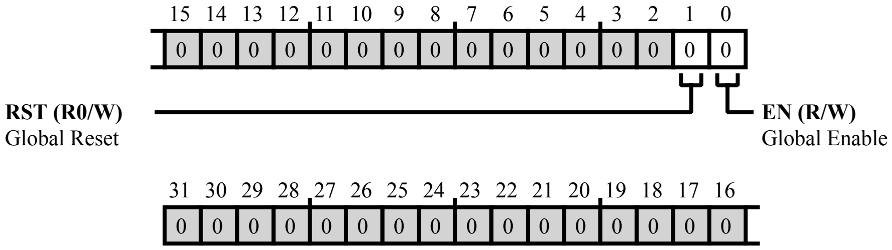

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RST (R0/W)
Global Reset
EN (R/W)
Global Enable
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 55-6: SWU_GCTL Register Diagram

Table 55-9: SWU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R0/W)	RST	Global Reset. The `SWU_GCTL.RST`) is write-1-action/read zero and controls the SWU operational state. Setting `SWU_GCTL.RST` resets all SWU registers to their default values and halts all SWU operations. 0: No Action 1: Reset
0 (R/W)	EN	Global Enable. The `SWU_GCTL.EN` controls the SWU operational state. Clearing `SWU_GCTL.EN` halts the execution of all watchpoint and bandwidth tracking operations without resetting status registers or associated signals. Setting `SWU_GCTL.EN` enables the SWU to begin/resume operation with the current configuration and status. 0: Disable 1: Enable

Global Status Register

The SWU global status register (`SWU_GSTAT`) contains status bits for all four watchpoint groups.

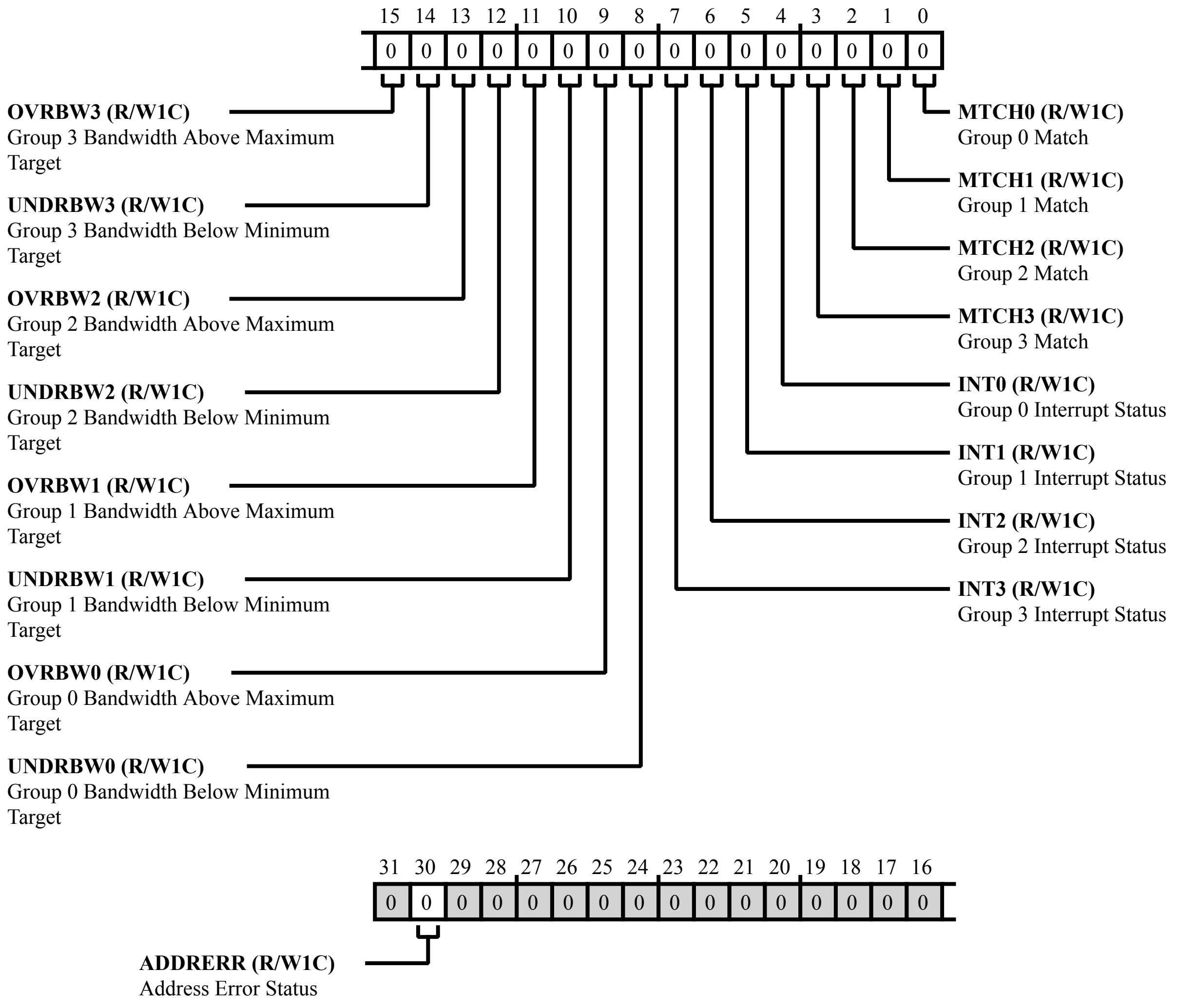

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OVRBW3 (R/W1C)
Group 3 Bandwidth Above Maximum Target
UNDRBW3 (R/W1C)
Group 3 Bandwidth Below Minimum Target
OVRBW2 (R/W1C)
Group 2 Bandwidth Above Maximum Target
UNDRBW2 (R/W1C)
Group 2 Bandwidth Below Minimum Target
OVRBW1 (R/W1C)
Group 1 Bandwidth Above Maximum Target
UNDRBW1 (R/W1C)
Group 1 Bandwidth Below Minimum Target
OVRBW0 (R/W1C)
Group 0 Bandwidth Above Maximum Target
UNDRBW0 (R/W1C)
Group 0 Bandwidth Below Minimum Target
MTCH0 (R/W1C)
Group 0 Match
MTCH1 (R/W1C)
Group 1 Match
MTCH2 (R/W1C)
Group 2 Match
MTCH3 (R/W1C)
Group 3 Match
INT0 (R/W1C)
Group 0 Interrupt Status
INT1 (R/W1C)
Group 1 Interrupt Status
INT2 (R/W1C)
Group 2 Interrupt Status
INT3 (R/W1C)
Group 3 Interrupt Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDRERR (R/W1C)
Address Error Status

Figure 55-7: SWU_GSTAT Register Diagram

Table 55-10: SWU_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W1C)	ADDRERR	Address Error Status. The `SWU_GSTAT.ADDRERR` indicates that the SWU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	Inactive
		1	Active

Table 55-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	OVRBW3	Group 3 Bandwidth Above Maximum Target. See `SWU_GSTAT.OVRBW0` description.	
		0	Group 3 was not above maximum bandwidth
		1	Group 3 was above maximum bandwidth
14 (R/W1C)	UNDRBW3	Group 3 Bandwidth Below Minimum Target. See `SWU_GSTAT.UNDRBW0` description.	
		0	Group 3 was not below minimum bandwitdth
		1	Group 3 was below minimum bandwitdth
13 (R/W1C)	OVRBW2	Group 2 Bandwidth Above Maximum Target. See `SWU_GSTAT.OVRBW0` description.	
		0	Group 2 was not above maximum bandwidth
		1	Group 2 was above maximum bandwidth
12 (R/W1C)	UNDRBW2	Group 2 Bandwidth Below Minimum Target. See `SWU_GSTAT.UNDRBW0` description.	
		0	Group 2 was not below minimum bandwidth
		1	Group 2 was below minimum bandwidth
11 (R/W1C)	OVRBW1	Group 1 Bandwidth Above Maximum Target. See `SWU_GSTAT.OVRBW0` description.	
		0	Group 1 was not above maximum bandwidth
		1	Group 1 was above maximum bandwidth
10 (R/W1C)	UNDRBW1	Group 1 Bandwidth Below Minimum Target. See `SWU_GSTAT.UNDRBW0` description.	
		0	Group 1 was not below minimum bandwidth
		1	Group 1 was below minimum bandwidth
9 (R/W1C)	OVRBW0	Group 0 Bandwidth Above Maximum Target. The `SWU_GSTAT.OVRBW0` - `SWU_GSTAT.OVRBW3` -- Group 0 through 3 watchpoint bandwidth over maximum target bits. Each maximum bandwidth bit indicate (for each group)s that the measured bandwidth over the period defined by the `SWU_CNT[n]` register was over the maximum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not above maximum bandwidth
		1	Group 0 was above maximum bandwidth

Table 55-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	UNDRBW0	Group 0 Bandwidth Below Minimum Target. The `SWU_GSTAT.UNDRBW0` - `SWU_GSTAT.UNDRBW3` -- Group 0 through 3 watchpoint bandwidth below minimum target bits. Each minimum bandwidth bit indicates (for each group) that the measured bandwidth over the period defined by the `SWU_CNT[n]` register was below the minimum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not below minimum bandwidth
		1	Group 0 was below minimum bandwidth
7 (R/W1C)	INT3	Group 3 Interrupt Status. See `SWU_GSTAT.INT0` description.	
		0	No Interrupt
		1	Interrupt Occurred
6 (R/W1C)	INT2	Group 2 Interrupt Status. See `SWU_GSTAT.INT0` description.	
		0	No Interrupt
		1	Interrupt Occurred
5 (R/W1C)	INT1	Group 1 Interrupt Status. See `SWU_GSTAT.INT0` description.	
		0	No Interrupt
		1	Interrupt Occurred
4 (R/W1C)	INT0	Group 0 Interrupt Status. The `SWU_GSTAT.INT0` - `SWU_GSTAT.INT3` -- Group 0 through 3 interrupt bits. Each interrupt bit indicates (for each group) whether a watchpoint group is contributing to the SWU's interrupt output. This status bit is sticky; write-1-to-clear it.	
		0	No interrupt
		1	Interrupt Occurred
3 (R/W1C)	MTCH3	Group 3 Match. See `SWU_GSTAT.MTCH0` description.	
		0	No Match
		1	Group 3 Watchpoint Match

Table 55-10: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MTCH2	Group 2 Match. See `SWU_GSTAT.MTCH0` description.	
		0	No match
		1	Group 2 Watchpoint Match
1 (R/W1C)	MTCH1	Group 1 Match. See `SWU_GSTAT.MTCH0` description.	
		0	No match
		1	Group 1 Watchpoint Match
0 (R/W1C)	MTCH0	Group 0 Match. The `SWU_GSTAT.MTCH0` - `SWU_GSTAT.MTCH3` -- Group 0 through 3 match bits. Each match bit indicates (for each group) whether a watchpoint match has occurred in a SWU watchpoint group, as controlled by the group's related watchpoint control register (`SWU_CTL[n]`). This status bit is sticky; write-1-to-clear it.	
		0	No match
		1	Group 0 Watchpoint Match

Bandwidth History Register n

The SWU bandwidth history registers (`SWU_HIST[n]`) contain data copied from a watchpoint group's current bandwidth value (`SWU_CUR[n]` register, `SWU_CUR[n].CURBW` bits) at the end of the last two watchpoint periods. At the end of each watchpoint period, the SWU copies the previous bandwidth value from the `SWU_HIST[n].BWHIST0` field to the `SWU_HIST[n].BWHIST1` field and copies the new bandwidth value from the `SWU_CUR[n].CURBW` field to the `SWU_HIST[n].BWHIST0` field.

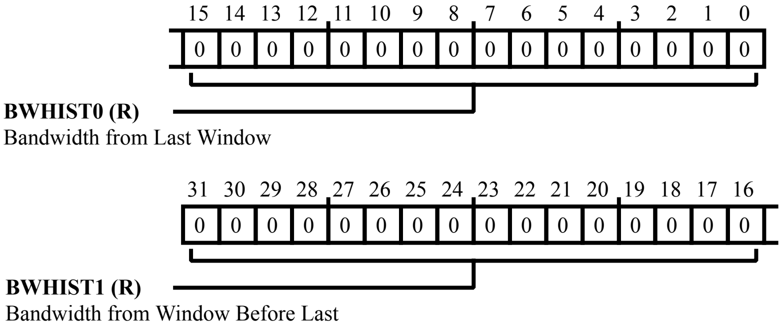

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BWHIST0 (R)
Bandwidth from Last Window
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BWHIST1 (R)
Bandwidth from Window Before Last

Figure 55-8: SWU_HIST[n] Register Diagram

Table 55-11: SWU_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	BWHIST1	Bandwidth from Window Before Last.
15:0 (R/NW)	BWHIST0	Bandwidth from Last Window.

ID Register n

The SWU ID registers (`SWU_ID[n]`) contain a 16-bit ID field (`SWU_ID[n].ID`) and a 16-bit ID mask field (`SWU_ID[n].IDMASK`) that watchpoint groups use for ID comparison. The ID on the bus is AND'ed with the `SWU_ID[n].IDMASK` field, then the watchpoint group compares the result against the `SWU_ID[n].ID` field.

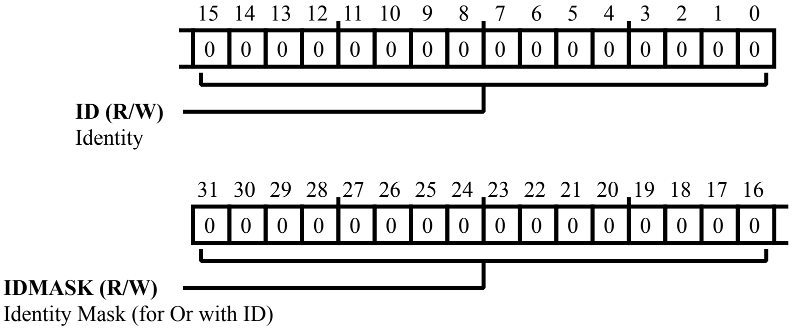

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ID (R/W)
Identity
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
IDMASK (R/W)
Identity Mask (for Or with ID)

Figure 55-9: SWU_ID[n] Register Diagram

Table 55-12: SWU_ID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	IDMASK	Identity Mask (for Or with ID).
15:0 (R/W)	ID	Identity.

Lower Address Register n

The SWU lower address registers (`SWU_LA[n]`) contain each watchpoint group's lower address for address match comparison. In exact match on `SWU_LA[n]` address mode (`SWU_CTL[n].ACMPM` bits =01), the watchpoint group uses only this address for match comparison.

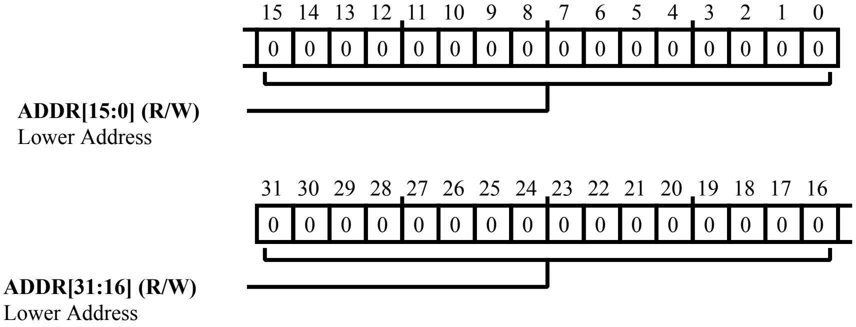

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R/W)
Lower Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R/W)
Lower Address

Figure 55-10: SWU_LA[n] Register Diagram

Table 55-13: SWU_LA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Lower Address.

Target Register n

The SWU target registers (`SWU_TARG[n]`) contain a minimum value field (`SWU_TARG[n].BWMIN`) and maximum value field (`SWU_TARG[n].BWMAX`) of bandwidth targets used by watchpoint groups in bandwidth mode. When the bandwidth period expires, if the current bandwidth value (`SWU_CUR[n]` register, `SWU_CUR[n].CURBW` bits) is below the minimum target or above the maximum target, the watchpoint group takes action as enabled by the `SWU_CTL[n]` register's `SWU_CTL[n].MINACT` or `SWU_CTL[n].MAXACT` bits.

In bandwidth mode, note that the watchpoint group increments its count of either data bus transactions or address bus transactions (bursts) as selected by the `SWU_CTL[n].BLENINC` bit. Keep this mode selection in mind when programming the bandwidth target values.

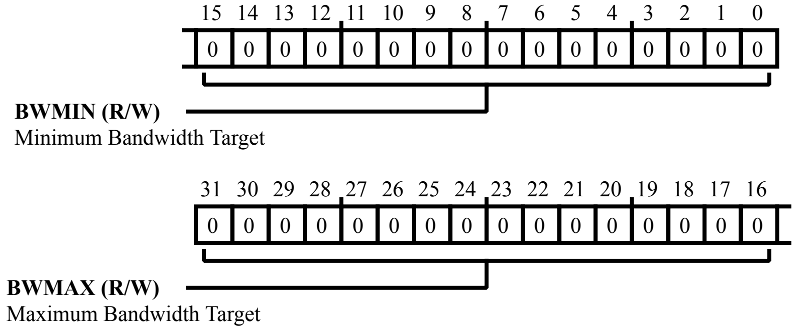

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BWMIN (R/W)
Minimum Bandwidth Target
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BWMAX (R/W)
Maximum Bandwidth Target

Figure 55-11: SWU_TARG[n] Register Diagram

Table 55-14: SWU_TARG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	BWMAX	Maximum Bandwidth Target.
15:0 (R/W)	BWMIN	Minimum Bandwidth Target.

Upper Address Register n

The SWU upper address registers (`SWU_UA[n]`) contain each watchpoint group's upper address for address match comparison. In exact match on `SWU_LA[n]` address mode (`SWU_CTL[n].ACMPM` bits =01), the `SWU_UA[n]` is not used for match comparison.

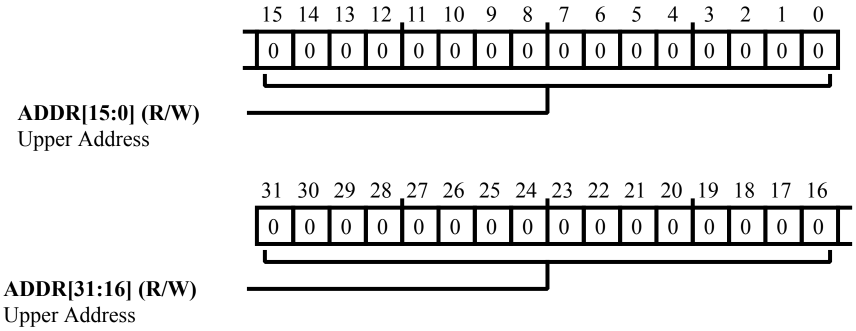

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R/W)
Upper Address
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R/W)
Upper Address

Figure 55-12: SWU_UA[n] Register Diagram

Table 55-15: SWU_UA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

56 System Debug and Trace Unit (DBG)

The system debug and trace unit is based on ARM Core Sight technology. CoreSight™ is a set of architecture specifications defining debug and trace architecture. The processor uses CoreSight infrastructure to provide industry standard debug and trace capabilities.

http://infocenter.arm.com/help/

The applicable documentation for more details about the ARM CoreSight feature includes:

- CoreSight PFT Architecture Specification , ARM IHI 0035B (PFT)
- System Trace Macrocell, Programmers' Model Architecture Specification, ARM IHI 0054A (STM)
- CoreSight Trace Memory Controller, ARM DDI0461B (TMC)
- CoreSight Components Technical Reference Manual, ARM DDI 0314H (TPIU)
- Embedded Cross Trigger Technical Reference Manual, ARM DDI 0291A
- ETM Architecture Specification, ARM IHI0014Q
- ETM for A5 Technical Reference Manual, DDI0435C

DBG Features

The system debug and trace unit contains the following features.

- System JTAG TAP controller for system debug features, boundary scan, and public JTAG features
- A debug interface to cores, and other system resources
- Direct and run-time access to the memory system and system MMRs
- Direct control over system reset
- Support for debug immediately after reset (boot debug)
- Group halt (debug event immediately halts all specified endpoints)
- Real-time on-chip visibility is made available to all developers, including software developers

DBG Functional Description

The following sections provide functional descriptions of the DBG unit.

ADSP-SC58x CSPFT Register List

Table 56-1: ADSP-SC58x CSPFT Register List

Name	Description
CSPFT_ACTR[n]	Address Comparator Access Type Register
CSPFT_ACVR[n]	Address Comparator Value Register
CSPFT_AUTHSTATUS	Authentication Status Register
CSPFT_CCER	Configuration Code Extension Register
CSPFT_CID0	Component ID0 Register
CSPFT_CID1	Component ID1 Register
CSPFT_CID2	Component ID2 Register
CSPFT_CID3	Component ID3 Register
CSPFT_CIDCMR	Context ID Comparator Mask Register
CSPFT_CIDCVR[n]	Context ID Comparator Value
CSPFT_CLAIMCLR	Claim Tag Clear Register
CSPFT_CLAIMSET	Claim Tag Set Register
CSPFT_CNTENR[n]	Counter Enable Event Register
CSPFT_CNTRLDEVR[n]	Counter Reload Event Register
CSPFT_CNTRLDVR[n]	Counter Reload Value Register
CSPFT_CNTVR[n]	Counter Value Register
CSPFT_CTL	Main Control Register
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register

Table 56-1: ADSP-SC58x CSPFT Register List (Continued)

Name	Description
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

ADSP-SC58x TAPC Register List

The Test Access Port Controller (TAPC) provides access to debug features. A set of registers governs TAPC operations. For more information on TAPC functionality, see the TAPC register descriptions.

Table 56-2: ADSP-SC58x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control Register
TAPC_IDCODE	IDCODE Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register
TAPC_SDBGKEY1	Secure Debug Key 1 Register
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

ADSP-SC58x TAPC Register List

The Test Access Port Controller (TAPC) provides access to debug features. A set of registers governs TAPC operations. For more information on TAPC functionality, see the TAPC register descriptions.

Table 56-3: ADSP-SC58x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control Register
TAPC_IDCODE	IDCODE Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register

Table 56-3: ADSP-SC58x TAPC Register List (Continued)

Name	Description
TAPC_SDBGKEY1	Secure Debug Key 1 Register
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

ADSP-SC58x STM Trigger List

Table 56-4: ADSP-SC58x STM Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 56-5: ADSP-SC58x STM Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
52	STM0_EVT0	STM0 STM0 Event 0	Edge
53	STM0_EVT1	STM0 STM0 Event 1	Edge
54	STM0_EVT2	STM0 STM0 Event 2	Edge
55	STM0_EVT3	STM0 STM0 Event 3	Edge
56	STM0_EVT4	STM0 STM0 Event 4	Edge
57	STM0_EVT5	STM0 STM0 Event 5	Edge
58	STM0_EVT6	STM0 STM0 Event 6	Edge
59	STM0_EVT7	STM0 STM0 Event 7	Edge
60	STM0_EVT8	STM0 STM0 Event 8	Edge
61	STM0_EVT9	STM0 STM0 Event 9	Edge
62	STM0_EVT10	STM0 STM0 Event 10	Edge
63	STM0_EVT11	STM0 STM0 Event 11	Edge
64	STM0_EVT12	STM0 STM0 Event 12	Edge
65	STM0_EVT13	STM0 STM0 Event 13	Edge
66	STM0_EVT14	STM0 STM0 Event 14	Edge
67	STM0_EVT15	STM0 STM0 Event 15	Edge
68	STM0_EVT16	STM0 STM0 Event 16	Edge

Table 56-5: ADSP-SC58x STM Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
69	STM0_EVT17	STM0 STM0 Event 17	Edge
70	STM0_EVT18	STM0 STM0 Event 18	Edge
71	STM0_EVT19	STM0 STM0 Event 19	Edge
72	STM0_EVT20	STM0 STM0 Event 20	Edge
73	STM0_EVT21	STM0 STM0 Event 21	Edge
74	STM0_EVT22	STM0 STM0 Event 22	Edge
75	STM0_EVT23	STM0 STM0 Event 23	Edge
76	STM0_EVT24	STM0 STM0 Event 24	Edge
77	STM0_EVT25	STM0 STM0 Event 25	Edge
78	STM0_EVT26	STM0 STM0 Event 26	Edge
79	STM0_EVT27	STM0 STM0 Event 27	Edge
80	STM0_EVT28	STM0 STM0 Event 28	Edge
81	STM0_EVT29	STM0 STM0 Event 29	Edge
82	STM0_EVT30	STM0 STM0 Event 30	Edge
83	STM0_EVT31	STM0 STM0 Event 31	Edge

ADSP-SC58x CTI Trigger List

Table 56-6: ADSP-SC58x CTI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
84	CTI3_MST0	CTI3 SYSCTI (CTI3) System Halt Slave 0	Edge
85	CTI3_MST1	CTI3 SYSCTI (CTI3) System Halt Slave 1	Edge
86	CTI3_MST2	CTI3 SYSCTI (CTI3) System Halt Slave 2	Edge
87	CTI3_MST3	CTI3 SYSCTI (CTI3) System Halt Slave 3	Edge
88	CTI3_MST4	CTI3 SYSCTI (CTI3) System Halt Slave 4	Edge
89	CTI3_MST5	CTI3 SYSCTI (CTI3) System Halt Slave 5	Edge
90	CTI3_MST6	CTI3 SYSCTI (CTI3) System Halt Slave 6	Edge
91	CTI3_MST7	CTI3 SYSCTI (CTI3) System Halt Slave 7	Edge

Table 56-7: ADSP-SC58x CTI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
110	CTI3_SLV0	CTI3 SYSCTI System Halt Master 0	

Table 56-7: ADSP-SC58x CTI Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
111	`CTI3_SLV1`	CTI3 SYSCTI System Halt Master 1	
112	`CTI3_SLV2`	CTI3 SYSCTI System Halt Master 2	
113	`CTI3_SLV3`	CTI3 SYSCTI System Halt Master 3	
114	`CTI3_SLV4`	CTI3 SYSCTI System Halt Master 4	
115	`CTI3_SLV5`	CTI3 SYSCTI System Halt Master 5	
116	`CTI3_SLV6`	CTI3 SYSCTI System Halt Master 6	
117	`CTI3_SLV7`	CTI3 SYSCTI System Halt Master 7	

ADSP-SC58x CTI Interrupt List

Table 56-8: ADSP-SC58x CTI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
127	CTI1_EVT0	CTI1 Core 1 CTI Event		
128	CTI2_EVT0	CTI2 Core 2 CTI Event		
252	CTI0_EVT0	CTI0 Core 0 CTI Event		

DBG Block Diagram

The block diagram is shown below.

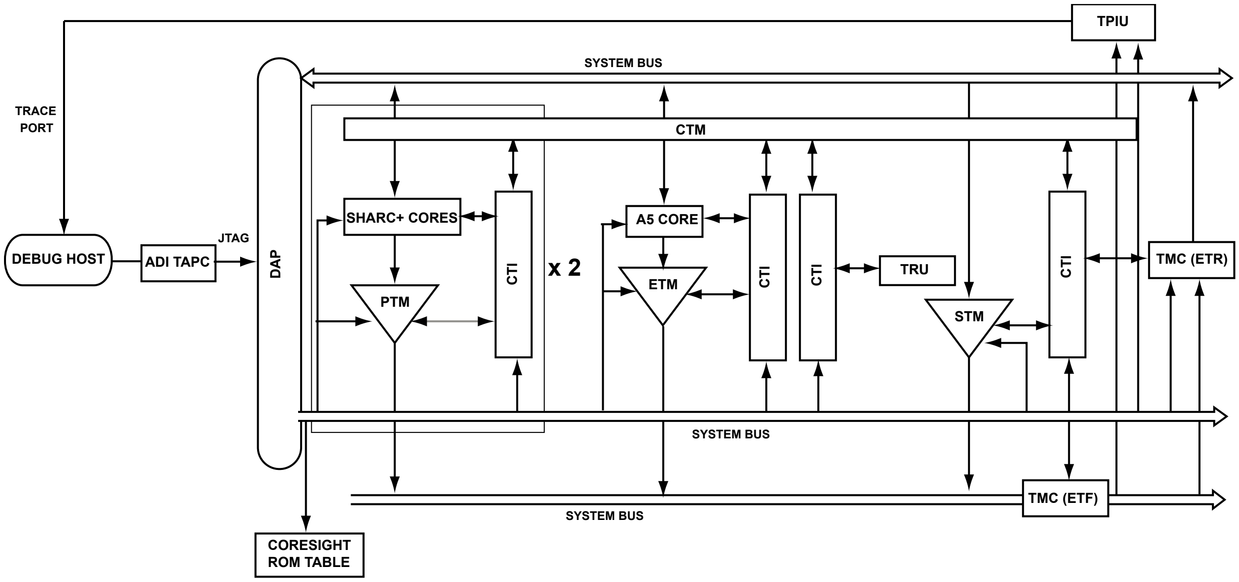

TPIU
SYSTEM BUS
TRACE PORT
CTM
SHARC+ CORES
A5 CORE
JTAG
DEBUG HOST
ADI TAPC
DAP
CTI
x 2
ETM
CTI
CTI
TRU
CTI
TMC (ETR)
PTM
STM
SYSTEM BUS
SYSTEM BUS
TMC (ETF)
CORESIGHT ROM TABLE

Figure 56-1: ADSP-SC5xx Block Diagram

DBG Definitions

The following terms are useful when working with the debug features of the processor and programming tools.

Test Access Port Controller (ADI TAPC)

Provides IDCODE and SDBGKEY features.

Debug Access Port (DAP)

Core Sight Interface providing a single port for two debug options: JTAG-DP (JTAP Debug Port), SW-DP (Serial Wire Debug Port)

Embedded Trace Macrocell (ETM)

Provides ARM Core Trace.

Program Trace Macrocell (PTM)

Provides DSP (SHARC +)Core Trace.

Standard Trace Macrocell (STM)

Provides capability to trace up to 32 hardware events and supports 32 software stimulus for data transfer between the user and emulator.

Embedded Cross Trigger (ECT)

The ECTs are responsible managing events and triggers as follows.

- CTI (Cross Trigger Interface) is a CoreSight component for enabling cross triggering of events across a system. From the view of the ECT, it is responsible for combining and mapping trigger requests.
- CTM (Cross Trigger Matrix) is a CoreSight component for connecting multiple CTIs. From the view of the ECT, it is responsible for connecting CTIs and distribution of events.

NOTE: An embedded cross trigger is not the same as the master and slave trigger in the trigger routing unit.

Trace Capture Devices

The trace capture devices capture and format the trace data. There are two trace capture devices in the system:

- ETF - Embedded Trace Funnel - Provides a buffer for burst trace data.
- ETR - Embedded Trace Router - Provides interface for trace data to be stored in system memories.

Trace Port Interface Unit (TPIU)

The TPIU acts as a bridge between the on-chip trace data, with separate IDs, to a data stream. It encapsulates IDs, when required, that the Trace Port Analyzer (TPA) captures.

Serial Wire Output (SWO)

SWO is a trace data drain that acts as a bridge between the on-chip trace data and a data stream that the Trace Port Analyzer (TPA) captures.

Test Access Port Controller (TAPC)

TAPC is an independent component daisy-chained to the DAP in the JTAG scan path. The TAPC component provides the product IDCODE (Chip ID) and a security feature.

The TAPC component provides security features to the chip using a debug key match features. This feature permits only those components which have the SDBGKEY to connect and debug the chip. Initially, a 128-bits user key is programmed in the SDBGKEY registers (`TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, `TAPC_SDBGKEY3`) in TAPC by a secure master. The `TAPC_SDBGKEY_CTL.VALID` bit is set.

Then, a user key is entered through the emulator for a match to the initially programmed-entered user key. On a successful match, the chip connects to the emulator. It can be debugged on a failed user key match. Further attempts at keys matching are disabled. JTAG reset or system reset is required to reenable the user key match logic.

Embedded Trace Macrocell (ETM)

ETM is the standard trace support provided for the A5 processor. For details of programming and functionality, refer to the ARM documentation.

Debug Access Ports

The Debug Access Port (DAP) is an implementation of an ARM Debug Interface version 5 (ADIv5) comprising a number of components supplied in a single configuration. All the supplied components fit into the various architectural components for Debug Ports (DPs). The components are used to access the DAP from an external debugger and APs, to access on-chip system resources.

The DAP provides access to all debug and trace capabilities through a single external interface. DAP has some additions to support BSCA and IDCODE features that Core Sight DAP does not support. The DAP provides a combined single debug interface port called SWJ_DP that includes:

- JTAG Debug Port (JTAG-DP). The JTAG-DP is based on the IEEE 1149.1 Test Access Port (TAP) and Boundary Scan Architecture
- Serial Wire Debug Port (SW-DP). The serial wire debug port provides a bidirectional serial connection to the ARM debug interface

The SWJ-DP is a combined JTAG-DP and SW-DP that allows connections to either a Serial Wire Debug (SWD) or JTAG probe to a target. It is the standard CoreSight debug port, and enables access either to the JTAG-DP or SW-DP blocks. The JTAG-DP is selected by default.

Trace Unit

The trace module provides instruction, data tracing, and system activity tracing for the processor. The program trace module on the processor is similar to the embedded trace macrocell module provided by the ARM processor. System trace is provided using the system trace macrocell as part of the CoreSight debug and trace interface. The trace module uses an interface based on the AMBA Trace Bus (ATB) standard to output its trace data. The trace data can be either exported to an off-chip trace port analyzer or captured on an on-chip buffer. The PFT and STM modules capture information on the processor both before and after a specific event. The modules add no burden to the processor performance when it runs at full speed.

- Programmable Flow Trace (CSPFT)
- System Trace Module (STM)
- Embedded Trace Macrocell (ETM)

Programmable Flow Trace (CSPFT)

When tracing processor execution, trace information can be generated for every instruction the processor executes. This information would be easy to interpret, but would require a prohibitively high trace bandwidth to get the trace data off the chip. With program flow tracing, only branch points are traced. The debugger uses the source code to infer the rest of the executed code.

Certain instructions in the program and events are identified as waypoints. A waypoint is a point where instruction execution involves a change of program flow. The CSPFT only traces those waypoints. These waypoints are:

- All indirect branches
- All direct branches
- Exceptions or interrupts
- Emulator debug entry and exit

When a waypoint occurs, trace data is generated to describe it. From this data and the source code, a trace decompressor can determine what instructions were executed and recreate the instruction flow. To allow the decompressor to calculate where it is in the source code, conditional instructions are marked as waypoints, regardless of whether they pass or fail their condition test. Events like interrupts or debug entry and exit can be promoted from non-waypoint instructions to waypoints to trace the interrupted program flow.

Tracing a waypoint implies the execution of all instructions from the target address of the previous waypoint up to the current waypoint. Non-waypoint instructions are not explicitly traced but the debugger must infer them using the source code. The concept of an instruction block is used throughout this manual and refers to the contiguous block of instructions between two waypoints.

The programming model and function is a subset of the PTM (Programmable Trace Module) of ARM.

System Trace Module (STM)

The STM is a trace source that is integrated into a CoreSight system, and is designed primarily for high-bandwidth trace of instrumentation embedded into software. The STM enables tracing of system activity from various sources:

- Instrumented software, using memory-mapped stimulus ports
- Hardware events

The STM supports the following features:

- Multiple software masters writing software instrumentation independently. Each master can use multiple stimulus ports.
- Time stamping of the system activity. The time stamp is a global time stamp which can be shared with other trace sources in the system to enable correlation of activity from multiple trace sources.
- Indication that specific events have occurred, such as a particular hardware event or a piece of software instrumentation. These events are known as triggers and can be indicated in the trace stream, or through signals to other system components.

Thirty-two hardware event resources are connected as output from the TRU which allows monitoring all of the hardware events that can generate a trigger.

Embedded Cross Trigger (ECT)

ECT provides an interface to the CoreSight debug system enabling the subsystems to interact (cross trigger) with each other. ECT provides a mechanism to forward debug events from one connected subsystem to another connected subsystem. The different subsystems connected to the ECT depend on the processor design. For example, in a multiprocessor system, the interface can be connected to each of the cores and one to the trace subsystem. For a uniprocessor system, the interface can include just the core and trace subsystem connection.

- CTI Cross trigger interface. A CoreSight component for enabling cross triggering of events across a system.
- CTM Cross trigger matrix. A CoreSight component for connecting multiple cross trigger interfaces.

The main function of the ECT (CTI and CTM) is to pass debug events from one connected subsystem to another connected subsystem.

For example, the ECT can communicate debug state information from the core to trace subsystem for a single processor system or to another core in a multiprocessor-based system. Program execution on both the subsystem can be stopped at the same time.

The ***Trigger Flow*** figure shows a simple debug trigger flow sequence. On each CTI, there are four channel, eight input, and eight output debug triggers. All the eight inputs and outputs can be mapped to a single channel or different channels based on the debug trigger to channel mapping. When a trigger input occurs, it creates a channel event. The channel event causes all the output debug triggers to be triggered. The embedded cross trigger depends on the debug trigger it connects to.

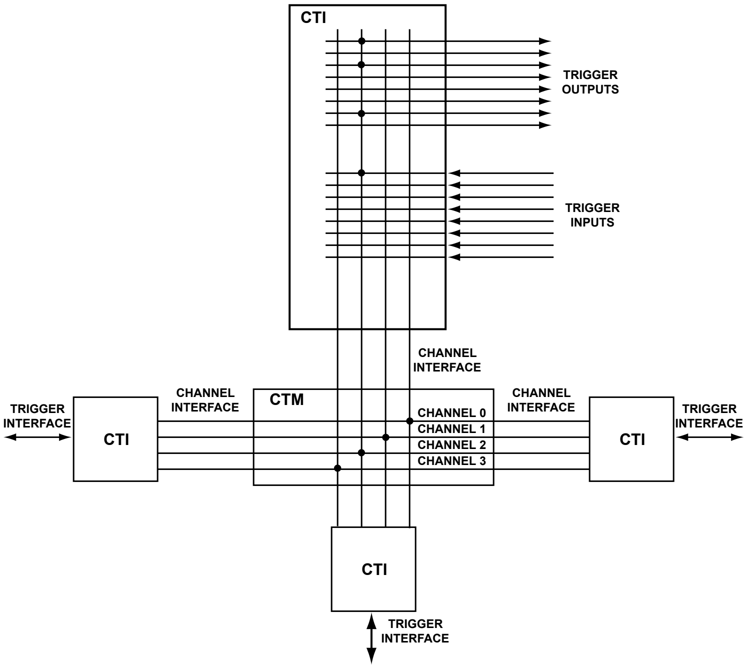

CTI
TRIGGER OUTPUTS
TRIGGER INPUTS
CHANNEL INTERFACE
TRIGGER INTERFACE
CTI
CHANNEL INTERFACE
CTM
CHANNEL 0
CHANNEL 1
CHANNEL 2
CHANNEL 3
CHANNEL INTERFACE
CTI
TRIGGER INTERFACE
CTI
TRIGGER INTERFACE

Figure 56-2: Trigger Flow

Refer to the ***ECT Integration*** figure. There are 5 CTIs in the system that connect to the core, trace, and system module, respectively. The CTIs all interconnect through the CTM. This configuration allows the core to trigger debug events on the trace, on the system and on the core itself. CTI0 handles all the ETM and core0 debug triggers. CTI1 handles all the PFT and core1 debug triggers. CTI2 handles all the PFT and core2 debug triggers. CTI3 handles all the system debug triggers. CTI4 handles all the trace components debug triggers.

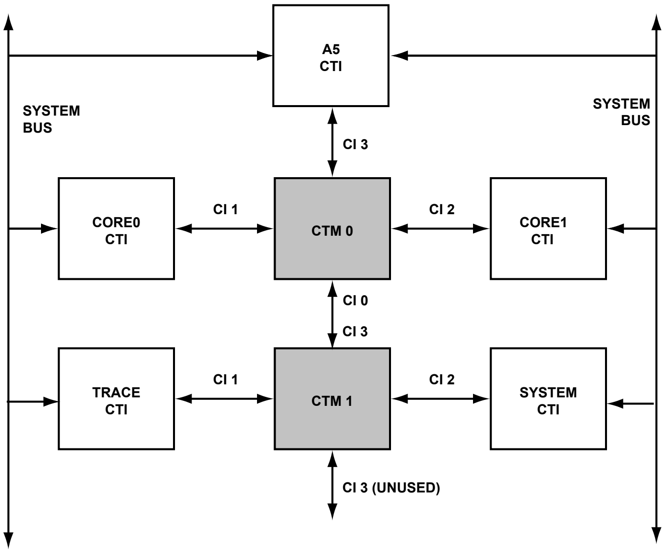

A5
CTI
SYSTEM BUS
SYSTEM BUS
CI 3
CORE0 CTI
CI 1
CTM 0
CI 2
CORE1 CTI
CI 0
CI 3
TRACE CTI
CI 1
CTM 1
CI 2
SYSTEM CTI
CI 3 (UNUSED)

Figure 56-3: ECT Integration

CTI Debug Trigger Tables

The ***System CTI Trigger Connection*** tables show the debug trigger that connects to each CTI.

Table 56-9: System CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	From TRU	CTITRIGOUT[7]	To TRU and also as SYS_DBGRESTART
CTITRIGIN [6:2]	From TRU	CTITRIGOUT[6:2]	To TRU
CTITRIGIN [1]	From TRU	CTITRIGOUT[1]	To TRU and also as peripheral halt
CTITRIGIN[0]	From TRU	CTITRIGOUT[0]	To TRU and also as system fabric halt

Table 56-10: Trace CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	STM ASYNCOUT	CTITRIGOUT[7]	Unused
CTITRIGIN[6]	STM TRIGOUTHETE	CTITRIGOUT [6]	TPIU FLUSHIN
CTITRIGIN[5]	STM TRIGOUTSW	CTITRIGOUT [5]	TPIU TRIGIN
CTITRIGIN[4]	STM TRIGOUTSPTE	CTITRIGOUT [4]	ETR FLUSHIN
CTITRIGIN[3]	ETR FULL	CTITRIGOUT [3]	ETR TRIGIN
CTITRIGIN[2]	ETR ACQCOMP	CTITRIGOUT [2]	ETF FLUSHIN
CTITRIGIN[1]	ETF FULL	CTITRIGOUT [1]	ETF TRIGIN
CTITRIGIN[0]	ETF ACQCOMP	CTITRIGOUT [0]	Unused

Table 56-11: SC5xx- Core 1/2 CTI Trace Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	Tied Low	CTITRIGOUT[7]	DBGRESTART
CTITRIGIN[6]	PTM TRIGGER	CTITRIGOUT[6]	SEC
CTITRIGIN[5:2]	PTM EXTOUT[3:0]	CTITRIGOUT[5:2]	PTM EXTIN[3:0]
CTITRIGIN[1]	Tied Low	CTITRIGOUT[1]	Unused
CTITRIGIN[0]	DBGTRIGGER	CTITRIGOUT[0]	EDBGRQ

Table 56-12: SC5xx ARM Core CTI Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	Tied Low	CTITRIGOUT[7]	DBGRESTART
CTITRIGIN[6]	ETM TRIGGER	CTITRIGOUT[6]	SEC
CTITRIGIN[5]	COMMRX	CTITRIGOUT[5]	Unused
CTITRIGIN[4]	COMMTX	CTITRIGOUT[4]	ETM EXTIN[3]

Table 56-12: SC5xx ARM Core CTI Connection (Continued)

CTI Port	Input	CTI Port	Output
CTITRIGIN[3:2]	ETMEXTOUT[1:0]	CTITRIGOUT[3:2]	ETM EXTIN[2:1]
CTITRIGIN[1]	PMU IRQ	CTITRIGOUT[1]	ETM EXTIN[0]
CTITRIGIN[0]	DBGTRIGGER	CTITRIGOUT[0]	EDBGRQ

Table 56-13: Trigger Descriptions

Signal Name	Description
STM ASYNCOUT	Alignment synchronization output. This signal is asserted for one clock cycle when an ASYNC-VERSION-FREQ sequence is completely output on the ATB, and can be used for cross-triggering.
STM TRIGOUTHETE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMHETER.
STM TRIGOUTSW	Trigger output. This signal is asserted for one clock cycle when a trigger event is generated on writes to a TRIG location in the extended stimulus port registers.
STM TRIGOUTSPTE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMSPTER.
ETR / ETF FULL	This output indicates the value of the full bit in the ETR/ETF status register. A full bit indicates the amount of data in ETF/ETR. An output signal indicating when the circular buffer or FIFO is full, or within a programmable amount of being full.
ETR / ETF ACQCOMP	This output indicates the value of the FtEmpty bit in the ETR/ETF status register. The bit it is set when trace capture has stopped. An output signal indicating when trace capture has stopped, usually following a trigger condition.
ETR/ETF/ TPIU TRIGIN	This input can cause a trigger event (Start /Stop Trigger). An input signal indicating when a trigger condition has occurred.
ETR/ETF /TPIU FLUSHIN	This input can cause a Trace flush. An input signal indicating a flush request
PTM/ETM TRIGGER	Trigger Input. Trigger event specifies the conditions that must be met to generate a trigger.
PTM/ETM EXTOUT[3:0]	PTM Output.
PTM/ETM EXTIN[3:0]	PTM Input
SEC	CTI Interrupt
DBGRESTART	This is an output from the CTI to a processor core or to system to return from debug mode.
DBGTRIGGER	This is a processor core output signal indicating that the core has moved to debug mode. If the CTIs are setup for synchronous halt, it will generate EDBGRQ to everyone else.

Table 56-13: Trigger Descriptions (Continued)

Signal Name	Description
EDBGRQ	This is an output from CTI and an input (as debug halt) to a processor core and to the system in general. It can assert as a result of another core going to emulation space (DBGTRIGGER) or by setting the corresponding bit in the CTI.
SYS_DBGRESTART	This is an output from the CTI to system to return from debug mode.
To TRU	TRU Master Event
From TRU	TRU Slave Event
COMMRX	The COMMRX output signals enable interrupt-driven communications over the DTR
COMMTX	The COMMTX output signals enable interrupt-driven communications over the DTR
PMU IRQ	Performance Monitor Unit on ARM Interrupt Signal

ADSP-SC58x CSPFT Register Descriptions

Program Flow Trace (CSPFT) contains the following registers.

Table 56-14: ADSP-SC58x CSPFT Register List

Name	Description
`CSPFT_ACTR[n]`	Address Comparator Access Type Register
`CSPFT_ACVR[n]`	Address Comparator Value Register
`CSPFT_AUTHSTATUS`	Authentication Status Register
`CSPFT_CCER`	Configuration Code Extension Register
`CSPFT_CID0`	Component ID0 Register
`CSPFT_CID1`	Component ID1 Register
`CSPFT_CID2`	Component ID2 Register
`CSPFT_CID3`	Component ID3 Register
`CSPFT_CIDCMR`	Context ID Comparator Mask Register
`CSPFT_CIDCVR[n]`	Context ID Comparator Value
`CSPFT_CLAIMCLR`	Claim Tag Clear Register
`CSPFT_CLAIMSET`	Claim Tag Set Register
`CSPFT_CNTENR[n]`	Counter Enable Event Register
`CSPFT_CNTRLDEVR[n]`	Counter Reload Event Register
`CSPFT_CNTRLDVR[n]`	Counter Reload Value Register
`CSPFT_CNTVR[n]`	Counter Value Register
`CSPFT_CTL`	Main Control Register

Table 56-14: ADSP-SC58x CSPFT Register List (Continued)

Name	Description
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

Address Comparator Access Type Register

The `CSPFT_ACTR[n]` register specifies whether the context ID needs to match.

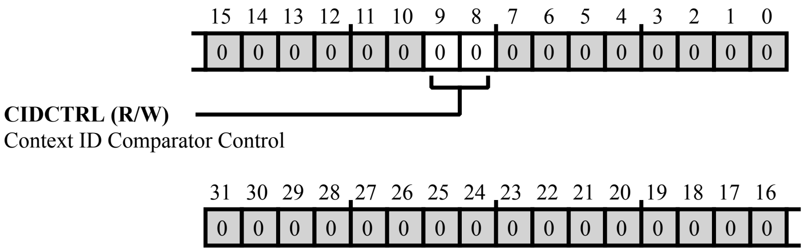

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CIDCTRL (R/W)
Context ID Comparator Control
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-4: CSPFT_ACTR[n] Register Diagram

Table 56-15: CSPFT_ACTR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	CIDCTRL	Context ID Comparator Control. The `CSPFT_ACTR[n].CIDCTRL` contains the ID comparator control value.	
		0	Ignore Context ID
		1	Match if Context ID Comparator 0 Matches
		2	Match if Context ID Comparator 1 Matches
		3	Match if Context ID Comparator 2 Matches

Address Comparator Value Register

The `CSPFT_ACVR[n]` register holds an address for comparison.

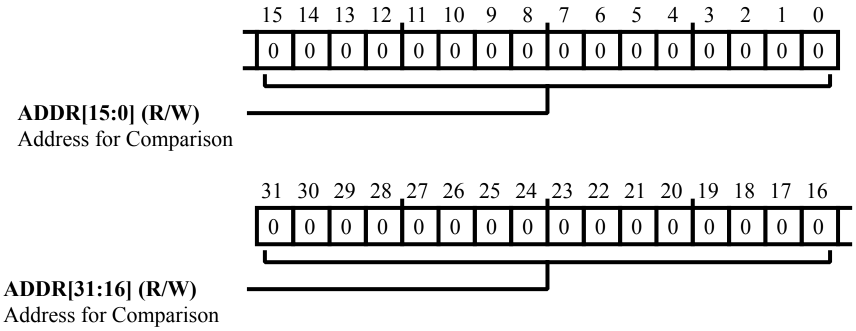

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[15:0] (R/W)
Address for Comparison
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ADDR[31:16] (R/W)
Address for Comparison

Figure 56-5: CSPFT_ACVR[n] Register Diagram

Table 56-16: CSPFT_ACVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address for Comparison. The `CSPFT_ACVR[n].ADDR` bit field contains the address used for comparison.

Authentication Status Register

The `CSPFT_AUTHSTATUS` register reports the level of tracing currently permitted based on the DBGEN signal.

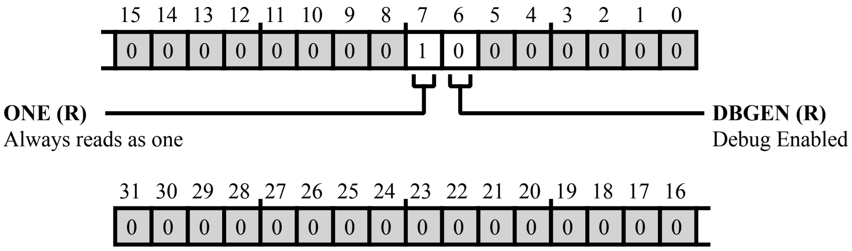

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
ONE (R)
Always reads as one
DBGEN (R)
Debug Enabled
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-6: CSPFT_AUTHSTATUS Register Diagram

Table 56-17: CSPFT_AUTHSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	ONE	Always reads as one.
6 (R/NW)	DBGEN	Debug Enabled. The `CSPFT_AUTHSTATUS.DBGEN` bit indicates that invasive debug is enabled. Normally, NIDEN is used in conjunction with a signal that enables invasive debug, DBGEN. Non-invasive debug is disabled only if both NIDEN and DBGEN signals are LOW. In a PTM, typically these signals are ORed together and the result is used to determine whether non-invasive debug is enabled.

Configuration Code Extension Register

The `CSPFT_CCER` register holds extra feature information. (See `CSPFT_HWFEAT`.)

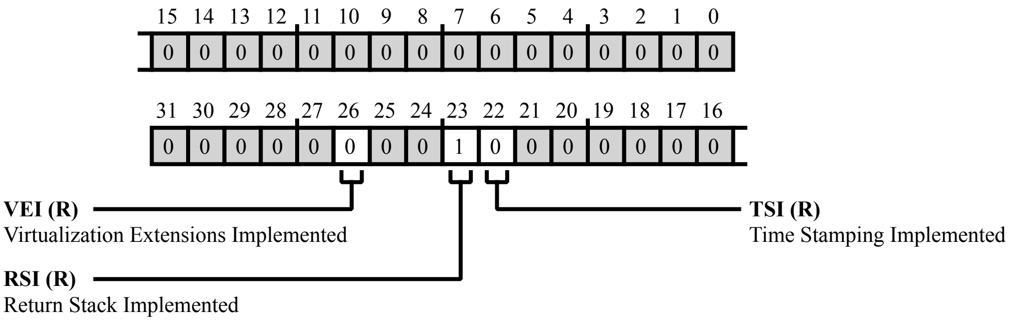

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
VEI (R)
Virtualization Extensions Implemented
RSI (R)
Return Stack Implemented
TSI (R)
Time Stamping Implemented

Figure 56-7: CSPFT_CCER Register Diagram

Table 56-18: CSPFT_CCER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	VEI	Virtualization Extensions Implemented. The `CSPFT_CCER.VEI` bit indicates if the virtualization extensions are implemented	
		0	Not Implemented
		1	Implemented
23 (R/NW)	RSI	Return Stack Implemented. The `CSPFT_CCER.RSI` bit indicates if a return stack is implemented.	
		0	Not Implemented
		1	Implemented
22 (R/NW)	TSI	Time Stamping Implemented. The `CSPFT_CCER.TSI` bit indicates if time stamping is implemented.	
		0	Disabled
		1	Enabled

Component ID0 Register

The `CSPFT_CID0` register holds sections of the CoreSight Component ID for CSPFT.

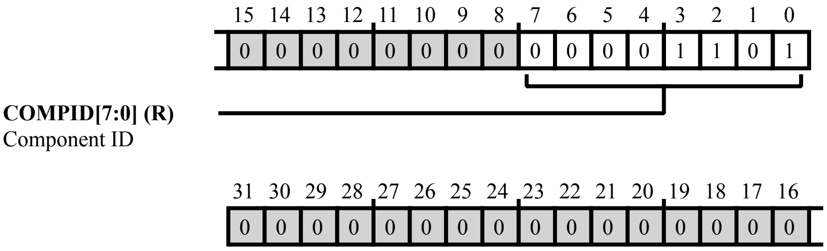

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1
COMPID[7:0] (R)
Component ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-8: CSPFT_CID0 Register Diagram

Table 56-19: CSPFT_CID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID0.COMPID` bit field identifies this component as a CoreSight component.

Component ID1 Register

The `CSPFT_CID1` register holds sections of the CoreSight Component ID for CSPFT.

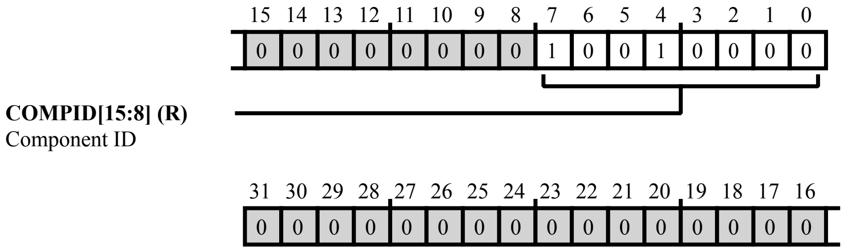

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0
COMPID[15:8] (R)
Component ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-9: CSPFT_CID1 Register Diagram

Table 56-20: CSPFT_CID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID1.COMPID` bit field identifies this component as a CoreSight component.

Component ID2 Register

The `CSPFT_CID2` register holds sections of the CoreSight Component ID for CSPFT.

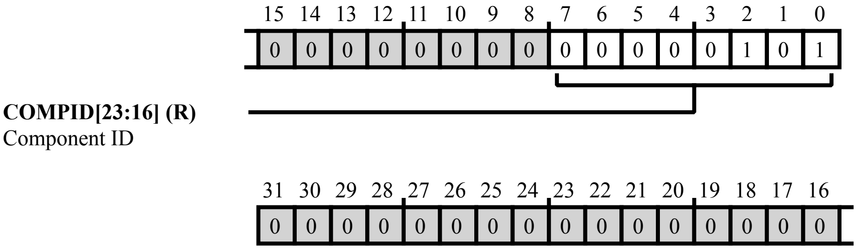

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
COMPID[23:16] (R)
Component ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-10: CSPFT_CID2 Register Diagram

Table 56-21: CSPFT_CID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID2.COMPID` bit field identifies this component as a CoreSight component.

Component ID3 Register

The `CSPFT_CID3` register holds sections of the CoreSight Component ID for CSPFT.

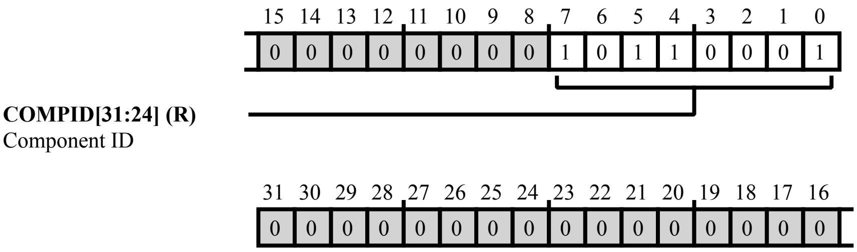

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1
COMPID[31:24] (R)
Component ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-11: CSPFT_CID3 Register Diagram

Table 56-22: CSPFT_CID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID3.COMPID` bit field identifies this component as a CoreSight component.

Context ID Comparator Mask Register

The `CSPFT_CIDCMR` register holds a 32-bit mask for use for all context ID comparisons.

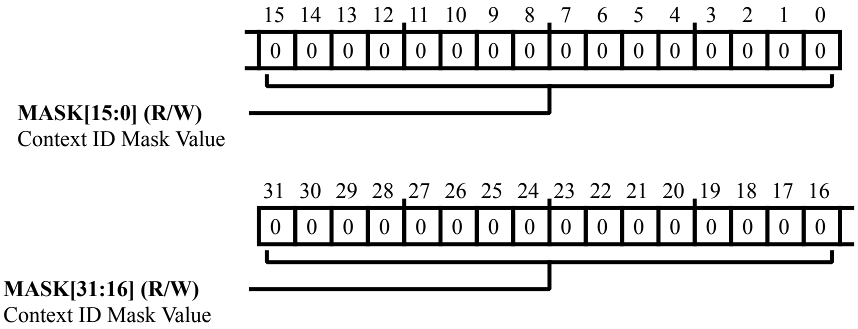

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MASK[15:0] (R/W)
Context ID Mask Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
MASK[31:16] (R/W)
Context ID Mask Value

Figure 56-12: CSPFT_CIDCMR Register Diagram

Table 56-23: CSPFT_CIDCMR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MASK	Context ID Mask Value. The `CSPFT_CIDCMR.MASK` bit field holds a 32-bit mask for use in all context ID comparisons.

Context ID Comparator Value

The `CSPFT_CIDCVR[n]` register holds a context ID value for comparison.

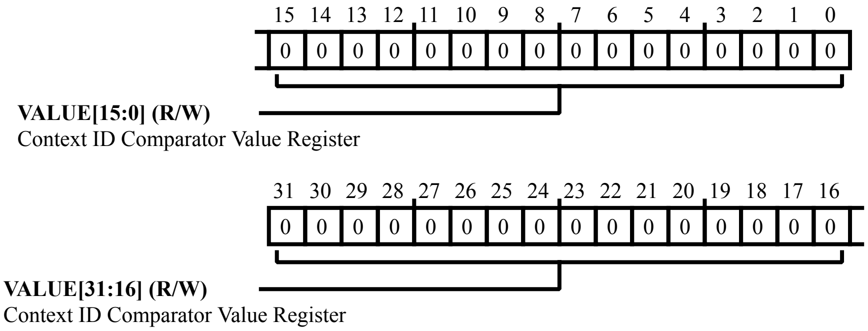

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R/W)
Context ID Comparator Value Register
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R/W)
Context ID Comparator Value Register

Figure 56-13: CSPFT_CIDCVR[n] Register Diagram

Table 56-24: CSPFT_CIDCVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Context ID Comparator Value Register. The `CSPFT_CIDCVR[n].VALUE` bit field holds a context ID value for comparison.

Claim Tag Clear Register

The `CSPFT_CLAIMCLR` register is used to clear bits in the claim tag or get the current value of the claim tag.

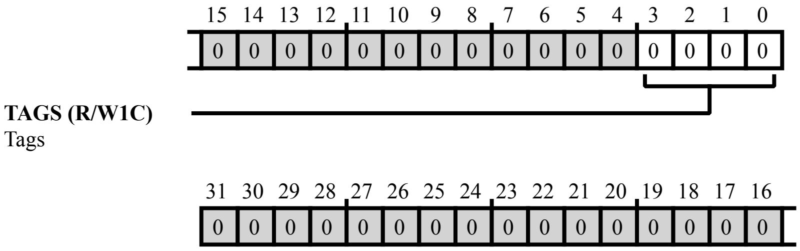

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TAGS (R/W1C)
Tags
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-14: CSPFT_CLAIMCLR Register Diagram

Table 56-25: CSPFT_CLAIMCLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1C)	TAGS	Tags. A read of the `CSPFT_CLAIMCLR.TAGS` bit field returns the current value, a write clears bits.

Claim Tag Set Register

The `CSPFT_CLAIMSET` register is used to set bits in the claim tag and find the number of bits supported by the claim tag.

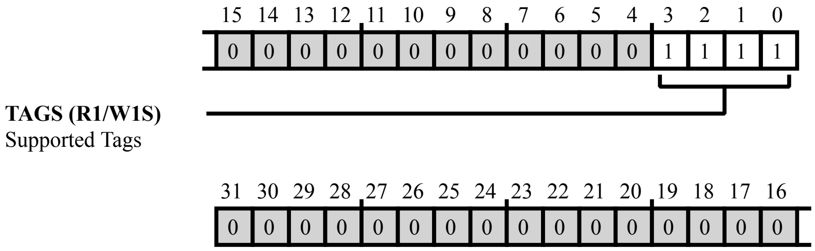

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
TAGS (R1/W1S)
Supported Tags
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-15: CSPFT_CLAIMSET Register Diagram

Table 56-26: CSPFT_CLAIMSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R1/W1S)	TAGS	Supported Tags. The `CSPFT_CLAIMSET.TAGS` bit field sets bits in the claim tag and finds the number of bits supported by the claim tag.

Counter Enable Event Register

The `CSPFT_CNTENR[n]` register describes the event that enables the corresponding counter.

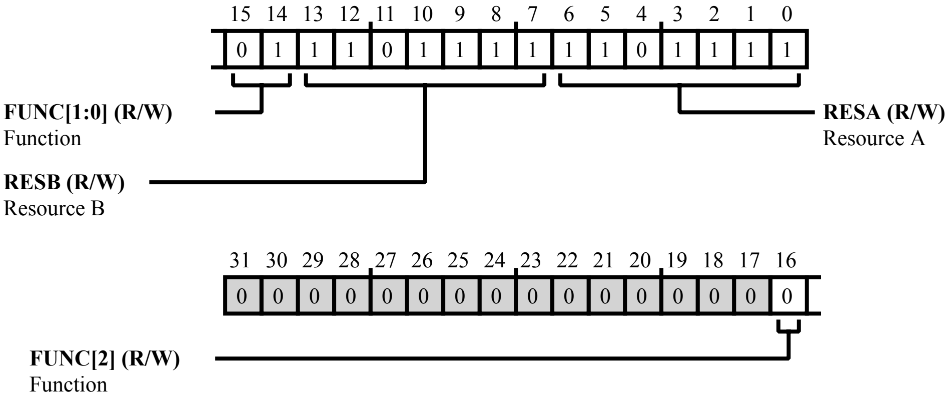

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
FUNC[1:0] (R/W)
Function
RESB (R/W)
Resource B
RESA (R/W)
Resource A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNC[2] (R/W)
Function

Figure 56-16: CSPFT_CNTENR[n] Register Diagram

Table 56-27: CSPFT_CNTENR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The `CSPFT_CNTENR[n].FUNC` bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The `CSPFT_CNTENR[n].RESB` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_CNTENR[n].FUNC` field (See `CSPFT_CNTENR[n].RESA` for list of possible values).	
6:0 (R/W)	RESA	Resource A. The `CSPFT_CNTENR[n].RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_CNTENR[n].FUNC` field.	

Table 56-27: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Single Addr Comparator 0
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1

Table 56-27: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		90	Context ID Comparator 2
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Counter Reload Event Register

The `CSPFT_CNTRLDEVR[n]` register defines the event that causes the corresponding counter to be reloaded with the value held in the corresponding `CSPFT_CNTRLDVR[n]` register.

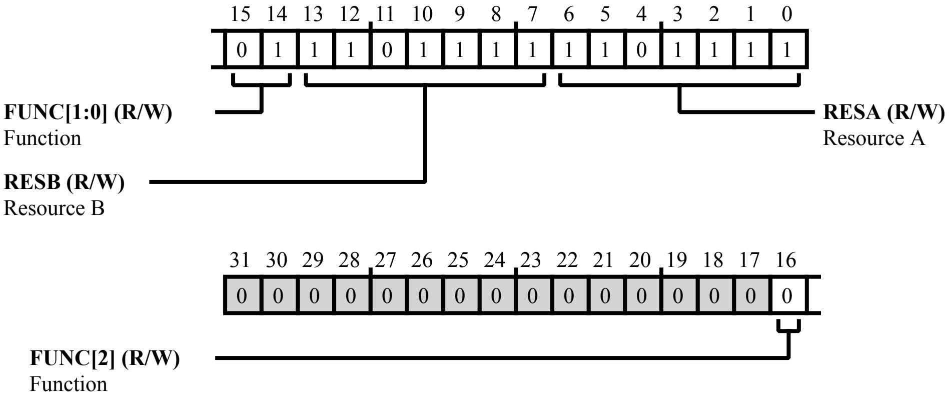

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
FUNC[1:0] (R/W)
Function
RESB (R/W)
Resource B
RESA (R/W)
Resource A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNC[2] (R/W)
Function

Figure 56-17: CSPFT_CNTRLDEVR[n] Register Diagram

Table 56-28: CSPFT_CNTRLDEVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The `CSPFT_CNTRLDEVR[n].FUNC` bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The `CSPFT_CNTRLDEVR[n].RESB` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_CNTRLDEVR[n].FUNC` field (See `CSPFT_CNTRLDEVR[n].RESA` for list of possible values).	

Table 56-28: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:0 (R/W)	RESA	Resource A. The `CSPFT_CNTRLDEVR[n].RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_CNTRLDEVR[n].FUNC` field.	
		0	Single Addr Comparator 0
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at zero
		65	Counter 1 at zero
		66	Counter 2 at zero

Table 56-28: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		67	Counter 3 at zero
		88	Context ID comparator 0
		89	Context ID comparator 1
		90	Context ID comparator 2
		95	TraceEnable start/stop resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace prohibited
		111	Always TRUE

Counter Reload Value Register

The `CSPFT_CNTRLDVR[n]` register specifies the starting value of the corresponding counter.

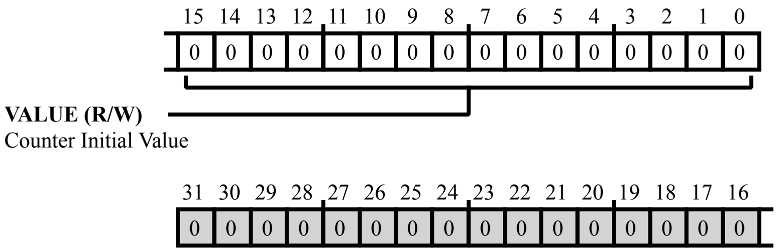

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Counter Initial Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-18: CSPFT_CNTRLDVR[n] Register Diagram

Table 56-29: CSPFT_CNTRLDVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Counter Initial Value. The `CSPFT_CNTRLDVR[n].VALUE` bit field specifies the starting value of the corresponding counter.

Counter Value Register

The `CSPFT_CNTVR[n]` register holds the current value of the corresponding counter.

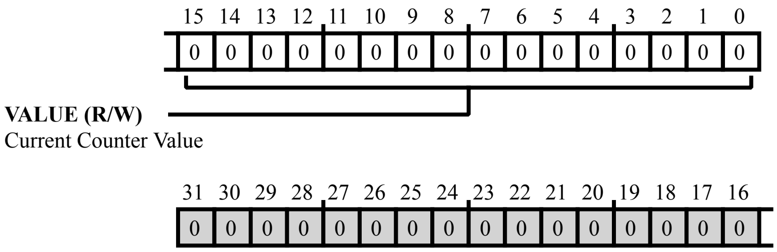

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE (R/W)
Current Counter Value
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-19: CSPFT_CNTVR[n] Register Diagram

Table 56-30: CSPFT_CNTVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Current Counter Value. The `CSPFT_CNTVR[n].VALUE` bit field specifies the current value of the corresponding counter.

Main Control Register

The `CSPFT_CTL` register controls general operation of the PTM, such as whether tracing is enabled.

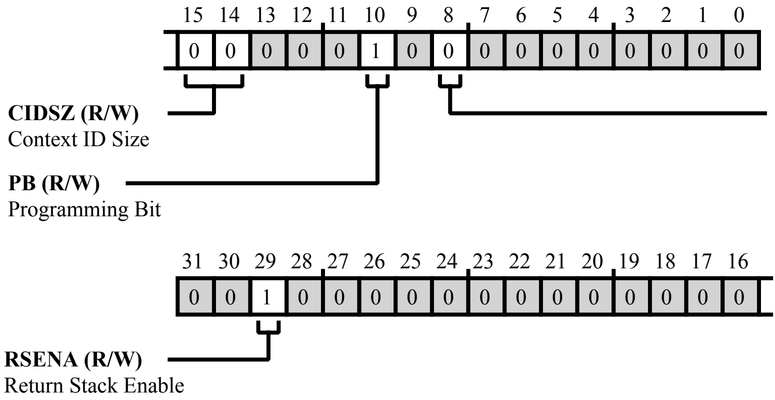

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
CIDSZ (R/W)
Context ID Size
PB (R/W)
Programming Bit
BBRAN (R/W)
Branch Broadcast
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
RSENA (R/W)
Return Stack Enable

Figure 56-20: CSPFT_CTL Register Diagram

Table 56-31: CSPFT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	RSENA	Return Stack Enable. When the `CSPFT_CTL.RSENA` bit is set, the first indirect branch back to an address generates a branch without exception packet, and subsequent branches back to the same address generate E Atoms. This compresses inner loops of HW loops and code that indirectly branches back.	
15:14 (R/W)	CIDSZ	Context ID Size. The `CSPFT_CTL.CIDSZ` bit field specifies the byte size to trace. Only the bytes specified are traced, even if the new Context ID value is larger than this.	
		0	No Context ID Tracing
		1	One byte Traced
		2	Two Bytes Traced
		3	Three Bytes Traced

Table 56-31: CSPFT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	PB	Programming Bit. To program the CSPFT, use the following procedure. 1. Set the `CSPFT_CTL.PB` bit to disable all trace functionality. 2. Poll the `CSPFT_STAT.PB` bit waiting for it to be 1 (FIFO drained, trace halted). 3. Program the trace registers, counter and other registers, as required. 4. Set this bit to 0. 5. Poll the `CSPFT_STAT.PB` bit until it reads 0 (trace status reset, trace restarted). When the `CSPFT_CTL.PB` bit is set, the FIFO is drained and no more trace is produced. All counters are held in their present state and the external outputs are forced low. After the FIFO is drained, the `CSPFT_STAT.PB` is set to reflect that the part is ready to program. When this bit is cleared, the trace status is cleared and trace is restarted. 0 Trace Enabled 1 Trace Disabled
8 (R/W)	BBRAN	Branch Broadcast. Set the `CSPFT_CTL.BBRAN` bit to 1 to enable branch broadcasting. Branch broadcasting traces the address of direct branch instructions rather than producing E atoms.

Device Type Identifier Register

The `CSPFT_DEVTYPE` register is read-only. It provides a debugger with information about the component when the part number field is not recognized. The debugger can then report this information.

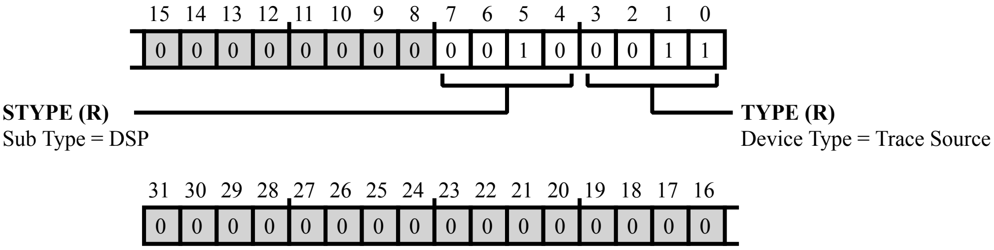

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1
STYPE (R)
Sub Type = DSP
TYPE (R)
Device Type = Trace Source
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-21: CSPFT_DEVTYPE Register Diagram

Table 56-32: CSPFT_DEVTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	STYPE	Sub Type = DSP.
3:0 (R/NW)	TYPE	Device Type = Trace Source.

External Output Event Register

The `CSPFT_EXTOUTEVR[n]` register defines the event that controls the corresponding EXTOUT external output signal.

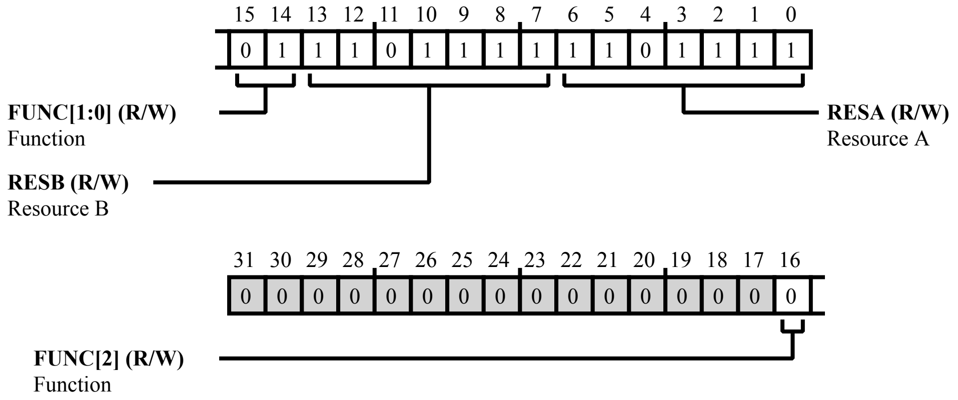

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
FUNC[1:0] (R/W)
Function
RESB (R/W)
Resource B
RESA (R/W)
Resource A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNC[2] (R/W)
Function

Figure 56-22: CSPFT_EXTOUTEVR[n] Register Diagram

Table 56-33: CSPFT_EXTOUTEVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The `CSPFT_EXTOUTEVR[n].FUNC` bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The `CSPFT_EXTOUTEVR[n].RESB` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_EXTOUTEVR[n].FUNC` field (See `CSPFT_EXTOUTEVR[n].RESA` for list of possible values).	

Table 56-33: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:0 (R/W)	RESA	Resource A. The `CSPFT_EXTOUTEVR[n].RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_EXTOUTEVR[n].FUNC` field.	
		0	Single Addr Comparator 0
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero

Table 56-33: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Hardware Feature Register

The `CSPFT_HWFEAT` register enables software to read the implementation defined configuration of the PTM, giving the number of each type of hardware resource. Each field indicates the number of instances of a particular resource, zero indicates that there are no implemented resources of that type.

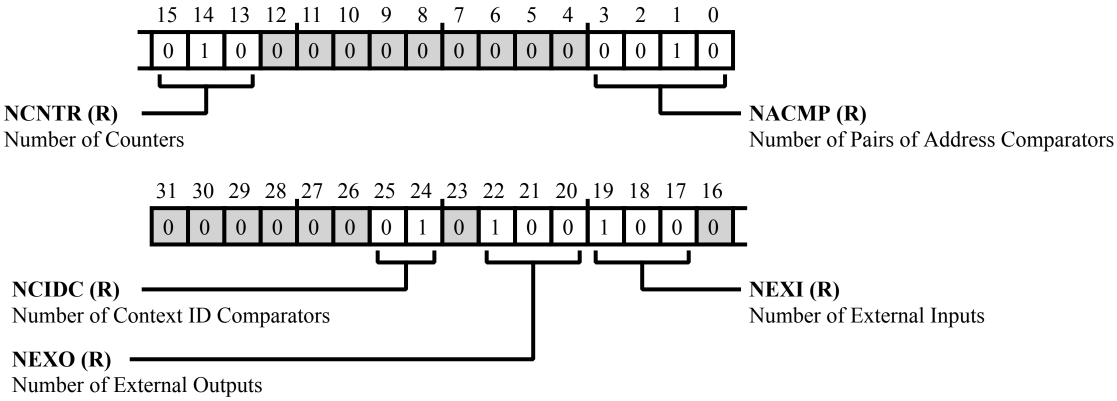

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
NCNTR (R)
Number of Counters
NACMP (R)
Number of Pairs of Address Comparators
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0
NCIDC (R)
Number of Context ID Comparators
NEXO (R)
Number of External Outputs
NEXI (R)
Number of External Inputs

Figure 56-23: CSPFT_HWFEAT Register Diagram

Table 56-34: CSPFT_HWFEAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/NW)	NCIDC	Number of Context ID Comparators. The `CSPFT_HWFEAT.NCIDC` bit field identifies the number of context ID comparators.
22:20 (R/NW)	NEXO	Number of External Outputs. The `CSPFT_HWFEAT.NEXO` bit field identifies the number of external outputs (up to four).
19:17 (R/NW)	NEXI	Number of External Inputs. The `CSPFT_HWFEAT.NEXI` bit field identifies the number of external inputs (up to four).
15:13 (R/NW)	NCNTR	Number of Counters. The `CSPFT_HWFEAT.NCNTR` bit field identifies the number of counters (up to four) that are configured using the counter registers.
3:0 (R/NW)	NACMP	Number of Pairs of Address Comparators. The `CSPFT_HWFEAT.NACMP` bit field identifies the number of pairs of address comparators as address range comparators (ARCs). In this case, two adjacent address comparators form the ARC, so you can use address comparators 1 and 2 to define the first ARC. An ARC matches when any instruction in the specified range is committed for execution, regardless of whether the instruction passes its condition code test.

Table 56-34: CSPFT_HWFEAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	No Address Comparators
		1	1 Pair of Address Comparators
		2	2 Pairs of Address Comparators
		3	3 Pairs of Address Comparators
		4	4 Pairs of Address Comparators
		5	5 Pairs of Address Comparators
		6	6 Pairs of Address Comparators
		7	7 Pairs of Address Comparators
		8	8 Pairs of Address Comparators

Lock Access Register

The `CSPFT_LAR` register is used to provide lock and unlock access to all other CSPFT registers.

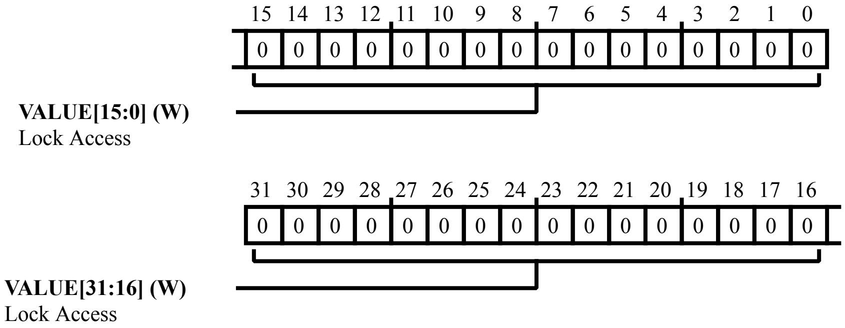

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (W)
Lock Access
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (W)
Lock Access

Figure 56-24: CSPFT_LAR Register Diagram

Table 56-35: CSPFT_LAR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Lock Access. Write 0xC5ACCE55 to the `CSPFT_LAR.VALUE` bit field to unlock. Write any other value to lock.

Lock Status Register

The `CSPFT_LSR` register is used to detect if the lock registers are implemented and if they are currently locked.

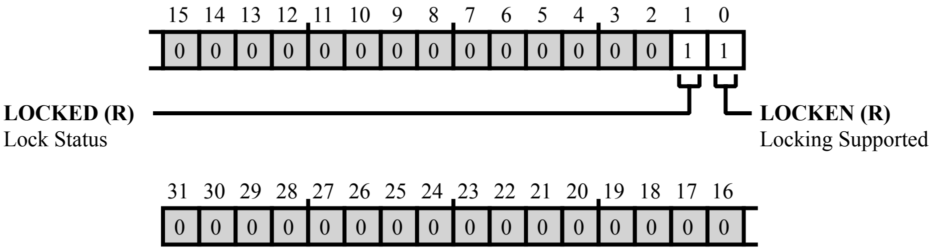

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
LOCKED (R)
Lock Status
LOCKEN (R)
Locking Supported
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-25: CSPFT_LSR Register Diagram

Table 56-36: CSPFT_LSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	LOCKED	Lock Status. The `CSPFT_LSR.LOCKED` bit indicates whether the PFT is locked.	
		0	Writes are permitted
		1	Locked. Writes are ignored
0 (R/NW)	LOCKEN	Locking Supported. The `CSPFT_LSR.LOCKEN` bit indicates whether the lock registers are implemented for this interface.	
		0	Locking is Not Required. This access is from an interface that ignores the lock registers.
		1	Locking is Required. This access is from an interface that requires the PFT to be unlocked.

Peripheral ID0 Register

The `CSPFT_PID0` register holds peripheral identification information.

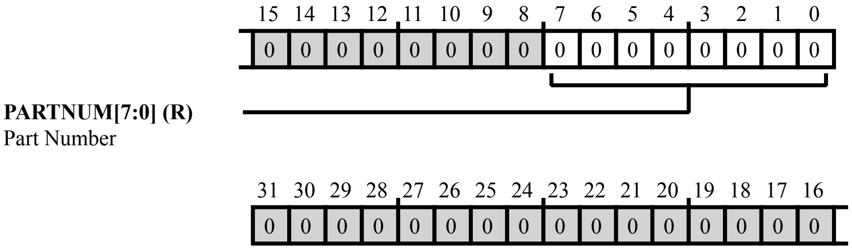

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PARTNUM[7:0] (R)
Part Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-26: CSPFT_PID0 Register Diagram

Table 56-37: CSPFT_PID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PARTNUM	Part Number. The `CSPFT_PID0.PARTNUM` bit field holds the peripheral identification number.

Peripheral ID1 Register

The `CSPFT_PID1` register holds peripheral identification information.

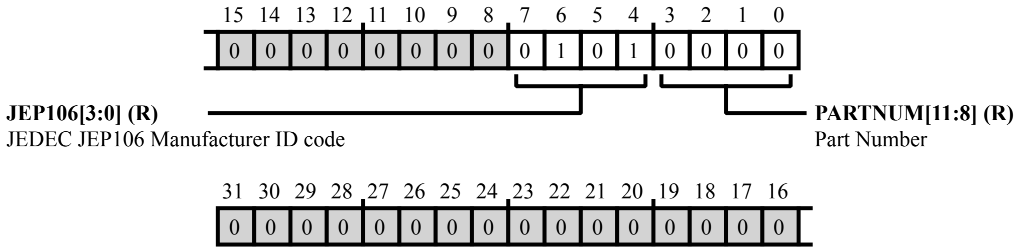

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
JEP106[3:0] (R)
JEDEC JEP106 Manufacturer ID code
PARTNUM[11:8] (R)
Part Number
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-27: CSPFT_PID1 Register Diagram

Table 56-38: CSPFT_PID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.
3:0 (R/NW)	PARTNUM	Part Number. The `CSPFT_PID1.PARTNUM` bit field holds the peripheral identification number.

Peripheral ID2 Register

The `CSPFT_PID2` register holds peripheral identification information.

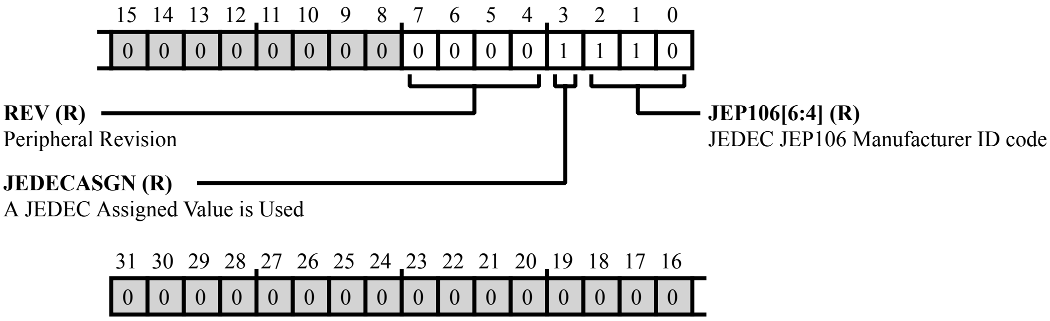

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
REV (R)
Peripheral Revision
JEDECASGN (R)
A JEDEC Assigned Value is Used
JEP106[6:4] (R)
JEDEC JEP106 Manufacturer ID code
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-28: CSPFT_PID2 Register Diagram

Table 56-39: CSPFT_PID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV	Peripheral Revision.
3 (R/NW)	JEDECASGN	A JEDEC Assigned Value is Used. The `CSPFT_PID2.JEDECASGN` bit indicates that a JEDEC assigned value is used.
2:0 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.

Peripheral ID3 Register

The CSPFT_PID3 register holds peripheral identification information.

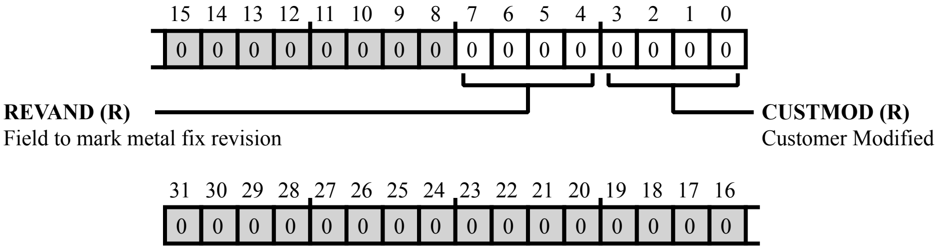

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
REVAND (R)
Field to mark metal fix revision
CUSTMOD (R)
Customer Modified
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-29: CSPFT_PID3 Register Diagram

Table 56-40: CSPFT_PID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REVAND	Field to mark metal fix revision.
3:0 (R/NW)	CUSTMOD	Customer Modified.

Peripheral ID4 Register

The CSPFT_PID4 register holds peripheral identification information.

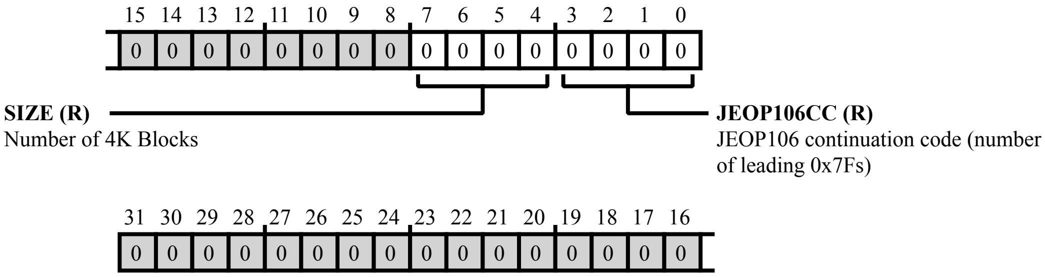

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SIZE (R)
Number of 4K Blocks
JEOP106CC (R)
JEOP106 continuation code (number of leading 0x7Fs)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-30: CSPFT_PID4 Register Diagram

Table 56-41: CSPFT_PID4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	SIZE	Number of 4K Blocks. The `CSPFT_PID4.SIZE` bit field contains the size of the component in 4K chunks minus 1 (for example 0=4K).
3:0 (R/NW)	JEOP106CC	JEOP106 continuation code (number of leading 0x7Fs).

Status Register

The CSPFT_STAT register provides information about the current status of the trace and trigger logic.

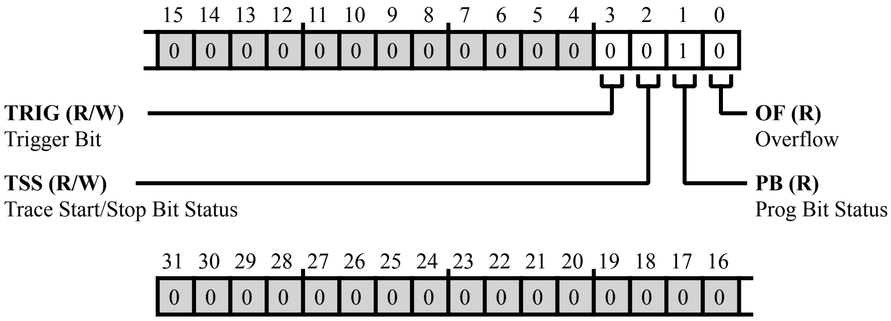

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
TRIG (R/W)
Trigger Bit
TSS (R/W)
Trace Start/Stop Bit Status
OF (R)
Overflow
PB (R)
Prog Bit Status
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-31: CSPFT_STAT Register Diagram

Table 56-42: CSPFT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TRIG	Trigger Bit. The CSPFT_STAT.TRIG bit is set when the trigger occurs, and prevents the trigger from being output until the CSPFT is next programmed. This bit is reset when the CSPFT_CTL.PB bit transitions from 1 to 0.
2 (R/W)	TSS	Trace Start/Stop Bit Status. The CSPFT_STAT.TSS bit holds the current status of the trace start/stop resource. If = 1, it indicates that a trace start address has been matched, without a corresponding trace stop address match. This bit =0 when trace is restarted (the CSPFT_CTL.PB bit transitions from 1 to 0).
1 (R/NW)	PB	Prog Bit Status. The CSPFT_STAT.PB bit indicates the current effective value of the CSPFT_CTL.PB bit. The program must wait for this bit to =1 before programming the CSPFT. (See the CSPFT_CTL.PB bit description).
0 (R/NW)	OF	Overflow. If the CSPFT_STAT.OF bit is =1, there is an overflow. This bit is cleared =0 when the trace is restarted (CSPFT_CTL.PB transitions from 1 to 0)

Synchronization Frequency Register

The `CSPFT_SYNCFR` register holds the trace synchronization frequency value.

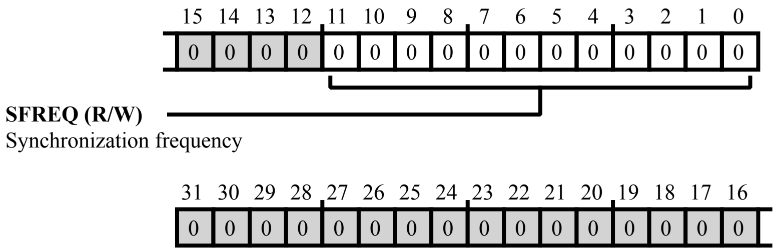

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SFREQ (R/W)
Synchronization frequency
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-32: CSPFT_SYNCFR Register Diagram

Table 56-43: CSPFT_SYNCFR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	SFREQ	Synchronization frequency. The `CSPFT_SYNCFR.SFREQ` bit field is the number of 128-byte blocks of trace data after which you want to drop an address synchronization packet. If the circular buffer size is 16k, ensure that there are a few A-syncs in the buffer, so setting this to 16 means that every 2k there is an A-Sync packet.

TraceEnable Control Register

The `CSPFT_TECTL` register controls the start stop logic, whether resources specified are used for include or exclude, and specifies the address range comparators to use.

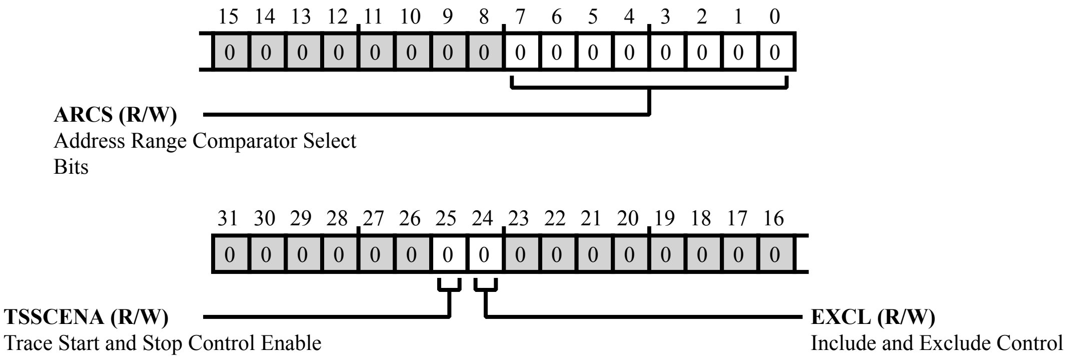

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ARCS (R/W)
Address Range Comparator Select Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TSSCENA (R/W)
Trace Start and Stop Control Enable
EXCL (R/W)
Include and Exclude Control

Figure 56-33: CSPFT_TECTL Register Diagram

Table 56-44: CSPFT_TECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	TSSCENA	Trace Start and Stop Control Enable.	
		0	Tracing is not affected by the trace start/stop logic
		1	Tracing is controlled by the trace on and off address comparators
24 (R/W)	EXCL	Include and Exclude Control.	
		0	Include. The specified address range comparators indicate the regions where tracing can occur. When outside the region trace is prevented.
		1	Exclude. The specified address range comparators indicate regions to be excluded from the trace. When outside the range tracing is enabled.
7:0 (R/W)	ARCS	Address Range Comparator Select Bits. When a bit in the `CSPFT_TECTL.ARCS` bit field is set to 1, it selects an address range comparator, for include/exclude control.	

TraceEnable Event Register

The `CSPFT_TEEVENT` register defines the TraceEnable enabling event.

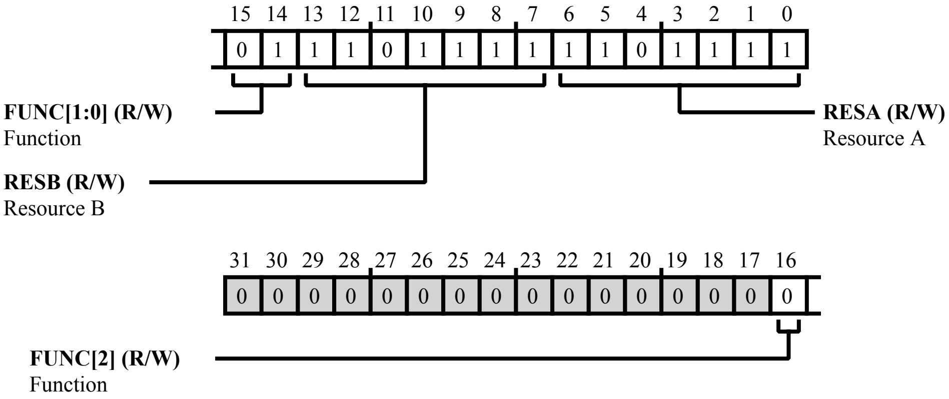

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
FUNC[1:0] (R/W)
Function
RESB (R/W)
Resource B
RESA (R/W)
Resource A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNC[2] (R/W)
Function

Figure 56-34: CSPFT_TEEVENT Register Diagram

Table 56-45: CSPFT_TEEVENT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The `CSPFT_TEEVENT.FUNC` bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The `CSPFT_TEEVENT.RESB` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_TEEVENT.FUNC` field. (See `CSPFT_TEEVENT.RESA` for list of possible values).	
6:0 (R/W)	RESA	Resource A. The `CSPFT_TEEVENT.RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_TEEVENT.FUNC` field.	
		0	Single Addr Comparator 0

Table 56-45: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2

Table 56-45: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

CoreSight Trace ID Register

The `CSPFT_TRACEIDR` register defines the 7-bit trace ID, for output to the trace bus.

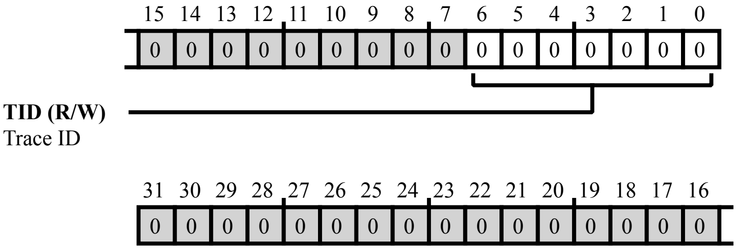

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TID (R/W)
Trace ID
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-35: CSPFT_TRACEIDR Register Diagram

Table 56-46: CSPFT_TRACEIDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	TID	Trace ID. Defines the 7-bit trace ID, for output to the trace bus. Used in systems where multiple trace sources are present and tracing simultaneously. For example, when outputs trace onto the AMBA 3 Advanced Trace Bus, a unique ID is required for each trace source.

Trigger Event Register

The `CSPFT_TRIGGER` register defines the event that controls the trigger. This event creates the trigger output signal that is in the ATCLK domain.

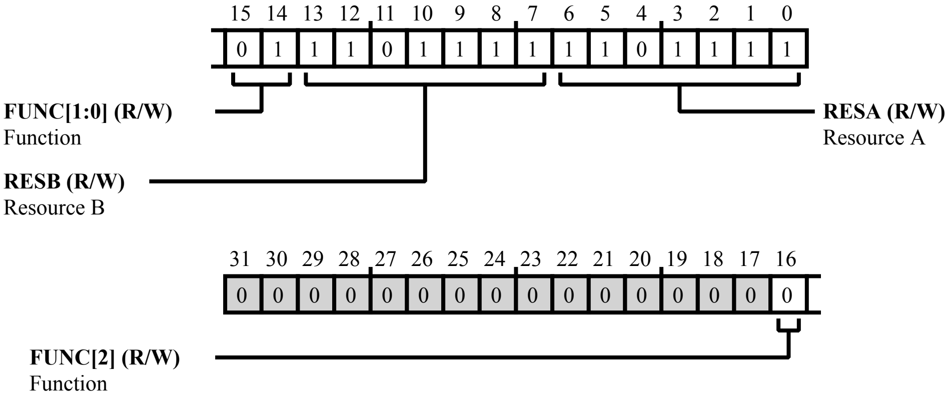

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
FUNC[1:0] (R/W)
Function
RESB (R/W)
Resource B
RESA (R/W)
Resource A
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FUNC[2] (R/W)
Function

Figure 56-36: CSPFT_TRIGGER Register Diagram

Table 56-47: CSPFT_TRIGGER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. Specifies the logical operation that combines the two resources that define the event	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. Specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_TRIGGER.FUNC` field. (See `CSPFT_TRIGGER.RESA` for list of possible values.)	
6:0 (R/W)	RESA	Resource A. Specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_TRIGGER.FUNC` field	
		0	Single Addr Comparator 0

Table 56-47: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2

Table 56-47: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

TraceEnable Start/Stop Control Register

The `CSPFT_TSSCTL` register specifies the single address comparators that hold the trace start and stop addresses.

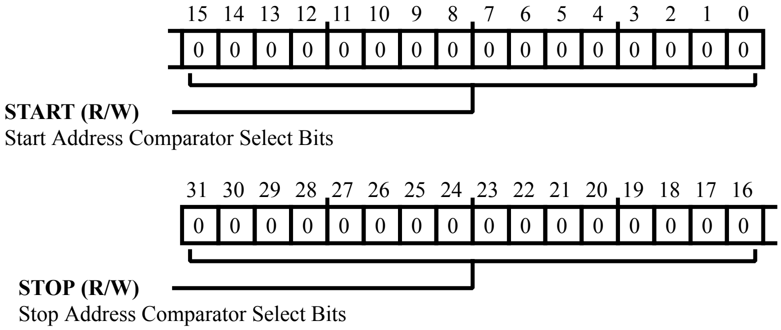

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
START (R/W)
Start Address Comparator Select Bits
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
STOP (R/W)
Stop Address Comparator Select Bits

Figure 56-37: CSPFT_TSSCTL Register Diagram

Table 56-48: CSPFT_TSSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16 (R/W)	STOP	Stop Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a stop address for the TraceEnable start/stop block.	
		0	disabled
		1	Address Comparator 1
		2	Address Comparator 2
		4	Address Comparator 3
		8	Address Comparator 4
		16	Address Comparator 5
		32	Address Comparator 6
		64	Address Comparator 7
		128	Address Comparator 8
		256	Address Comparator 9
		512	Address Comparator 10
		1024	Address Comparator 11
		2048	Address Comparator 12
		4096	Address Comparator 13
		8192	Address Comparator 14
		16384	Address Comparator 15

Table 56-48: CSPFT_TSSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		32768	Address Comparator 16
15:0 (R/W)	START	Start Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a start address for the TraceEnable start/stop block.	
		0	Disabled
		1	Address Comparator 1
		2	Address Comparator 2
		4	Address Comparator 3
		8	Address Comparator 4
		16	Address Comparator 5
		32	Address Comparator 6
		64	Address Comparator 7
		128	Address Comparator 8
		256	Address Comparator 9
		512	Address Comparator 10
		1024	Address Comparator 11
		2048	Address Comparator 12
		4096	Address Comparator 13
		8192	Address Comparator 14
		16384	Address Comparator 15
		32768	Address Comparator 16

ADSP-SC58x TAPC Register Descriptions

TAPC (TAPC) contains the following registers.

Table 56-49: ADSP-SC58x TAPC Register List

Name	Description
`TAPC_DBGCTL`	Debug Control Register
`TAPC_IDCODE`	IDCODE Register
`TAPC_SDBGKEY0`	Secure Debug Key 0 Register
`TAPC_SDBGKEY1`	Secure Debug Key 1 Register

Table 56-49: ADSP-SC58x TAPC Register List (Continued)

Name	Description
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

Debug Control Register

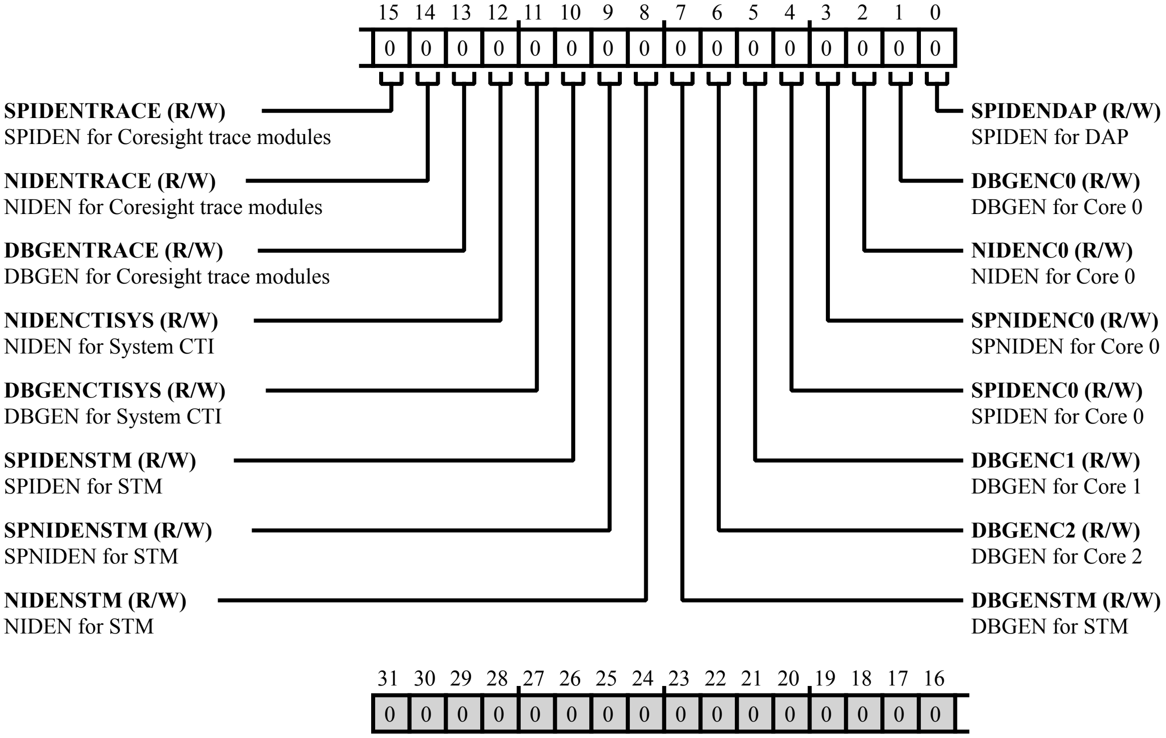

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SPIDENTRACE (R/W)
SPIDEN for Coresight trace modules
NIDENTRACE (R/W)
NIDEN for Coresight trace modules
DBGENTRACE (R/W)
DBGEN for Coresight trace modules
NIDENCTISYS (R/W)
NIDEN for System CTI
DBGENCTISYS (R/W)
DBGEN for System CTI
SPIDENSTM (R/W)
SPIDEN for STM
SPNIDENSTM (R/W)
SPNIDEN for STM
NIDENSTM (R/W)
NIDEN for STM
SPIDENDAP (R/W)
SPIDEN for DAP
DBGENC0 (R/W)
DBGEN for Core 0
NIDENC0 (R/W)
NIDEN for Core 0
SPNIDENC0 (R/W)
SPNIDEN for Core 0
SPIDENC0 (R/W)
SPIDEN for Core 0
DBGENC1 (R/W)
DBGEN for Core 1
DBGENC2 (R/W)
DBGEN for Core 2
DBGENSTM (R/W)
DBGEN for STM
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-38: TAPC_DBGCTL Register Diagram

Table 56-50: TAPC_DBGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	SPIDENTRACE	SPIDEN for Coresight trace modules.
14 (R/W)	NIDENTRACE	NIDEN for Coresight trace modules.
13 (R/W)	DBGENTRACE	DBGEN for Coresight trace modules.
12 (R/W)	NIDENCTISYS	NIDEN for System CTI.
11 (R/W)	DBGENCTISYS	DBGEN for System CTI.
10 (R/W)	SPIDENSTM	SPIDEN for STM.

Table 56-50: TAPC_DBGCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	SPNIDENSTM	SPNIDEN for STM.
8 (R/W)	NIDENSTM	NIDEN for STM.
7 (R/W)	DBGENSTM	DBGEN for STM.
6 (R/W)	DBGENC2	DBGEN for Core 2.
5 (R/W)	DBGENC1	DBGEN for Core 1.
4 (R/W)	SPIDENC0	SPIDEN for Core 0.
3 (R/W)	SPNIDENC0	SPNIDEN for Core 0.
2 (R/W)	NIDENC0	NIDEN for Core 0.
1 (R/W)	DBGENC0	DBGEN for Core 0.
0 (R/W)	SPIDENDAP	SPIDEN for DAP.

IDCODE Register

The `TAPC_IDCODE` register holds the IDCODE. The bit field is defined as follows.

IDCODE[31:28] = 0x1* – REVID

IDCODE[27:12] = 0x280B – JTAG ID

IDCODE[11:1] = 0x65 – Manufacturer ID

IDCODE[0] = 0x1

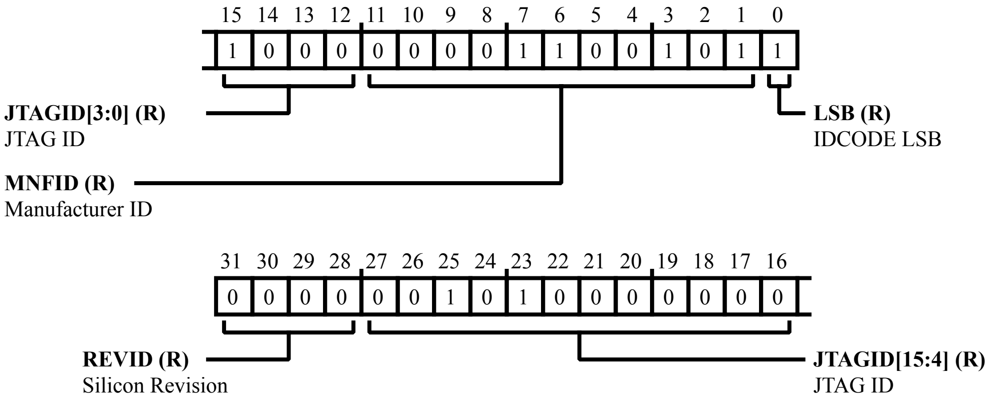

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1
JTAGID[3:0] (R)
JTAG ID
MNFID (R)
Manufacturer ID
LSB (R)
IDCODE LSB
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
REVID (R)
Silicon Revision
JTAGID[15:4] (R)
JTAG ID

Figure 56-39: TAPC_IDCODE Register Diagram

Table 56-51: TAPC_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Silicon Revision. The `TAPC_IDCODE.REVID` bit field holds the silicon revision. See the processor anomaly list for details.
27:12 (R/NW)	JTAGID	JTAG ID.
11:1 (R/NW)	MNFID	Manufacturer ID.
0 (R/NW)	LSB	IDCODE LSB.

Secure Debug Key 0 Register

The `TAPC_SDBGKEY0` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. A debug key of 128 bits needs to be written into the Secure Debug Key registers (`TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, `TAPC_SDBGKEY3`) in the TAPC through the peripheral bus interface.

These registers hold the value of the key against which a matching key provided by the debug user is compared to enable a debug session. The task of writing these registers is performed (initially) by boot ROM code which copies a customer-selected key from the Flash memory info block to these registers.

An SDBGKEY value of all 0's is always an invalid key, a value of all 1's match the default value of the Secure Debug Key Compare registers and requires no entry in these registers. It is recommended programs have a significant number of 0's and 1's in a pseudo-random pattern throughout the 128-bit code for maximum protection.

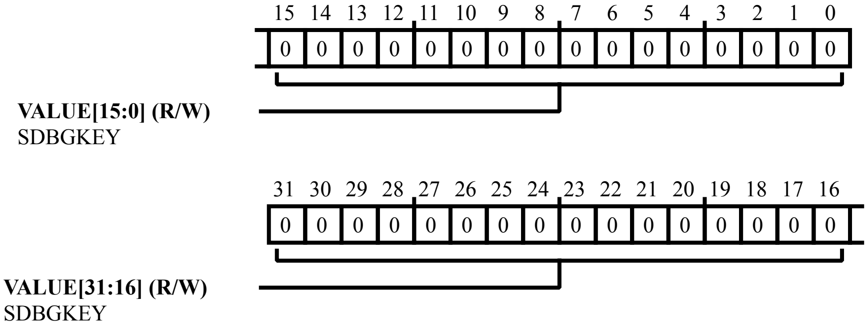


Figure 56-40: TAPC_SDBGKEY0 Register Diagram

Table 56-52: TAPC_SDBGKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The `TAPC_SDBGKEY0.VALUE` bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 1 Register

The TAPC_SDBGKEY1 register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the TAPC_SDBGKEY0 register description for more information.

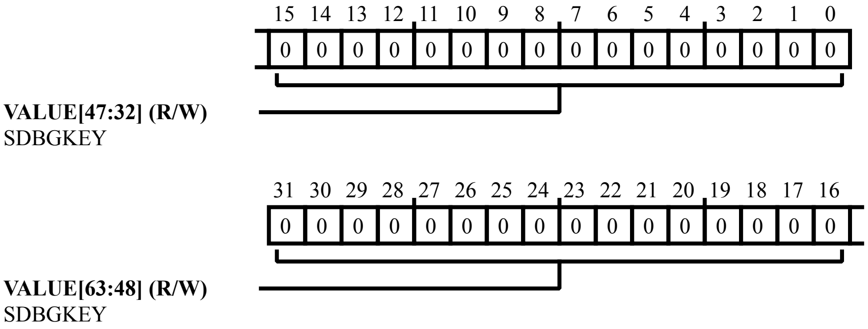

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[47:32] (R/W)
SDBGKEY
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[63:48] (R/W)
SDBGKEY

Figure 56-41: TAPC_SDBGKEY1 Register Diagram

Table 56-53: TAPC_SDBGKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The TAPC_SDBGKEY1.VALUE bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 2 Register

The `TAPC_SDBGKEY2` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the `TAPC_SDBGKEY0` register description for more information.

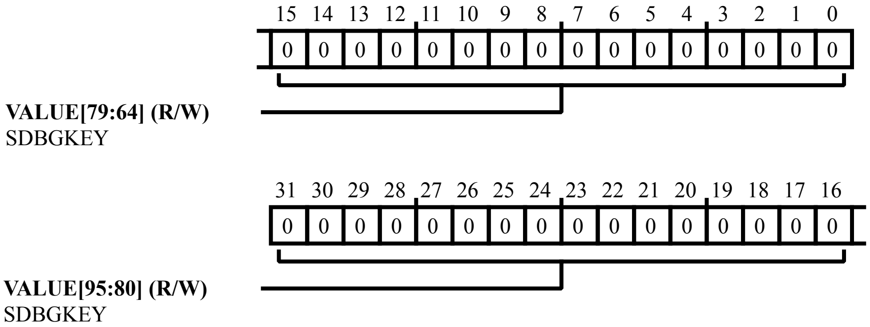

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[79:64] (R/W)
SDBGKEY
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[95:80] (R/W)
SDBGKEY

Figure 56-42: TAPC_SDBGKEY2 Register Diagram

Table 56-54: TAPC_SDBGKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The `TAPC_SDBGKEY2.VALUE` bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 3 Register

The `TAPC_SDBGKEY3` register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the `TAPC_SDBGKEY0` register description for more information.

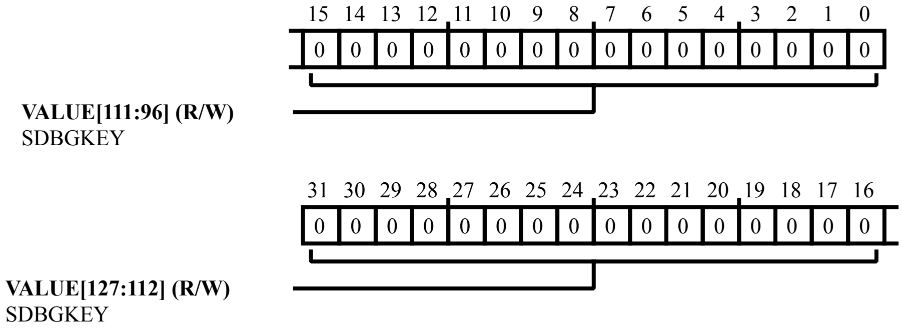

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[111:96] (R/W)
SDBGKEY
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[127:112] (R/W)
SDBGKEY

Figure 56-43: TAPC_SDBGKEY3 Register Diagram

Table 56-55: TAPC_SDBGKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The `TAPC_SDBGKEY3.VALUE` bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key Control Register

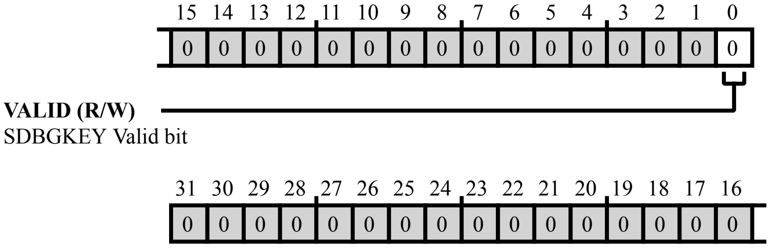

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALID (R/W)
SDBGKEY Valid bit
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-44: TAPC_SDBGKEY_CTL Register Diagram

Table 56-56: TAPC_SDBGKEY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALID	SDBGKEY Valid bit.

Secure Debug Key Status Register

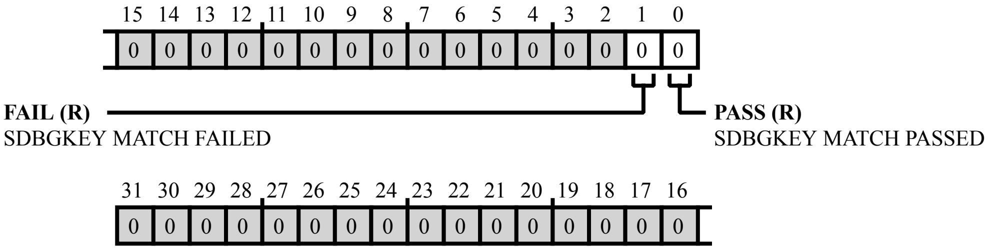

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FAIL (R)
SDBGKEY MATCH FAILED
PASS (R)
SDBGKEY MATCH PASSED
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 56-45: TAPC_SDBGKEY_STAT Register Diagram

Table 56-57: TAPC_SDBGKEY_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	FAIL	SDBGKEY MATCH FAILED.
0 (R/NW)	PASS	SDBGKEY MATCH PASSED.

USERCODE Register

The `TAPC_USERCODE` register

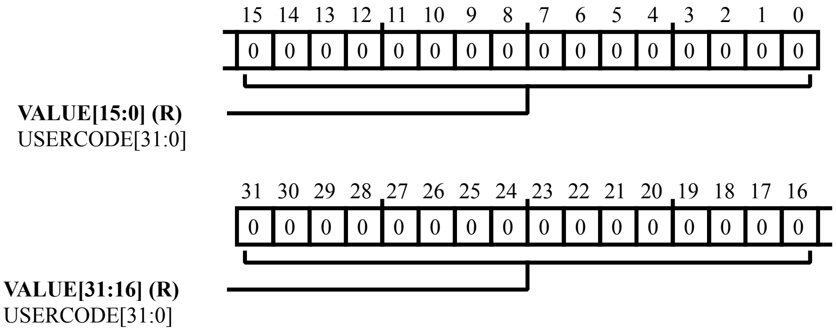

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[15:0] (R)
USERCODE[31:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
VALUE[31:16] (R)
USERCODE[31:0]

Figure 56-46: TAPC_USERCODE Register Diagram

Table 56-58: TAPC_USERCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	USERCODE[31:0].

Appendix A ADSP-SC58x Register List

This appendix lists Memory-Mapped Register address and register names. The modules are presented in alphabetical order.

Table A-1: ADSP-SC58x ACM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31020000	ACM0_CTL	ACM0 Control Register	0x00000000
0x31020004	ACM0_TC0	ACM0 Timing Configuration 0 Register	0x00000001
0x31020008	ACM0_TC1	ACM0 Timing Configuration 1 Register	0x0000000F
0x3102000C	ACM0_STAT	ACM0 Status Register	0x00000000
0x31020010	ACM0_EVSTAT	ACM0 Event Complete Status Register	0x00000000
0x31020014	ACM0_EVMSK	ACM0 Event Complete Interrupt Mask Register	0x00000000
0x31020018	ACM0_MEVSTAT	ACM0 Missed Event Status Register	0x00000000
0x3102001C	ACM0_MEVMSK	ACM0 Missed Event Interrupt Mask Register	0x00000000
0x31020020	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020024	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020028	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x3102002C	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020030	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020034	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020038	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x3102003C	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020040	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020044	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020048	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x3102004C	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020050	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020054	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020058	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x3102005C	ACM0_EVCTL[n]	ACM0 Event N Control Register	0x00000000
0x31020060	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020064	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020068	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000

Table A-1: ADSP-SC58x ACM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102006C	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020070	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020074	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020078	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x3102007C	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020080	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020084	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020088	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x3102008C	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020090	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020094	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x31020098	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x3102009C	ACM0_EVTIME[n]	ACM0 Event N Time Register	0x00000000
0x310200A0	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200A4	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200A8	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200AC	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200B0	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200B4	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200B8	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200BC	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200C0	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200C4	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200C8	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200CC	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200D0	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200D4	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200D8	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200DC	ACM0_EVORD[n]	ACM0 Event N Order Register	0x00000000
0x310200E8	ACM0_TMR0	ACM0 Timer 0 Register	0x00000000
0x310200EC	ACM0_TMR1	ACM0 Timer 1 Register	0x00000000

Table A-2: ADSP-SC58x ARMDBG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31120000	ARMDBG0_DIDR	ARMDBG0 Debug ID Register.	0x1203F001
0x31120018	ARMDBG0_WFAR	ARMDBG0 The Watchpoint Fault Address Register.	0x00000000
0x3112001C	ARMDBG0_VCR	ARMDBG0 Vector Catch Register.	0x00000000
0x31120024	ARMDBG0_ECR	ARMDBG0 Event Catch Register.	0x00000000
0x31120028	ARMDBG0_DSCCR	ARMDBG0 Debug State Cache Control Register.	0x00000000
0x3112002C	ARMDBG0_DSMCR	ARMDBG0 Debug State MMU Control Register.	0x00000000
0x31120080	ARMDBG0_DTRRX	ARMDBG0 Read Data Transfer Register.	0x00000000
0x31120084	ARMDBG0_ITR	ARMDBG0 Instruction Transfer Register.	0xFFFFFFFF
0x31120088	ARMDBG0_DSCR	ARMDBG0 Debug Status and Control Register.	0x02030002
0x3112008C	ARMDBG0_DTRTX	ARMDBG0 Write Data Transfer Register.	0x00000000
0x31120090	ARMDBG0_DRCR	ARMDBG0 Debug Run Control Register.	0x00000000
0x311200A0	ARMDBG0_PCSR	ARMDBG0 Program Counter Sampling Register.	0xFFFFFFFF
0x311200A4	ARMDBG0_CIDSR	ARMDBG0 Context ID Sampling Register.	0x00000000
0x31120100	ARMDBG0_BVR0	ARMDBG0 Breakpoint Value Register 0.	0x00000000
0x31120104	ARMDBG0_BVR1	ARMDBG0 Breakpoint Value Register 1.	0x00000000
0x31120108	ARMDBG0_BVR2	ARMDBG0 Breakpoint Value Register 2.	0x00000000
0x31120140	ARMDBG0_BCR0	ARMDBG0 Breakpoint Control Register 0.	0x00000000
0x31120144	ARMDBG0_BCR1	ARMDBG0 Breakpoint Control Register 1.	0x00000000
0x31120148	ARMDBG0_BCR2	ARMDBG0 Breakpoint Control Register 2.	0x00000000
0x31120180	ARMDBG0_WVR0	ARMDBG0 Watchpoint Value Register 0.	0x00000000
0x31120184	ARMDBG0_WVR1	ARMDBG0 Watchpoint Value Register 1.	0x00000000
0x311201C0	ARMDBG0_WCR0	ARMDBG0 Watchpoint Control Register 0.	0x00000000
0x311201C4	ARMDBG0_WCR1	ARMDBG0 Watchpoint Control Register 1.	0x00000000
0x31120300	ARMDBG0_OSLAR	ARMDBG0 Operating System Lock Access Register.	0x00000000
0x31120304	ARMDBG0_OSLSR	ARMDBG0 Operating System Lock Status Register.	0x00000000
0x31120310	ARMDBG0_PRCR	ARMDBG0 Device Powerdown and Reset Control Register.	0x00000000
0x31120314	ARMDBG0_PRSR	ARMDBG0 Device Powerdown and Reset Status Register.	0x00000009
0x31120D00	ARMDBG0_MIDR	ARMDBG0 Main ID Register.	0x410FC051
0x31120D04	ARMDBG0_CTR	ARMDBG0 Cache Type Register.	0x83338003
0x31120D08	ARMDBG0_TCMTR	ARMDBG0 TCM Type Register.	0x00000000

Table A-2: ADSP-SC58x ARMDBG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31120D0C	ARMDBG0_TLBTR	ARMDBG0 TCM Type Register.	0x00000000
0x31120D10	ARMDBG0_MPUIR	ARMDBG0 MPU Type Register (alias of MIDR in VMSA).	0x410FC051
0x31120D14	ARMDBG0_MPIDR	ARMDBG0 Multiprocessor Affinity Register.	0xC0000000
0x31120D18	ARMDBG0_MIDRALIAS1	ARMDBG0 Main ID Register (alias at 0xd18).	0x410FC051
0x31120D1C	ARMDBG0_MIDRALIAS2	ARMDBG0 Main ID Register (alias at 0xd1c).	0x410FC051
0x31120D20	ARMDBG0_ID_PFR0	ARMDBG0 Processor Feature Register 0.	0x00001231
0x31120D24	ARMDBG0_ID_PFR1	ARMDBG0 Processor Feature Register 1.	0x00000011
0x31120D28	ARMDBG0_ID_DFR0	ARMDBG0 Debug Feature Register 0.	0x02010444
0x31120D2C	ARMDBG0_ID_AFR0	ARMDBG0 Auxiliary Feature Register 0.	0x00000000
0x31120D30	ARMDBG0_ID_MMFR0	ARMDBG0 Memory Model Feature Register 0.	0x00100003
0x31120D34	ARMDBG0_ID_MMFR1	ARMDBG0 Memory Model Feature Register 1.	0x40000000
0x31120D38	ARMDBG0_ID_MMFR2	ARMDBG0 Memory Model Feature Register 2.	0x01230000
0x31120D3C	ARMDBG0_ID_MMFR3	ARMDBG0 Memory Model Feature Register 3.	0x00102211
0x31120D40	ARMDBG0_ID_ISAR0	ARMDBG0 ISA Feature Register 0.	0x00101111
0x31120D44	ARMDBG0_ID_ISAR1	ARMDBG0 ISA Feature Register 1.	0x13112111
0x31120D48	ARMDBG0_ID_ISAR2	ARMDBG0 ISA Feature Register 2.	0x21232041
0x31120D4C	ARMDBG0_ID_ISAR3	ARMDBG0 ISA Feature Register 3.	0x11112131
0x31120D50	ARMDBG0_ID_ISAR4	ARMDBG0 ISA Feature Register 4.	0x00011142
0x31120D54	ARMDBG0_ID_ISAR5	ARMDBG0 ISA Feature Register 5.	0x00000000
0x31120EF8	ARMDBG0_ITMISCOUT	ARMDBG0 Miscellaneous Outputs Integration Register.	0x00000000
0x31120EFC	ARMDBG0_ITMISCIN	ARMDBG0 Miscellaneous Inputs Integration Register.	0x00000006
0x31120F00	ARMDBG0_ITCTRL	ARMDBG0 Integration Mode Control Register.	0x00000000
0x31120FA0	ARMDBG0_CLAIMSET	ARMDBG0 Claim Tag Set Register.	0x000000FF
0x31120FA4	ARMDBG0_CLAIMCLR	ARMDBG0 Claim Tag Clear Register.	0x00000000
0x31120FB0	ARMDBG0_LOCKACCESS	ARMDBG0 Lock Access Register.	0x00000000
0x31120FB4	ARMDBG0_LOCKSTATUS	ARMDBG0 Lock Status Register.	0x00000003
0x31120FB8	ARMDBG0_AUTHSTATUS	ARMDBG0 Authentication Status Register.	0x000000AA
0x31120FC8	ARMDBG0_DEVID	ARMDBG0 Device Indentifier.	0x00000F13
0x31120FCC	ARMDBG0_DEVTYPE	ARMDBG0 Device Type Register.	0x00000015
0x31120FD0	ARMDBG0_PIR4	ARMDBG0 Peripheral ID Register 4.	0x00000004

Table A-2: ADSP-SC58x ARMDBG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31120FE0	ARMDBG0_PIR0	ARMDBG0 Peripheral ID Register 0.	0x00000005
0x31120FE4	ARMDBG0_PIR1	ARMDBG0 Peripheral ID Register 1.	0x000000BC
0x31120FE8	ARMDBG0_PIR2	ARMDBG0 Peripheral ID Register 2.	0x0000001B
0x31120FEC	ARMDBG0_PIR3	ARMDBG0 Peripheral ID Register 3.	0x00000000
0x31120FF0	ARMDBG0_CIR0	ARMDBG0 Component ID Register 0.	0x0000000D
0x31120FF4	ARMDBG0_CIR1	ARMDBG0 Component ID Register 1.	0x00000090
0x31120FF8	ARMDBG0_CIR2	ARMDBG0 Component ID Register 2.	0x00000005
0x31120FFC	ARMDBG0_CIR3	ARMDBG0 Component ID Register 3.	0x000000B1

Table A-3: ADSP-SC58x ARMETM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3112C000	ARMETM0_CR	ARMETM0 Main Control Register	0x00000441
0x3112C004	ARMETM0_CCR	ARMETM0 Configuration Code Register	0x8D294024
0x3112C008	ARMETM0_TRIGGER	ARMETM0 Trigger Event Register	0x00000000
0x3112C00C	ARMETM0_ASICCTLR	ARMETM0 ASIC Control Register	0x00000000
0x3112C010	ARMETM0_SR	ARMETM0 Status Register	0x00000002
0x3112C014	ARMETM0_SCR	ARMETM0 System Configuration Register	0x00020C0C
0x3112C018	ARMETM0_TSSCR	ARMETM0 TraceEnable Start/Stop Control Register	0x00000000
0x3112C01C	ARMETM0_TECR2	ARMETM0 TraceEnable Control Register 2	0x00000000
0x3112C020	ARMETM0_TEEVR	ARMETM0 TraceEnable Event Register	0x00000000
0x3112C024	ARMETM0_TECR1	ARMETM0 TraceEnable Control Register 1	0x00000000
0x3112C02C	ARMETM0_FFLR	ARMETM0 FIFOFULL Level Register	0x00000000
0x3112C030	ARMETM0_VDEVR	ARMETM0 ViewData Event Register	0x00000000
0x3112C034	ARMETM0_VDCR1	ARMETM0 ViewData Control Register 1	0x00000000
0x3112C03C	ARMETM0_VDCR3	ARMETM0 ViewData Control Register 3	0x00000000
0x3112C040	ARMETM0_ACVR1	ARMETM0 Address Comparator Value Register 1	0x00000000
0x3112C044	ARMETM0_ACVR2	ARMETM0 Address Comparator Value Register 2	0x00000000
0x3112C048	ARMETM0_ACVR3	ARMETM0 Address Comparator Value Register 3	0x00000000
0x3112C04C	ARMETM0_ACVR4	ARMETM0 Address Comparator Value Register 4	0x00000000
0x3112C050	ARMETM0_ACVR5	ARMETM0 Address Comparator Value Register 5	0x00000000

Table A-3: ADSP-SC58x ARMETM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3112C054	ARMETM0_ACVR6	ARMETM0 Address Comparator Value Register 6	0x00000000
0x3112C058	ARMETM0_ACVR7	ARMETM0 Address Comparator Value Register 7	0x00000000
0x3112C05C	ARMETM0_ACVR8	ARMETM0 Address Comparator Value Register 8	0x00000000
0x3112C080	ARMETM0_ACTR1	ARMETM0 Address Comparator Access Type Register 1	0x00000000
0x3112C084	ARMETM0_ACTR2	ARMETM0 Address Comparator Access Type Register 2	0x00000000
0x3112C088	ARMETM0_ACTR3	ARMETM0 Address Comparator Access Type Register 3	0x00000000
0x3112C08C	ARMETM0_ACTR4	ARMETM0 Address Comparator Access Type Registers 4	0x00000000
0x3112C090	ARMETM0_ACTR5	ARMETM0 Address Comparator Access Type Register 5	0x00000000
0x3112C094	ARMETM0_ACTR6	ARMETM0 Address Comparator Access Type Register 6	0x00000000
0x3112C098	ARMETM0_ACTR7	ARMETM0 Address Comparator Access Type Register 7	0x00000000
0x3112C09C	ARMETM0_ACTR8	ARMETM0 Address Comparator Access Type Register 8	0x00000000
0x3112C0C0	ARMETM0_DCVR1	ARMETM0 Data Comparator Value Register 1	0x00000000
0x3112C0C8	ARMETM0_DCVR3	ARMETM0 Data Comparator Value Register 3	0x00000000
0x3112C100	ARMETM0_DCMR1	ARMETM0 Data Comparator Mask Register 1	0x00000000
0x3112C108	ARMETM0_DCMR3	ARMETM0 Data Comparator Mask Register 3	0x00000000
0x3112C140	ARMETM0_CNTRLDVR1	ARMETM0 Counter Reload Value Register 1	0x00000000
0x3112C144	ARMETM0_CNTRLDVR2	ARMETM0 Counter Reload Value Register 2	0x00000000
0x3112C150	ARMETM0_CNTENR1	ARMETM0 Counter Enable Register 1	0x00000000
0x3112C154	ARMETM0_CNTENR2	ARMETM0 Counter Enable Register 2	0x00000000
0x3112C160	ARMETM0_CNTRLDEVR1	ARMETM0 Counter Reload Event Register 1	0x00000000
0x3112C164	ARMETM0_CNTRLDEVR2	ARMETM0 Counter Reload Event Register 2	0x00000000
0x3112C170	ARMETM0_CNTVR1	ARMETM0 Counter Value Register 1	0x00000000
0x3112C174	ARMETM0_CNTVR2	ARMETM0 Counter Value Register 2	0x00000000
0x3112C180	ARMETM0_SQ12EVR	ARMETM0 Sequencer State Transition 12 Event Register	0x00000000
0x3112C184	ARMETM0_SQ21EVR	ARMETM0 Sequencer State Transition 21 Event Register	0x00000000
0x3112C188	ARMETM0_SQ23EVR	ARMETM0 Sequencer State Transition 23 Event Register	0x00000000
0x3112C18C	ARMETM0_SQ31EVR	ARMETM0 Sequencer State Transition 31 Event Register	0x00000000
0x3112C190	ARMETM0_SQ32EVR	ARMETM0 Sequencer State Transition 32 Event Register	0x00000000
0x3112C194	ARMETM0_SQ13EVR	ARMETM0 Sequencer State Transition 13 Event Register	0x00000000
0x3112C19C	ARMETM0_SQR	ARMETM0 Current Sequencer State Register	0x00000000
0x3112C1A0	ARMETM0_EXTOUTEVR1	ARMETM0 External Output Event Register 1	0x00000000

Table A-3: ADSP-SC58x ARMETM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3112C1A4	ARMETM0_EXTOUTEVR2	ARMETM0 External Output Event Register 2	0x00000000
0x3112C1B0	ARMETM0_CIDCVR	ARMETM0 Context ID Comparator Value Register	0x00000000
0x3112C1BC	ARMETM0_CIDCMR	ARMETM0 Context ID Comparator Mask Register	0x00000000
0x3112C1E0	ARMETM0_SYNCFR	ARMETM0 Synchronization Frequency Register	0x00000400
0x3112C1E4	ARMETM0_IDR	ARMETM0 ETM ID Register	0x410CF252
0x3112C1E8	ARMETM0_CCER	ARMETM0 Configuration Code Extension Register	0x304008F2
0x3112C1EC	ARMETM0_EXTINSELR	ARMETM0 Extended External Input Selection Register	0x00000000
0x3112C1F8	ARMETM0_TSEVR	ARMETM0 Timestamp Event Register	0x00000000
0x3112C1FC	ARMETM0_AUXCR	ARMETM0 Auxiliary Control Register	0x00000000
0x3112C200	ARMETM0_TRACEIDR	ARMETM0 CoreSight Trace ID Register	0x00000000
0x3112C208	ARMETM0_ETMAUXIDR	ARMETM0 Auxiliary ID Register	0x00000003
0x3112C314	ARMETM0_PDSR	ARMETM0 Power-Down Status Register	0x00000001
0x3112CEDC	ARMETM0_ITMISCOUT	ARMETM0 Miscellaneous Outputs Register	0x00000000
0x3112CEE0	ARMETM0_ITMISCIN	ARMETM0 Miscellaneous Inputs Register	0x00000000
0x3112CEE8	ARMETM0_ITTRIGGERREQ	ARMETM0 Trigger Request Register	0x00000000
0x3112CEEC	ARMETM0_ITATBDATA0	ARMETM0 ATB Data Register 0	0x00000000
0x3112CEF0	ARMETM0_ITATBCTR2	ARMETM0 ATB Control Register 2	0x00000000
0x3112CEF4	ARMETM0_ITATBCTR1	ARMETM0 ATB Control Register 1	0x00000000
0x3112CEF8	ARMETM0_ITATBCTR0	ARMETM0 ATB Control Register 0	0x00000000
0x3112CF00	ARMETM0_ITCTRL	ARMETM0 Integration Mode Control Register	0x00000000
0x3112CFA0	ARMETM0_CLAIMSET	ARMETM0 Claim Tag Set Register	0x000000FF
0x3112CFA4	ARMETM0_CLAIMCLR	ARMETM0 Claim Tag Clear Register	0x00000000
0x3112CFB0	ARMETM0_LAR	ARMETM0 Lock Access Register	0x00000000
0x3112CFB4	ARMETM0_LSR	ARMETM0 Lock Status Register	0x00000003
0x3112CFB8	ARMETM0_AUTHSTATUS	ARMETM0 Authentication Status Register	0x00000008
0x3112CFC8	ARMETM0_DEVID	ARMETM0 Device Indentifier	0x00000000
0x3112CFCC	ARMETM0_DEVTYPE	ARMETM0 Device Type Register	0x00000013
0x3112CFD0	ARMETM0_PIR4	ARMETM0 Peripheral ID Register 4	0x00000004
0x3112CFD4	ARMETM0_PIR5	ARMETM0 Peripheral ID Register 5	0x00000000
0x3112CFD8	ARMETM0_PIR6	ARMETM0 Peripheral ID Register 6	0x00000000

Table A-3: ADSP-SC58x ARMETM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3112CFDC	ARMETM0_PIR7	ARMETM0 Peripheral ID Register 7	0x00000000
0x3112CFE0	ARMETM0_PIR0	ARMETM0 Peripheral ID Register 0	0x00000055
0x3112CFE4	ARMETM0_PIR1	ARMETM0 Peripheral ID Register 1	0x000000B9
0x3112CFE8	ARMETM0_PIR2	ARMETM0 Peripheral ID Register 2	0x0000002B
0x3112CFEC	ARMETM0_PIR3	ARMETM0 Peripheral ID Register 3	0x00000000
0x3112CFF0	ARMETM0_CIR0	ARMETM0 Component ID Register 0	0x0000000D
0x3112CFF4	ARMETM0_CIR1	ARMETM0 Component ID Register 1	0x00000090
0x3112CFF8	ARMETM0_CIR2	ARMETM0 Component ID Register 2	0x00000005
0x3112CFFC	ARMETM0_CIR3	ARMETM0 Component ID Register 3	0x000000B1

Table A-4: ADSP-SC58x ARMPMU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31121000	ARMPMU0_PMXEVCNTR0	ARMPMU0 PM0 Counter Register.	0x00000000
0x31121004	ARMPMU0_PMXEVCNTR1	ARMPMU0 PM1 Counter Register.	0x00000000
0x3112107C	ARMPMU0_PMCCNTR	ARMPMU0 Cycle Count Register.	0x00000000
0x31121400	ARMPMU0_PMXEVTYPER0	ARMPMU0 PM0 Event Type Register.	0x00000000
0x31121404	ARMPMU0_PMXEVTYPER1	ARMPMU0 PM0 Event Type Register.	0x00000000
0x3112147C	ARMPMU0_PMCCFILTR	ARMPMU0 Cycle Count Filter Control Register.	0x00000000
0x31121C00	ARMPMU0_PMCNTENSET	ARMPMU0 Count Enable Set Register.	0x00000000
0x31121C20	ARMPMU0_PMCNTENCLR	ARMPMU0 Count Enable Clear Register.	0x00000000
0x31121C40	ARMPMU0_PMINTENSET	ARMPMU0 Interrupt Enable Set Register.	0x00000000
0x31121C60	ARMPMU0_PMINTENCLR	ARMPMU0 Interrupt Enable Clear Register.	0x00000000
0x31121C80	ARMPMU0_PMOVSR	ARMPMU0 Overflow Flag Status Register.	0x00000000
0x31121CA0	ARMPMU0_PMSWINC	ARMPMU0 Software Increment Register.	0x00000000
0x31121E00	ARMPMU0_PMCFGR	ARMPMU0 Configuration Register.	0x0009DF02
0x31121E04	ARMPMU0_PMCR	ARMPMU0 Control Register.	0x41051000

Table A-4: ADSP-SC58x ARMPMU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31121E08	ARMPMU0_PMUSERENR	ARMPMU0 User Enable Register.	0x00000000
0x31121E20	ARMPMU0_PMCEID0	ARMPMU0 Common Event Identification Register 0.	0x003FFFFF
0x31121E24	ARMPMU0_PMCEID1	ARMPMU0 Common Event Identification Register 1.	0x00000000
0x31121FB0	ARMPMU0_PMLAR	ARMPMU0 Lock Access Register.	0x00000000
0x31121FB4	ARMPMU0_PMLSR	ARMPMU0 Lock Status Register.	0x00000003
0x31121FB8	ARMPMU0_PMAUTHSTATUS	ARMPMU0 Authentication Status Register.	0x00000088
0x31121FCC	ARMPMU0_PMDEVTYPE	ARMPMU0 Device Type Register.	0x00000016
0x31121FD0	ARMPMU0_PIR4	ARMPMU0 Peripheral ID Register 4.	0x00000004
0x31121FE0	ARMPMU0_PIR0	ARMPMU0 Peripheral ID Register 0.	0x000000A5
0x31121FE4	ARMPMU0_PIR1	ARMPMU0 Peripheral ID Register 1.	0x000000B9
0x31121FE8	ARMPMU0_PIR2	ARMPMU0 Peripheral ID Register 2.	0x0000001B
0x31121FEC	ARMPMU0_PIR3	ARMPMU0 Peripheral ID Register 3.	0x00000000
0x31121FF0	ARMPMU0_CIR0	ARMPMU0 Component ID Register 0.	0x0000000D
0x31121FF4	ARMPMU0_CIR1	ARMPMU0 Component ID Register 1.	0x00000090
0x31121FF8	ARMPMU0_CIR2	ARMPMU0 Component ID Register 2.	0x00000005
0x31121FFC	ARMPMU0_CIR3	ARMPMU0 Component ID Register 3.	0x000000B1

Table A-5: ADSP-SC58x ARMROM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31110000	ARMROM0_ROMENTRY00	ARMROM0 ROM Entry 00	0x00010003
0x31110004	ARMROM0_ROMENTRY01	ARMROM0 ROM Entry 01	0x00011003
0x31110008	ARMROM0_ROMENTRY02	ARMROM0 ROM Entry 02	0x00012002
0x3111000C	ARMROM0_ROMENTRY03	ARMROM0 ROM Entry 03	0x00013002
0x31110010	ARMROM0_ROMENTRY04	ARMROM0 ROM Entry 04	0x00014002
0x31110014	ARMROM0_ROMENTRY05	ARMROM0 ROM Entry 05	0x00015002

Table A-5: ADSP-SC58x ARMROM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31110018	ARMROM0_ROMENTRY06	ARMROM0 ROM Entry 06	0x00016002
0x3111001C	ARMROM0_ROMENTRY07	ARMROM0 ROM Entry 07	0x00017002
0x31110020	ARMROM0_ROMENTRY08	ARMROM0 ROM Entry 08	0x00018003
0x31110024	ARMROM0_ROMENTRY09	ARMROM0 ROM Entry 09	0x00019002
0x31110028	ARMROM0_ROMENTRY10	ARMROM0 ROM Entry 10	0x0001A002
0x3111002C	ARMROM0_ROMENTRY11	ARMROM0 ROM Entry 11	0x0001B002
0x31110030	ARMROM0_ROMENTRY12	ARMROM0 ROM Entry 12	0x0001C003
0x31110034	ARMROM0_ROMENTRY13	ARMROM0 ROM Entry 13	0x0001D002
0x31110038	ARMROM0_ROMENTRY14	ARMROM0 ROM Entry 14	0x0001E002
0x3111003C	ARMROM0_ROMENTRY15	ARMROM0 ROM Entry 15	0x0001F002

Table A-6: ADSP-SC58x ASRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9240	ASRC0_CTL01	ASRC0 Control Register for ASRC 0 and 1	0x00000000
0x310C9244	ASRC0_CTL23	ASRC0 Control Register for ASRC 2 and 3	0x00000000
0x310C9248	ASRC0_MUTE	ASRC0 Mute Register	0x00000000
0x310C9260	ASRC0_RAT01	ASRC0 Ratio Register for ASRC 0 and 1	0x00000000
0x310C9264	ASRC0_RAT23	ASRC0 Ratio Register for ASRC 2 and 3	0x00000000

Table A-7: ADSP-SC58x ASRC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB240	ASRC1_CTL01	ASRC1 Control Register for ASRC 0 and 1	0x00000000
0x310CB244	ASRC1_CTL23	ASRC1 Control Register for ASRC 2 and 3	0x00000000

Table A-7: ADSP-SC58x ASRC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB248	ASRC1_MUTE	ASRC1 Mute Register	0x00000000
0x310CB260	ASRC1_RAT01	ASRC1 Ratio Register for ASRC 0 and 1	0x00000000
0x310CB264	ASRC1_RAT23	ASRC1 Ratio Register for ASRC 2 and 3	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31000200	CAN0_MC1	CAN0 Mailbox Configuration 1 Register	0x00000000
0x31000204	CAN0_MD1	CAN0 Mailbox Direction 1 Register	0x000000FF
0x31000208	CAN0_TRS1	CAN0 Transmission Request Set 1 Register	0x00000000
0x3100020C	CAN0_TRR1	CAN0 Transmission Request Reset 1 Register	0x00000000
0x31000210	CAN0_TA1	CAN0 Transmission Acknowledge 1 Register	0x00000000
0x31000214	CAN0_AA1	CAN0 Abort Acknowledge 1 Register	0x00000000
0x31000218	CAN0_RMP1	CAN0 Receive Message Pending 1 Register	0x00000000
0x3100021C	CAN0_RML1	CAN0 Receive Message Lost 1 Register	0x00000000
0x31000220	CAN0_MBTIF1	CAN0 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x31000224	CAN0_MBRIF1	CAN0 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x31000228	CAN0_MBIM1	CAN0 Mailbox Interrupt Mask 1 Register	0x00000000
0x3100022C	CAN0_RFH1	CAN0 Remote Frame Handling 1 Register	0x00000000
0x31000230	CAN0_OPSS1	CAN0 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x31000240	CAN0_MC2	CAN0 Mailbox Configuration 2 Register	0x00000000
0x31000244	CAN0_MD2	CAN0 Mailbox Direction 2 Register	0x00000000
0x31000248	CAN0_TRS2	CAN0 Transmission Request Set 2 Register	0x00000000
0x3100024C	CAN0_TRR2	CAN0 Transmission Request Reset 2 Register	0x00000000
0x31000250	CAN0_TA2	CAN0 Transmission Acknowledge 2 Register	0x00000000
0x31000254	CAN0_AA2	CAN0 Abort Acknowledge 2 Register	0x00000000
0x31000258	CAN0_RMP2	CAN0 Receive Message Pending 2 Register	0x00000000
0x3100025C	CAN0_RML2	CAN0 Receive Message Lost 2 Register	0x00000000
0x31000260	CAN0_MBTIF2	CAN0 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x31000264	CAN0_MBRIF2	CAN0 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x31000268	CAN0_MBIM2	CAN0 Mailbox Interrupt Mask 2 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100026C	CAN0_RFH2	CAN0 Remote Frame Handling 2 Register	0x00000000
0x31000270	CAN0_OPSS2	CAN0 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x31000280	CAN0_CLK	CAN0 Clock Register	0x00000000
0x31000284	CAN0_TIMING	CAN0 Timing Register	0x00000000
0x31000288	CAN0_DBG	CAN0 Debug Register	0x00000008
0x3100028C	CAN0_STAT	CAN0 Status Register	0x00000080
0x31000290	CAN0_CEC	CAN0 Error Counter Register	0x00000000
0x31000294	CAN0_GIS	CAN0 Global CAN Interrupt Status Register	0x00000000
0x31000298	CAN0_GIM	CAN0 Global CAN Interrupt Mask Register	0x00000000
0x3100029C	CAN0_GIF	CAN0 Global CAN Interrupt Flag Register	0x00000000
0x310002A0	CAN0_CTL	CAN0 CAN Master Control Register	0x00000080
0x310002A4	CAN0_INT	CAN0 Interrupt Pending Register	0x00000000
0x310002AC	CAN0_MBTD	CAN0 Temporary Mailbox Disable Register	0x00000000
0x310002B0	CAN0_EWR	CAN0 Error Counter Warning Level Register	0x00006060
0x310002B4	CAN0_ESR	CAN0 Error Status Register	0x00000020
0x310002C4	CAN0_UCCNT	CAN0 Universal Counter Register	0x00000000
0x310002C8	CAN0_UCRC	CAN0 Universal Counter Reload/Capture Register	0x00000000
0x310002CC	CAN0_UCCNF	CAN0 Universal Counter Configuration Mode Register	0x00000000
0x31000300	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000304	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000308	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100030C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000310	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000314	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000318	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100031C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000320	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000324	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000328	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100032C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000330	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000334	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000338	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100033C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000340	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000344	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000348	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100034C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000350	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000354	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000358	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100035C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000360	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000364	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000368	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100036C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000370	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000374	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000378	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100037C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000380	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000384	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000388	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100038C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000390	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x31000394	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000398	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x3100039C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003A0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003A4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003A8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310003AC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003B0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003B4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003B8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003BC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003C0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003C4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003C8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003CC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003D0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003D4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003D8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003DC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003E0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003E4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003E8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003EC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003F0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003F4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x310003F8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x310003FC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x31000400	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000404	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000408	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100040C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000410	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000414	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000418	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100041C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000420	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000424	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000428	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100042C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000430	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000434	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000438	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100043C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000440	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000444	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000448	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100044C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000450	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000454	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000458	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100045C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000460	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000464	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000468	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100046C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000470	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000474	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000478	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100047C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000480	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000484	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000488	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100048C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000490	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000494	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000498	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100049C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310004A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310004A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310004A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310004AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310004B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310004B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310004B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310004BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310004C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310004C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310004C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310004CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310004D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310004D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310004D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310004DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310004E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310004E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310004E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310004EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310004F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310004F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310004F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310004FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000500	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000504	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000508	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100050C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000510	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000514	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000518	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100051C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000520	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000524	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000528	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100052C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000530	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000534	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000538	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100053C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000540	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000544	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000548	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100054C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000550	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000554	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000558	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100055C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000560	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000564	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000568	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100056C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000570	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000574	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000578	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100057C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000580	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000584	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000588	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100058C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000590	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000594	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000598	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100059C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310005A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310005A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310005A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310005AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310005B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310005B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310005B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310005BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310005C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310005C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310005C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310005CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310005D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310005D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310005D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310005DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310005E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310005E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310005E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310005EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310005F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310005F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310005F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310005FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000600	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000604	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000608	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100060C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000610	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000614	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000618	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100061C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000620	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000624	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000628	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100062C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000630	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000634	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000638	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100063C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000640	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000644	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000648	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100064C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000650	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000654	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000658	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100065C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000660	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000664	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000668	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100066C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000670	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000674	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000678	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100067C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000680	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000684	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000688	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100068C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000690	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000694	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000698	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100069C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310006A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310006A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310006A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310006AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310006B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310006B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310006B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310006BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310006C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310006C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310006C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310006CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310006D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310006D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310006D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310006DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310006E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310006E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310006E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310006EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310006F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310006F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310006F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310006FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000700	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000704	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000708	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100070C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000710	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000714	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000718	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100071C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000720	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000724	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000728	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100072C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000730	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000734	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000738	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100073C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000740	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000744	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000748	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100074C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000750	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000754	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000758	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100075C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000760	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000764	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000768	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100076C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000770	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000774	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000778	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100077C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x31000780	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x31000784	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x31000788	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x3100078C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x31000790	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x31000794	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x31000798	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x3100079C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310007A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-8: ADSP-SC58x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310007A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310007A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310007AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310007B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310007B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310007B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310007BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310007C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310007C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310007C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310007CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310007D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310007D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310007D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310007DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x310007E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x310007E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x310007E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x310007EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x310007F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x310007F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x310007F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x310007FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31000A00	CAN1_MC1	CAN1 Mailbox Configuration 1 Register	0x00000000
0x31000A04	CAN1_MD1	CAN1 Mailbox Direction 1 Register	0x000000FF

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000A08	CAN1_TRS1	CAN1 Transmission Request Set 1 Register	0x00000000
0x31000A0C	CAN1_TRR1	CAN1 Transmission Request Reset 1 Register	0x00000000
0x31000A10	CAN1_TA1	CAN1 Transmission Acknowledge 1 Register	0x00000000
0x31000A14	CAN1_AA1	CAN1 Abort Acknowledge 1 Register	0x00000000
0x31000A18	CAN1_RMP1	CAN1 Receive Message Pending 1 Register	0x00000000
0x31000A1C	CAN1_RML1	CAN1 Receive Message Lost 1 Register	0x00000000
0x31000A20	CAN1_MBTIF1	CAN1 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x31000A24	CAN1_MBRIF1	CAN1 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x31000A28	CAN1_MBIM1	CAN1 Mailbox Interrupt Mask 1 Register	0x00000000
0x31000A2C	CAN1_RFH1	CAN1 Remote Frame Handling 1 Register	0x00000000
0x31000A30	CAN1_OPSS1	CAN1 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x31000A40	CAN1_MC2	CAN1 Mailbox Configuration 2 Register	0x00000000
0x31000A44	CAN1_MD2	CAN1 Mailbox Direction 2 Register	0x00000000
0x31000A48	CAN1_TRS2	CAN1 Transmission Request Set 2 Register	0x00000000
0x31000A4C	CAN1_TRR2	CAN1 Transmission Request Reset 2 Register	0x00000000
0x31000A50	CAN1_TA2	CAN1 Transmission Acknowledge 2 Register	0x00000000
0x31000A54	CAN1_AA2	CAN1 Abort Acknowledge 2 Register	0x00000000
0x31000A58	CAN1_RMP2	CAN1 Receive Message Pending 2 Register	0x00000000
0x31000A5C	CAN1_RML2	CAN1 Receive Message Lost 2 Register	0x00000000
0x31000A60	CAN1_MBTIF2	CAN1 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x31000A64	CAN1_MBRIF2	CAN1 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x31000A68	CAN1_MBIM2	CAN1 Mailbox Interrupt Mask 2 Register	0x00000000
0x31000A6C	CAN1_RFH2	CAN1 Remote Frame Handling 2 Register	0x00000000
0x31000A70	CAN1_OPSS2	CAN1 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x31000A80	CAN1_CLK	CAN1 Clock Register	0x00000000
0x31000A84	CAN1_TIMING	CAN1 Timing Register	0x00000000
0x31000A88	CAN1_DBG	CAN1 Debug Register	0x00000008
0x31000A8C	CAN1_STAT	CAN1 Status Register	0x00000080
0x31000A90	CAN1_CEC	CAN1 Error Counter Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000A94	CAN1_GIS	CAN1 Global CAN Interrupt Status Register	0x00000000
0x31000A98	CAN1_GIM	CAN1 Global CAN Interrupt Mask Register	0x00000000
0x31000A9C	CAN1_GIF	CAN1 Global CAN Interrupt Flag Register	0x00000000
0x31000AA0	CAN1_CTL	CAN1 CAN Master Control Register	0x00000080
0x31000AA4	CAN1_INT	CAN1 Interrupt Pending Register	0x00000000
0x31000AAC	CAN1_MBTD	CAN1 Temporary Mailbox Disable Register	0x00000000
0x31000AB0	CAN1_EWR	CAN1 Error Counter Warning Level Register	0x00006060
0x31000AB4	CAN1_ESR	CAN1 Error Status Register	0x00000020
0x31000AC4	CAN1_UCCNT	CAN1 Universal Counter Register	0x00000000
0x31000AC8	CAN1_UCRC	CAN1 Universal Counter Reload/Capture Register	0x00000000
0x31000ACC	CAN1_UCCNF	CAN1 Universal Counter Configuration Mode Register	0x00000000
0x31000B00	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B04	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B08	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B0C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B10	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B14	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B18	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B1C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B20	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B24	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B28	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B2C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B30	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B34	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B38	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B3C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B40	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B44	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B48	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B4C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000B50	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B54	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B58	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B5C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B60	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B64	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B68	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B6C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B70	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B74	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B78	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B7C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B80	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B84	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B88	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B8C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B90	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B94	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000B98	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000B9C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BA0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BA4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BA8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BAC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BB0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BB4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BB8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BBC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BC0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BC4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BC8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000BCC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BD0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BD4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BD8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BDC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BE0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BE4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BE8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BEC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BF0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BF4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000BF8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x31000BFC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x31000C00	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000C04	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000C08	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000C0C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000C10	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000C14	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000C18	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000C1C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000C20	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000C24	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000C28	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000C2C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000C30	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000C34	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000C38	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000C3C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000C40	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000C44	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000C48	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000C4C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000C50	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000C54	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000C58	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000C5C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000C60	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000C64	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000C68	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000C6C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000C70	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000C74	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000C78	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000C7C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000C80	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000C84	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000C88	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000C8C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000C90	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000C94	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000C98	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000C9C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000CA0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000CA4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000CA8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000CAC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000CB0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000CB4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000CB8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000CBC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000CC0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000CC4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000CC8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000CCC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000CD0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000CD4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000CD8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000CDC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000CE0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000CE4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000CE8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000CEC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000CF0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000CF4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000CF8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000CFC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000D00	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000D04	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000D08	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000D0C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000D10	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000D14	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000D18	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000D1C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000D20	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000D24	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000D28	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000D2C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000D30	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000D34	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000D38	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000D3C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000D40	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000D44	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000D48	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000D4C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000D50	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000D54	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000D58	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000D5C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000D60	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000D64	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000D68	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000D6C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000D70	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000D74	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000D78	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000D7C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000D80	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000D84	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000D88	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000D8C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000D90	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000D94	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000D98	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000D9C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000DA0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000DA4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000DA8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000DAC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000DB0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000DB4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000DB8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000DBC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000DC0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000DC4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000DC8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000DCC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000DD0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000DD4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000DD8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000DDC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000DE0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000DE4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000DE8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000DEC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000DF0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000DF4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000DF8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000DFC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000E00	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000E04	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000E08	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000E0C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000E10	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000E14	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000E18	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000E1C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000E20	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000E24	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000E28	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000E2C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000E30	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000E34	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000E38	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000E3C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000E40	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000E44	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000E48	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000E4C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000E50	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000E54	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000E58	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000E5C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000E60	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000E64	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000E68	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000E6C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000E70	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000E74	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000E78	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000E7C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000E80	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000E84	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000E88	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000E8C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000E90	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000E94	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000E98	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000E9C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000EA0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000EA4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000EA8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000EAC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000EB0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000EB4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000EB8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000EBC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000EC0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000EC4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000EC8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000ECC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000ED0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000ED4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000ED8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000EDC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000EE0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000EE4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000EE8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000EEC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000EF0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000EF4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000EF8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000EFC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000F00	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000F04	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000F08	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000F0C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000F10	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000F14	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000F18	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000F1C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000F20	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000F24	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000F28	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000F2C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000F30	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000F34	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000F38	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000F3C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000F40	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000F44	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000F48	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000F4C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000F50	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000F54	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000F58	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000F5C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000F60	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000F64	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000F68	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000F6C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000F70	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000F74	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000F78	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000F7C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000F80	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000F84	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000F88	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000F8C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000F90	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000F94	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000F98	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000F9C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000FA0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000FA4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000FA8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000FAC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000FB0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000FB4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000FB8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000FBC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000FC0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-9: ADSP-SC58x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31000FC4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000FC8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000FCC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000FD0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000FD4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000FD8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000FDC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x31000FE0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x31000FE4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x31000FE8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x31000FEC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x31000FF0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x31000FF4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x31000FF8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x31000FFC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-10: ADSP-SC58x SCB1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30200020	SCB1_MST00_SYNC	SCB1 Mst00 Interface Block Sync Mode	0x00000004
0x30300020	SCB2_MST00_SYNC	SCB2 Mst00 Interface Block Sync Mode	0x00000004

Table A-11: ADSP-SC58x CDU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108F000	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F004	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F008	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F00C	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F010	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F014	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001

Table A-11: ADSP-SC58x CDU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108F018	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F01C	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F020	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F024	CDU0_CFG[n]	CDU0 CDU Configuration	0x00000001
0x3108F040	CDU0_STAT	CDU0 CDU Status	0x00000000
0x3108F044	CDU0_CLKINSEL	CDU0 CLKIN Select	0x00000000
0x3108F048	CDU0_REVID	CDU0 CDU Revision ID	0x00000010

Table A-12: ADSP-SC58x CGU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108D000	CGU0_CTL	CGU0 Control Register	0x00001000
0x3108D004	CGU0_PLLCTL	CGU0 PLL Control Register	0x00000000
0x3108D008	CGU0_STAT	CGU0 Status Register	0x0000000D
0x3108D00C	CGU0_DIV	CGU0 Clocks Divisor Register	0x04084844
0x3108D010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x3108D014	CGU0_OSCWDCTL	CGU0 Oscillator Watchdog Register	0x00007F00
0x3108D018	CGU0_TSCTL	CGU0 Time Stamp Control Register	0x00000000
0x3108D01C	CGU0_TSVALUE0	CGU0 Time Stamp Counter Initial 32 LSB Value Register	0x00000000
0x3108D020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x3108D024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x3108D028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000
0x3108D02C	CGU0_CCBF_DIS	CGU0 Core Clock Buffer Disable Register	0x00000000
0x3108D030	CGU0_CCBF_STAT	CGU0 Core Clock Buffer Status Register	0x00000000
0x3108D038	CGU0_SCBF_DIS	CGU0 System Clock Buffer Disable Register	0x00000000
0x3108D03C	CGU0_SCBF_STAT	CGU0 System Clock Buffer Status Register	0x00000000
0x3108D048	CGU0_REVID	CGU0 Revision ID Register	0x00000020

Table A-13: ADSP-SC58x CGU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108E000	CGU1_CTL	CGU1 Control Register	0x00001000

Table A-13: ADSP-SC58x CGU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108E004	CGU1_PLLCTL	CGU1 PLL Control Register	0x00000000
0x3108E008	CGU1_STAT	CGU1 Status Register	0x0000000D
0x3108E00C	CGU1_DIV	CGU1 Clocks Divisor Register	0x04084844
0x3108E014	CGU1_OSCWDCTL	CGU1 Oscillator Watchdog Register	0x00007F00
0x3108E02C	CGU1_CCBF_DIS	CGU1 Core Clock Buffer Disable Register	0x00000000
0x3108E030	CGU1_CCBF_STAT	CGU1 Core Clock Buffer Status Register	0x00000000
0x3108E038	CGU1_SCBF_DIS	CGU1 System Clock Buffer Disable Register	0x00000000
0x3108E03C	CGU1_SCBF_STAT	CGU1 System Clock Buffer Status Register	0x00000000
0x3108E048	CGU1_REVID	CGU1 Revision ID Register	0x00000020

Table A-14: ADSP-SC58x CNT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100B000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x3100B004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x3100B008	CNT0_STAT	CNT0 Status Register	0x00000000
0x3100B00C	CNT0_CMD	CNT0 Command Register	0x00000000
0x3100B010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x3100B014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x3100B018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x3100B01C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000

Table A-15: ADSP-SC58x CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001200	CRC0_CTL	CRC0 Control Register	0x00000000
0x31001204	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x31001208	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x31001214	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x31001218	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000
0x3100121C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x31001220	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000

Table A-15: ADSP-SC58x CRC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31001224	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x31001228	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x3100122C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x31001240	CRC0_STAT	CRC0 Status Register	0x00000000
0x31001244	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x3100124C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x31001250	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table A-16: ADSP-SC58x CRC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001300	CRC1_CTL	CRC1 Control Register	0x00000000
0x31001304	CRC1_DCNT	CRC1 Data Word Count Register	0x00000000
0x31001308	CRC1_DCNTRLD	CRC1 Data Word Count Reload Register	0x00000000
0x31001314	CRC1_COMP	CRC1 Data Compare Register	0x00000000
0x31001318	CRC1_FILLVAL	CRC1 Fill Value Register	0x00000000
0x3100131C	CRC1_DFIFO	CRC1 Data FIFO Register	0x00000000
0x31001320	CRC1_INEN	CRC1 Interrupt Enable Register	0x00000000
0x31001324	CRC1_INEN_SET	CRC1 Interrupt Enable Set Register	0x00000000
0x31001328	CRC1_INEN_CLR	CRC1 Interrupt Enable Clear Register	0x00000000
0x3100132C	CRC1_POLY	CRC1 Polynomial Register	0x00000000
0x31001340	CRC1_STAT	CRC1 Status Register	0x00000000
0x31001344	CRC1_DCNTCAP	CRC1 Data Count Capture Register	0x00000000
0x3100134C	CRC1_RESULT_FIN	CRC1 CRC Final Result Register	0x00000000
0x31001350	CRC1_RESULT_CUR	CRC1 CRC Current Result Register	0x00000000

Table A-17: ADSP-SC58x CSPFT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31103000	CSPFT0_CTL	CSPFT0 Main Control Register	0x20000400
0x31103004	CSPFT0_HWFEAT	CSPFT0 Hardware Feature Register	0x01484002
0x31103008	CSPFT0_TRIGGER	CSPFT0 Trigger Event Register	0x000077EF

Table A-17: ADSP-SC58x CSPFT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31103010	CSPFT0_STAT	CSPFT0 Status Register	0x00000002
0x31103018	CSPFT0_TSSCTL	CSPFT0 TraceEnable Start/Stop Control Register	0x00000000
0x31103020	CSPFT0_TEEVENT	CSPFT0 TraceEnable Event Register	0x000077EF
0x31103024	CSPFT0_TECTL	CSPFT0 TraceEnable Control Register	0x00000000
0x31103040	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103044	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103048	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x3110304C	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x31103080	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103084	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103088	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x3110308C	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x31103140	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x31103144	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x31103150	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x31103154	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x31103160	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x31103164	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x31103170	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x31103174	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x311031A0	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031A4	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031A8	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031AC	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x311031B0	CSPFT0_CIDCVR[n]	CSPFT0 Context ID Comparator Value	0x00000000
0x311031BC	CSPFT0_CIDCMR	CSPFT0 Context ID Comparator Mask Register	0x00000000
0x311031E0	CSPFT0_SYNCFR	CSPFT0 Synchronization Frequency Register	0x00000000
0x311031E8	CSPFT0_CCER	CSPFT0 Configuration Code Extension Register	0x00800000
0x31103200	CSPFT0_TRACEIDR	CSPFT0 CoreSight Trace ID Register	0x00000000
0x31103FA0	CSPFT0_CLAIMSET	CSPFT0 Claim Tag Set Register	0x0000000F
0x31103FA4	CSPFT0_CLAIMCLR	CSPFT0 Claim Tag Clear Register	0x00000000

Table A-17: ADSP-SC58x CSPFT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31103FB0	CSPFT0_LAR	CSPFT0 Lock Access Register	0x00000000
0x31103FB4	CSPFT0_LSR	CSPFT0 Lock Status Register	0x00000003
0x31103FB8	CSPFT0_AUTHSTATUS	CSPFT0 Authentication Status Register	0x00000080
0x31103FCC	CSPFT0_DEVTYPE	CSPFT0 Device Type Identifier Register	0x00000023
0x31103FD0	CSPFT0_PID4	CSPFT0 Peripheral ID4 Register	0x00000000
0x31103FE0	CSPFT0_PID0	CSPFT0 Peripheral ID0 Register	0x00000000
0x31103FE4	CSPFT0_PID1	CSPFT0 Peripheral ID1 Register	0x00000050
0x31103FE8	CSPFT0_PID2	CSPFT0 Peripheral ID2 Register	0x0000000E
0x31103FEC	CSPFT0_PID3	CSPFT0 Peripheral ID3 Register	0x00000000
0x31103FF0	CSPFT0_CID0	CSPFT0 Component ID0 Register	0x0000000D
0x31103FF4	CSPFT0_CID1	CSPFT0 Component ID1 Register	0x00000090
0x31103FF8	CSPFT0_CID2	CSPFT0 Component ID2 Register	0x00000005
0x31103FFC	CSPFT0_CID3	CSPFT0 Component ID3 Register	0x000000B1

Table A-18: ADSP-SC58x CSPFT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31107000	CSPFT1_CTL	CSPFT1 Main Control Register	0x20000400
0x31107004	CSPFT1_HWFEAT	CSPFT1 Hardware Feature Register	0x01484002
0x31107008	CSPFT1_TRIGGER	CSPFT1 Trigger Event Register	0x000077EF
0x31107010	CSPFT1_STAT	CSPFT1 Status Register	0x00000002
0x31107018	CSPFT1_TSSCTL	CSPFT1 TraceEnable Start/Stop Control Register	0x00000000
0x31107020	CSPFT1_TEEVENT	CSPFT1 TraceEnable Event Register	0x000077EF
0x31107024	CSPFT1_TECTL	CSPFT1 TraceEnable Control Register	0x00000000
0x31107040	CSPFT1_ACVR[n]	CSPFT1 Address Comparator Value Register	0x00000000
0x31107044	CSPFT1_ACVR[n]	CSPFT1 Address Comparator Value Register	0x00000000
0x31107048	CSPFT1_ACVR[n]	CSPFT1 Address Comparator Value Register	0x00000000
0x3110704C	CSPFT1_ACVR[n]	CSPFT1 Address Comparator Value Register	0x00000000
0x31107080	CSPFT1_ACTR[n]	CSPFT1 Address Comparator Access Type Register	0x00000000
0x31107084	CSPFT1_ACTR[n]	CSPFT1 Address Comparator Access Type Register	0x00000000
0x31107088	CSPFT1_ACTR[n]	CSPFT1 Address Comparator Access Type Register	0x00000000

Table A-18: ADSP-SC58x CSPFT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3110708C	CSPFT1_ACTR[n]	CSPFT1 Address Comparator Access Type Register	0x00000000
0x31107140	CSPFT1_CNTRLDVR[n]	CSPFT1 Counter Reload Value Register	0x00000000
0x31107144	CSPFT1_CNTRLDVR[n]	CSPFT1 Counter Reload Value Register	0x00000000
0x31107150	CSPFT1_CNTENR[n]	CSPFT1 Counter Enable Event Register	0x000077EF
0x31107154	CSPFT1_CNTENR[n]	CSPFT1 Counter Enable Event Register	0x000077EF
0x31107160	CSPFT1_CNTRLDEVR[n]	CSPFT1 Counter Reload Event Register	0x000077EF
0x31107164	CSPFT1_CNTRLDEVR[n]	CSPFT1 Counter Reload Event Register	0x000077EF
0x31107170	CSPFT1_CNTVR[n]	CSPFT1 Counter Value Register	0x00000000
0x31107174	CSPFT1_CNTVR[n]	CSPFT1 Counter Value Register	0x00000000
0x311071A0	CSPFT1_EXTOUTEVR[n]	CSPFT1 External Output Event Register	0x000077EF
0x311071A4	CSPFT1_EXTOUTEVR[n]	CSPFT1 External Output Event Register	0x000077EF
0x311071A8	CSPFT1_EXTOUTEVR[n]	CSPFT1 External Output Event Register	0x000077EF
0x311071AC	CSPFT1_EXTOUTEVR[n]	CSPFT1 External Output Event Register	0x000077EF
0x311071B0	CSPFT1_CIDCVR[n]	CSPFT1 Context ID Comparator Value	0x00000000
0x311071BC	CSPFT1_CIDCMR	CSPFT1 Context ID Comparator Mask Register	0x00000000
0x311071E0	CSPFT1_SYNCFR	CSPFT1 Synchronization Frequency Register	0x00000000
0x311071E8	CSPFT1_CCER	CSPFT1 Configuration Code Extension Register	0x00800000
0x31107200	CSPFT1_TRACEIDR	CSPFT1 CoreSight Trace ID Register	0x00000000
0x31107FA0	CSPFT1_CLAIMSET	CSPFT1 Claim Tag Set Register	0x0000000F
0x31107FA4	CSPFT1_CLAIMCLR	CSPFT1 Claim Tag Clear Register	0x00000000
0x31107FB0	CSPFT1_LAR	CSPFT1 Lock Access Register	0x00000000
0x31107FB4	CSPFT1_LSR	CSPFT1 Lock Status Register	0x00000003
0x31107FB8	CSPFT1_AUTHSTATUS	CSPFT1 Authentication Status Register	0x00000080
0x31107FCC	CSPFT1_DEVTYPE	CSPFT1 Device Type Identifier Register	0x00000023
0x31107FD0	CSPFT1_PID4	CSPFT1 Peripheral ID4 Register	0x00000000
0x31107FE0	CSPFT1_PID0	CSPFT1 Peripheral ID0 Register	0x00000000
0x31107FE4	CSPFT1_PID1	CSPFT1 Peripheral ID1 Register	0x00000050
0x31107FE8	CSPFT1_PID2	CSPFT1 Peripheral ID2 Register	0x0000000E
0x31107FEC	CSPFT1_PID3	CSPFT1 Peripheral ID3 Register	0x00000000
0x31107FF0	CSPFT1_CID0	CSPFT1 Component ID0 Register	0x0000000D
0x31107FF4	CSPFT1_CID1	CSPFT1 Component ID1 Register	0x00000090

Table A-18: ADSP-SC58x CSPFT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31107FF8	CSPFT1_CID2	CSPFT1 Component ID2 Register	0x00000005
0x31107FFC	CSPFT1_CID3	CSPFT1 Component ID3 Register	0x000000B1

Table A-19: ADSP-SC58x DAI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C90C0	DAI0_CLK0	DAI0 Clock Routing Control Register 0	0x252630C2
0x310C90C4	DAI0_CLK1	DAI0 Clock Routing Control Register 1	0x3DEF7BDE
0x310C90C8	DAI0_CLK2	DAI0 Clock Routing Control Register 2	0x00007BDE
0x310C90CC	DAI0_CLK3	DAI0 Clock Routing Control Register 3	0x3C000000
0x310C90D0	DAI0_CLK4	DAI0 Clock Routing Control Register 4	0x3DEF03DE
0x310C90D4	DAI0_CLK5	DAI0 Clock Routing Control Register 5	0x000003DE
0x310C9100	DAI0_DAT0	DAI0 Serial Data Routing Control Register 0	0x08144040
0x310C9104	DAI0_DAT1	DAI0 Serial Data Routing Control Register 1	0x0F38B289
0x310C9108	DAI0_DAT2	DAI0 Serial Data Routing Control Register 2	0x00000450
0x310C910C	DAI0_DAT3	DAI0 Serial Data Routing Control Register 3	0x00000000
0x310C9110	DAI0_DAT4	DAI0 Serial Data Routing Control Register 4	0x00000000
0x310C9114	DAI0_DAT5	DAI0 Serial Data Routing Control Register 5	0x00000000
0x310C9118	DAI0_DAT6	DAI0 Serial Data Routing Control Register 6	0x00FBEFBE
0x310C9140	DAI0_FS0	DAI0 Frame Sync Routing Control Register 0	0x2736B4E3
0x310C9144	DAI0_FS1	DAI0 Frame Sync Routing Control Register 1	0x3DEF7BDE
0x310C9148	DAI0_FS2	DAI0 Frame Sync Routing Control Register 2	0x00007BDE
0x310C9150	DAI0_FS4	DAI0 Frame Sync Routing Control Register 4	0x000003DE
0x310C9180	DAI0_PIN0	DAI0 Pin Buffer Assignment Register 0	0x04C80A94
0x310C9184	DAI0_PIN1	DAI0 Pin Buffer Assignment Register 1	0x04E84B96
0x310C9188	DAI0_PIN2	DAI0 Pin Buffer Assignment Register 2	0x03668C98
0x310C918C	DAI0_PIN3	DAI0 Pin Buffer Assignment Register 3	0x03A714A3
0x310C9190	DAI0_PIN4	DAI0 Pin Buffer Assignment Register 4	0x05694F9E
0x310C91C0	DAI0_MISC0	DAI0 Miscellaneous Control Register 0	0x3DEF7BDE
0x310C91C4	DAI0_MISC1	DAI0 Miscellaneous Control Register 1	0x3DEF7BDE
0x310C91E0	DAI0_PBEN0	DAI0 Pin Buffer Enable Register 0	0x0E2482CA

Table A-19: ADSP-SC58x DAI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C91E4	DAI0_PBEN1	DAI0 Pin Buffer Enable Register 1	0x1348D30F
0x310C91E8	DAI0_PBEN2	DAI0 Pin Buffer Enable Register 2	0x1A5545D6
0x310C91EC	DAI0_PBEN3	DAI0 Pin Buffer Enable Register 3	0x1D71F79B
0x310C9200	DAI0_IMSK_FE	DAI0 Falling-Edge Interrupt Mask Register	0x00000000
0x310C9204	DAI0_IMSK_RE	DAI0 Rising-Edge Interrupt Mask Register	0x00000000
0x310C9210	DAI0_IMSK_PRI	DAI0 Core Interrupt Priority Assignment Register	0x00000000
0x310C9220	DAI0_IRPTL_H	DAI0 High Priority Interrupt Latch Register	0x00000000
0x310C9224	DAI0_IRPTL_L	DAI0 Low Priority Interrupt Latch Register	0x00000000
0x310C9230	DAI0_IRPTL_HS	DAI0 Shadow High Priority Interrupt Latch Register	0x00000000
0x310C9234	DAI0_IRPTL_LS	DAI0 Shadow Low Priority Interrupt Latch Register	0x00000000
0x310C92E4	DAI0_PIN_STAT	DAI0 Pin Status Register	0x00000000

Table A-20: ADSP-SC58x DAI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB0C0	DAI1_CLK0	DAI1 Clock Routing Control Register 0	0x252630C2
0x310CB0C4	DAI1_CLK1	DAI1 Clock Routing Control Register 1	0x3DEF7BDE
0x310CB0C8	DAI1_CLK2	DAI1 Clock Routing Control Register 2	0x00007BDE
0x310CB0CC	DAI1_CLK3	DAI1 Clock Routing Control Register 3	0x3C000000
0x310CB0D0	DAI1_CLK4	DAI1 Clock Routing Control Register 4	0x3DEF03DE
0x310CB0D4	DAI1_CLK5	DAI1 Clock Routing Control Register 5	0x000003DE
0x310CB100	DAI1_DAT0	DAI1 Serial Data Routing Control Register 0	0x08144040
0x310CB104	DAI1_DAT1	DAI1 Serial Data Routing Control Register 1	0x0F38B289
0x310CB108	DAI1_DAT2	DAI1 Serial Data Routing Control Register 2	0x00000450
0x310CB10C	DAI1_DAT3	DAI1 Serial Data Routing Control Register 3	0x00000000
0x310CB110	DAI1_DAT4	DAI1 Serial Data Routing Control Register 4	0x00000000
0x310CB114	DAI1_DAT5	DAI1 Serial Data Routing Control Register 5	0x00000000
0x310CB118	DAI1_DAT6	DAI1 Serial Data Routing Control Register 6	0x00FBEFBE
0x310CB140	DAI1_FS0	DAI1 Frame Sync Routing Control Register 0	0x2736B4E3
0x310CB144	DAI1_FS1	DAI1 Frame Sync Routing Control Register 1	0x3DEF7BDE
0x310CB148	DAI1_FS2	DAI1 Frame Sync Routing Control Register 2	0x00007BDE

Table A-20: ADSP-SC58x DAI1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB150	DAI1_FS4	DAI1 Frame Sync Routing Control Register 4	0x000003DE
0x310CB180	DAI1_PIN0	DAI1 Pin Buffer Assignment Register 0	0x04C80A94
0x310CB184	DAI1_PIN1	DAI1 Pin Buffer Assignment Register 1	0x04E84B96
0x310CB188	DAI1_PIN2	DAI1 Pin Buffer Assignment Register 2	0x03668C98
0x310CB18C	DAI1_PIN3	DAI1 Pin Buffer Assignment Register 3	0x03A714A3
0x310CB190	DAI1_PIN4	DAI1 Pin Buffer Assignment Register 4	0x05694F9E
0x310CB1C0	DAI1_MISC0	DAI1 Miscellaneous Control Register 0	0x3DEF7BDE
0x310CB1C4	DAI1_MISC1	DAI1 Miscellaneous Control Register 1	0x3DEF7BDE
0x310CB1E0	DAI1_PBEN0	DAI1 Pin Buffer Enable Register 0	0x0E2482CA
0x310CB1E4	DAI1_PBEN1	DAI1 Pin Buffer Enable Register 1	0x1348D30F
0x310CB1E8	DAI1_PBEN2	DAI1 Pin Buffer Enable Register 2	0x1A5545D6
0x310CB1EC	DAI1_PBEN3	DAI1 Pin Buffer Enable Register 3	0x1D71F79B
0x310CB200	DAI1_IMSK_FE	DAI1 Falling-Edge Interrupt Mask Register	0x00000000
0x310CB204	DAI1_IMSK_RE	DAI1 Rising-Edge Interrupt Mask Register	0x00000000
0x310CB210	DAI1_IMSK_PRI	DAI1 Core Interrupt Priority Assignment Register	0x00000000
0x310CB220	DAI1_IRPTL_H	DAI1 High Priority Interrupt Latch Register	0x00000000
0x310CB224	DAI1_IRPTL_L	DAI1 Low Priority Interrupt Latch Register	0x00000000
0x310CB230	DAI1_IRPTL_HS	DAI1 Shadow High Priority Interrupt Latch Register	0x00000000
0x310CB234	DAI1_IRPTL_LS	DAI1 Shadow Low Priority Interrupt Latch Register	0x00000000
0x310CB2E4	DAI1_PIN_STAT	DAI1 Pin Status Register	0x00000000

Table A-21: ADSP-SC58x DAPROM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31100000	DAPROM0_ROMENTRY00	DAPROM0 ROM Entry 00	0x00001003
0x31100004	DAPROM0_ROMENTRY01	DAPROM0 ROM Entry 01	0x00002003
0x31100008	DAPROM0_ROMENTRY02	DAPROM0 ROM Entry 02	0x00003003
0x3110000C	DAPROM0_ROMENTRY03	DAPROM0 ROM Entry 03	0x00004003
0x31100010	DAPROM0_ROMENTRY04	DAPROM0 ROM Entry 04	0x00005003
0x31100014	DAPROM0_ROMENTRY05	DAPROM0 ROM Entry 05	0x00006003
0x31100018	DAPROM0_ROMENTRY06	DAPROM0 ROM Entry 06	0x00007003

Table A-21: ADSP-SC58x DAPROM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3110001C	DAPROM0_ROMENTRY07	DAPROM0 ROM Entry 07	0x00008003
0x31100020	DAPROM0_ROMENTRY08	DAPROM0 ROM Entry 08	0x00009003
0x31100024	DAPROM0_ROMENTRY09	DAPROM0 ROM Entry 09	0x0000A003
0x31100028	DAPROM0_ROMENTRY10	DAPROM0 ROM Entry 10	0x0000B003
0x3110002C	DAPROM0_ROMENTRY11	DAPROM0 ROM Entry 11	0x0000C003
0x31100030	DAPROM0_ROMENTRY12	DAPROM0 ROM Entry 12	0x0000D003
0x31100034	DAPROM0_ROMENTRY13	DAPROM0 ROM Entry 13	0x00010003

Table A-22: ADSP-SC58x DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022000	DMA0_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x31022004	DMA0_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x31022008	DMA0_CFG	DMA0 Configuration Register	0x00000000
0x3102200C	DMA0_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x31022010	DMA0_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x31022014	DMA0_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022018	DMA0_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022024	DMA0_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x31022028	DMA0_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x3102202C	DMA0_ADDR_CUR	DMA0 Current Address Register	0x00000000
0x31022030	DMA0_STAT	DMA0 Status Register	0x00006000
0x31022034	DMA0_XCNT_CUR	DMA0 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022038	DMA0_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000

Table A-23: ADSP-SC58x DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022080	DMA1_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x31022084	DMA1_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x31022088	DMA1_CFG	DMA1 Configuration Register	0x00000000

Table A-23: ADSP-SC58x DMA1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102208C	DMA1_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x31022090	DMA1_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x31022094	DMA1_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022098	DMA1_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x310220A4	DMA1_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x310220A8	DMA1_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000
0x310220AC	DMA1_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x310220B0	DMA1_STAT	DMA1 Status Register	0x00006000
0x310220B4	DMA1_XCNT_CUR	DMA1 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310220B8	DMA1_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000

Table A-24: ADSP-SC58x DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024000	DMA10_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000
0x31024004	DMA10_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000
0x31024008	DMA10_CFG	DMA10 Configuration Register	0x00000000
0x3102400C	DMA10_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x31024010	DMA10_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x31024014	DMA10_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024018	DMA10_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x31024024	DMA10_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x31024028	DMA10_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x3102402C	DMA10_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x31024030	DMA10_STAT	DMA10 Status Register	0x00006000
0x31024034	DMA10_XCNT_CUR	DMA10 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31024038	DMA10_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000

Table A-25: ADSP-SC58x DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024080	DMA11_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x31024084	DMA11_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x31024088	DMA11_CFG	DMA11 Configuration Register	0x00000000
0x3102408C	DMA11_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000
0x31024090	DMA11_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x31024094	DMA11_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024098	DMA11_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x310240A4	DMA11_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000
0x310240A8	DMA11_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000
0x310240AC	DMA11_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x310240B0	DMA11_STAT	DMA11 Status Register	0x00006000
0x310240B4	DMA11_XCNT_CUR	DMA11 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310240B8	DMA11_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000

Table A-26: ADSP-SC58x DMA12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024100	DMA12_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000
0x31024104	DMA12_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x31024108	DMA12_CFG	DMA12 Configuration Register	0x00000000
0x3102410C	DMA12_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x31024110	DMA12_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x31024114	DMA12_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024118	DMA12_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x31024124	DMA12_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x31024128	DMA12_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x3102412C	DMA12_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x31024130	DMA12_STAT	DMA12 Status Register	0x00006000

Table A-26: ADSP-SC58x DMA12 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31024134	DMA12_XCNT_CUR	DMA12 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31024138	DMA12_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000

Table A-27: ADSP-SC58x DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024180	DMA13_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x31024184	DMA13_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000
0x31024188	DMA13_CFG	DMA13 Configuration Register	0x00000000
0x3102418C	DMA13_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x31024190	DMA13_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x31024194	DMA13_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024198	DMA13_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x310241A4	DMA13_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000
0x310241A8	DMA13_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x310241AC	DMA13_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x310241B0	DMA13_STAT	DMA13 Status Register	0x00006000
0x310241B4	DMA13_XCNT_CUR	DMA13 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310241B8	DMA13_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000

Table A-28: ADSP-SC58x DMA14 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024200	DMA14_DSCPTR_NXT	DMA14 Pointer to Next Initial Descriptor Register	0x00000000
0x31024204	DMA14_ADDRSTART	DMA14 Start Address of Current Buffer Register	0x00000000
0x31024208	DMA14_CFG	DMA14 Configuration Register	0x00000000
0x3102420C	DMA14_XCNT	DMA14 Inner Loop Count Start Value Register	0x00000000
0x31024210	DMA14_XMOD	DMA14 Inner Loop Address Increment Register	0x00000000
0x31024214	DMA14_YCNT	DMA14 Outer Loop Count Start Value (2D only) Register	0x00000000

Table A-28: ADSP-SC58x DMA14 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31024218	DMA14_YMOD	DMA14 Outer Loop Address Increment (2D only) Register	0x00000000
0x31024224	DMA14_DSCPTR_CUR	DMA14 Current Descriptor Pointer Register	0x00000000
0x31024228	DMA14_DSCPTR_PRV	DMA14 Previous Initial Descriptor Pointer Register	0x00000000
0x3102422C	DMA14_ADDR_CUR	DMA14 Current Address Register	0x00000000
0x31024230	DMA14_STAT	DMA14 Status Register	0x00006000
0x31024234	DMA14_XCNT_CUR	DMA14 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31024238	DMA14_YCNT_CUR	DMA14 Current Row Count (2D only) Register	0x00000000

Table A-29: ADSP-SC58x DMA15 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024280	DMA15_DSCPTR_NXT	DMA15 Pointer to Next Initial Descriptor Register	0x00000000
0x31024284	DMA15_ADDRSTART	DMA15 Start Address of Current Buffer Register	0x00000000
0x31024288	DMA15_CFG	DMA15 Configuration Register	0x00000000
0x3102428C	DMA15_XCNT	DMA15 Inner Loop Count Start Value Register	0x00000000
0x31024290	DMA15_XMOD	DMA15 Inner Loop Address Increment Register	0x00000000
0x31024294	DMA15_YCNT	DMA15 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024298	DMA15_YMOD	DMA15 Outer Loop Address Increment (2D only) Register	0x00000000
0x310242A4	DMA15_DSCPTR_CUR	DMA15 Current Descriptor Pointer Register	0x00000000
0x310242A8	DMA15_DSCPTR_PRV	DMA15 Previous Initial Descriptor Pointer Register	0x00000000
0x310242AC	DMA15_ADDR_CUR	DMA15 Current Address Register	0x00000000
0x310242B0	DMA15_STAT	DMA15 Status Register	0x00006000
0x310242B4	DMA15_XCNT_CUR	DMA15 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310242B8	DMA15_YCNT_CUR	DMA15 Current Row Count (2D only) Register	0x00000000

Table A-30: ADSP-SC58x DMA16 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024300	DMA16_DSCPTR_NXT	DMA16 Pointer to Next Initial Descriptor Register	0x00000000

Table A-30: ADSP-SC58x DMA16 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31024304	DMA16_ADDRSTART	DMA16 Start Address of Current Buffer Register	0x00000000
0x31024308	DMA16_CFG	DMA16 Configuration Register	0x00000000
0x3102430C	DMA16_XCNT	DMA16 Inner Loop Count Start Value Register	0x00000000
0x31024310	DMA16_XMOD	DMA16 Inner Loop Address Increment Register	0x00000000
0x31024314	DMA16_YCNT	DMA16 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024318	DMA16_YMOD	DMA16 Outer Loop Address Increment (2D only) Register	0x00000000
0x31024324	DMA16_DSCPTR_CUR	DMA16 Current Descriptor Pointer Register	0x00000000
0x31024328	DMA16_DSCPTR_PRV	DMA16 Previous Initial Descriptor Pointer Register	0x00000000
0x3102432C	DMA16_ADDR_CUR	DMA16 Current Address Register	0x00000000
0x31024330	DMA16_STAT	DMA16 Status Register	0x00006000
0x31024334	DMA16_XCNT_CUR	DMA16 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31024338	DMA16_YCNT_CUR	DMA16 Current Row Count (2D only) Register	0x00000000

Table A-31: ADSP-SC58x DMA17 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31024380	DMA17_DSCPTR_NXT	DMA17 Pointer to Next Initial Descriptor Register	0x00000000
0x31024384	DMA17_ADDRSTART	DMA17 Start Address of Current Buffer Register	0x00000000
0x31024388	DMA17_CFG	DMA17 Configuration Register	0x00000000
0x3102438C	DMA17_XCNT	DMA17 Inner Loop Count Start Value Register	0x00000000
0x31024390	DMA17_XMOD	DMA17 Inner Loop Address Increment Register	0x00000000
0x31024394	DMA17_YCNT	DMA17 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31024398	DMA17_YMOD	DMA17 Outer Loop Address Increment (2D only) Register	0x00000000
0x310243A4	DMA17_DSCPTR_CUR	DMA17 Current Descriptor Pointer Register	0x00000000
0x310243A8	DMA17_DSCPTR_PRV	DMA17 Previous Initial Descriptor Pointer Register	0x00000000
0x310243AC	DMA17_ADDR_CUR	DMA17 Current Address Register	0x00000000
0x310243B0	DMA17_STAT	DMA17 Status Register	0x00006000

Table A-31: ADSP-SC58x DMA17 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310243B4	DMA17_XCNT_CUR	DMA17 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310243B8	DMA17_YCNT_CUR	DMA17 Current Row Count (2D only) Register	0x00000000

Table A-32: ADSP-SC58x DMA18 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31028100	DMA18_DSCPTR_NXT	DMA18 Pointer to Next Initial Descriptor Register	0x00000000
0x31028104	DMA18_ADDRSTART	DMA18 Start Address of Current Buffer Register	0x00000000
0x31028108	DMA18_CFG	DMA18 Configuration Register	0x00000000
0x3102810C	DMA18_XCNT	DMA18 Inner Loop Count Start Value Register	0x00000000
0x31028110	DMA18_XMOD	DMA18 Inner Loop Address Increment Register	0x00000000
0x31028114	DMA18_YCNT	DMA18 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31028118	DMA18_YMOD	DMA18 Outer Loop Address Increment (2D only) Register	0x00000000
0x31028124	DMA18_DSCPTR_CUR	DMA18 Current Descriptor Pointer Register	0x00000000
0x31028128	DMA18_DSCPTR_PRV	DMA18 Previous Initial Descriptor Pointer Register	0x00000000
0x3102812C	DMA18_ADDR_CUR	DMA18 Current Address Register	0x00000000
0x31028130	DMA18_STAT	DMA18 Status Register	0x00006000
0x31028134	DMA18_XCNT_CUR	DMA18 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31028138	DMA18_YCNT_CUR	DMA18 Current Row Count (2D only) Register	0x00000000
0x31028140	DMA18_BWLCNT	DMA18 Bandwidth Limit Count Register	0x00000000
0x31028144	DMA18_BWLCNT_CUR	DMA18 Bandwidth Limit Count Current Register	0x00000000
0x31028148	DMA18_BWMCNT	DMA18 Bandwidth Monitor Count Register	0x00000000
0x3102814C	DMA18_BWMCNT_CUR	DMA18 Bandwidth Monitor Count Current Register	0x00000000

Table A-33: ADSP-SC58x DMA19 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31028180	DMA19_DSCPTR_NXT	DMA19 Pointer to Next Initial Descriptor Register	0x00000000
0x31028184	DMA19_ADDRSTART	DMA19 Start Address of Current Buffer Register	0x00000000

Table A-33: ADSP-SC58x DMA19 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31028188	DMA19_CFG	DMA19 Configuration Register	0x00000000
0x3102818C	DMA19_XCNT	DMA19 Inner Loop Count Start Value Register	0x00000000
0x31028190	DMA19_XMOD	DMA19 Inner Loop Address Increment Register	0x00000000
0x31028194	DMA19_YCNT	DMA19 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31028198	DMA19_YMOD	DMA19 Outer Loop Address Increment (2D only) Register	0x00000000
0x310281A4	DMA19_DSCPTR_CUR	DMA19 Current Descriptor Pointer Register	0x00000000
0x310281A8	DMA19_DSCPTR_PRV	DMA19 Previous Initial Descriptor Pointer Register	0x00000000
0x310281AC	DMA19_ADDR_CUR	DMA19 Current Address Register	0x00000000
0x310281B0	DMA19_STAT	DMA19 Status Register	0x00006000
0x310281B4	DMA19_XCNT_CUR	DMA19 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310281B8	DMA19_YCNT_CUR	DMA19 Current Row Count (2D only) Register	0x00000000
0x310281C0	DMA19_BWLCNT	DMA19 Bandwidth Limit Count Register	0x00000000
0x310281C4	DMA19_BWLCNT_CUR	DMA19 Bandwidth Limit Count Current Register	0x00000000
0x310281C8	DMA19_BWMCNT	DMA19 Bandwidth Monitor Count Register	0x00000000
0x310281CC	DMA19_BWMCNT_CUR	DMA19 Bandwidth Monitor Count Current Register	0x00000000

Table A-34: ADSP-SC58x DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022100	DMA2_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x31022104	DMA2_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x31022108	DMA2_CFG	DMA2 Configuration Register	0x00000000
0x3102210C	DMA2_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x31022110	DMA2_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x31022114	DMA2_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022118	DMA2_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022124	DMA2_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x31022128	DMA2_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000
0x3102212C	DMA2_ADDR_CUR	DMA2 Current Address Register	0x00000000

Table A-34: ADSP-SC58x DMA2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31022130	DMA2_STAT	DMA2 Status Register	0x00006000
0x31022134	DMA2_XCNT_CUR	DMA2 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022138	DMA2_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000

Table A-35: ADSP-SC58x DMA20 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026080	DMA20_DSCPTR_NXT	DMA20 Pointer to Next Initial Descriptor Register	0x00000000
0x31026084	DMA20_ADDRSTART	DMA20 Start Address of Current Buffer Register	0x00000000
0x31026088	DMA20_CFG	DMA20 Configuration Register	0x00000000
0x3102608C	DMA20_XCNT	DMA20 Inner Loop Count Start Value Register	0x00000000
0x31026090	DMA20_XMOD	DMA20 Inner Loop Address Increment Register	0x00000000
0x31026094	DMA20_YCNT	DMA20 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026098	DMA20_YMOD	DMA20 Outer Loop Address Increment (2D only) Register	0x00000000
0x310260A4	DMA20_DSCPTR_CUR	DMA20 Current Descriptor Pointer Register	0x00000000
0x310260A8	DMA20_DSCPTR_PRV	DMA20 Previous Initial Descriptor Pointer Register	0x00000000
0x310260AC	DMA20_ADDR_CUR	DMA20 Current Address Register	0x00000000
0x310260B0	DMA20_STAT	DMA20 Status Register	0x00006000
0x310260B4	DMA20_XCNT_CUR	DMA20 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310260B8	DMA20_YCNT_CUR	DMA20 Current Row Count (2D only) Register	0x00000000

Table A-36: ADSP-SC58x DMA21 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026000	DMA21_DSCPTR_NXT	DMA21 Pointer to Next Initial Descriptor Register	0x00000000
0x31026004	DMA21_ADDRSTART	DMA21 Start Address of Current Buffer Register	0x00000000
0x31026008	DMA21_CFG	DMA21 Configuration Register	0x00000000
0x3102600C	DMA21_XCNT	DMA21 Inner Loop Count Start Value Register	0x00000000
0x31026010	DMA21_XMOD	DMA21 Inner Loop Address Increment Register	0x00000000

Table A-36: ADSP-SC58x DMA21 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026014	DMA21_YCNT	DMA21 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026018	DMA21_YMOD	DMA21 Outer Loop Address Increment (2D only) Register	0x00000000
0x31026024	DMA21_DSCPTR_CUR	DMA21 Current Descriptor Pointer Register	0x00000000
0x31026028	DMA21_DSCPTR_PRV	DMA21 Previous Initial Descriptor Pointer Register	0x00000000
0x3102602C	DMA21_ADDR_CUR	DMA21 Current Address Register	0x00000000
0x31026030	DMA21_STAT	DMA21 Status Register	0x00006000
0x31026034	DMA21_XCNT_CUR	DMA21 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026038	DMA21_YCNT_CUR	DMA21 Current Row Count (2D only) Register	0x00000000

Table A-37: ADSP-SC58x DMA22 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046000	DMA22_DSCPTR_NXT	DMA22 Pointer to Next Initial Descriptor Register	0x00000000
0x31046004	DMA22_ADDRSTART	DMA22 Start Address of Current Buffer Register	0x00000000
0x31046008	DMA22_CFG	DMA22 Configuration Register	0x00000000
0x3104600C	DMA22_XCNT	DMA22 Inner Loop Count Start Value Register	0x00000000
0x31046010	DMA22_XMOD	DMA22 Inner Loop Address Increment Register	0x00000000
0x31046014	DMA22_YCNT	DMA22 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046018	DMA22_YMOD	DMA22 Outer Loop Address Increment (2D only) Register	0x00000000
0x31046024	DMA22_DSCPTR_CUR	DMA22 Current Descriptor Pointer Register	0x00000000
0x31046028	DMA22_DSCPTR_PRV	DMA22 Previous Initial Descriptor Pointer Register	0x00000000
0x3104602C	DMA22_ADDR_CUR	DMA22 Current Address Register	0x00000000
0x31046030	DMA22_STAT	DMA22 Status Register	0x00006000
0x31046034	DMA22_XCNT_CUR	DMA22 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31046038	DMA22_YCNT_CUR	DMA22 Current Row Count (2D only) Register	0x00000000

Table A-38: ADSP-SC58x DMA23 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046080	DMA23_DSCPTR_NXT	DMA23 Pointer to Next Initial Descriptor Register	0x00000000
0x31046084	DMA23_ADDRSTART	DMA23 Start Address of Current Buffer Register	0x00000000
0x31046088	DMA23_CFG	DMA23 Configuration Register	0x00000000
0x3104608C	DMA23_XCNT	DMA23 Inner Loop Count Start Value Register	0x00000000
0x31046090	DMA23_XMOD	DMA23 Inner Loop Address Increment Register	0x00000000
0x31046094	DMA23_YCNT	DMA23 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046098	DMA23_YMOD	DMA23 Outer Loop Address Increment (2D only) Register	0x00000000
0x310460A4	DMA23_DSCPTR_CUR	DMA23 Current Descriptor Pointer Register	0x00000000
0x310460A8	DMA23_DSCPTR_PRV	DMA23 Previous Initial Descriptor Pointer Register	0x00000000
0x310460AC	DMA23_ADDR_CUR	DMA23 Current Address Register	0x00000000
0x310460B0	DMA23_STAT	DMA23 Status Register	0x00006000
0x310460B4	DMA23_XCNT_CUR	DMA23 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310460B8	DMA23_YCNT_CUR	DMA23 Current Row Count (2D only) Register	0x00000000

Table A-39: ADSP-SC58x DMA24 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046100	DMA24_DSCPTR_NXT	DMA24 Pointer to Next Initial Descriptor Register	0x00000000
0x31046104	DMA24_ADDRSTART	DMA24 Start Address of Current Buffer Register	0x00000000
0x31046108	DMA24_CFG	DMA24 Configuration Register	0x00000000
0x3104610C	DMA24_XCNT	DMA24 Inner Loop Count Start Value Register	0x00000000
0x31046110	DMA24_XMOD	DMA24 Inner Loop Address Increment Register	0x00000000
0x31046114	DMA24_YCNT	DMA24 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046118	DMA24_YMOD	DMA24 Outer Loop Address Increment (2D only) Register	0x00000000
0x31046124	DMA24_DSCPTR_CUR	DMA24 Current Descriptor Pointer Register	0x00000000
0x31046128	DMA24_DSCPTR_PRV	DMA24 Previous Initial Descriptor Pointer Register	0x00000000
0x3104612C	DMA24_ADDR_CUR	DMA24 Current Address Register	0x00000000
0x31046130	DMA24_STAT	DMA24 Status Register	0x00006000

Table A-39: ADSP-SC58x DMA24 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31046134	DMA24_XCNT_CUR	DMA24 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31046138	DMA24_YCNT_CUR	DMA24 Current Row Count (2D only) Register	0x00000000

Table A-40: ADSP-SC58x DMA25 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046180	DMA25_DSCPTR_NXT	DMA25 Pointer to Next Initial Descriptor Register	0x00000000
0x31046184	DMA25_ADDRSTART	DMA25 Start Address of Current Buffer Register	0x00000000
0x31046188	DMA25_CFG	DMA25 Configuration Register	0x00000000
0x3104618C	DMA25_XCNT	DMA25 Inner Loop Count Start Value Register	0x00000000
0x31046190	DMA25_XMOD	DMA25 Inner Loop Address Increment Register	0x00000000
0x31046194	DMA25_YCNT	DMA25 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046198	DMA25_YMOD	DMA25 Outer Loop Address Increment (2D only) Register	0x00000000
0x310461A4	DMA25_DSCPTR_CUR	DMA25 Current Descriptor Pointer Register	0x00000000
0x310461A8	DMA25_DSCPTR_PRV	DMA25 Previous Initial Descriptor Pointer Register	0x00000000
0x310461AC	DMA25_ADDR_CUR	DMA25 Current Address Register	0x00000000
0x310461B0	DMA25_STAT	DMA25 Status Register	0x00006000
0x310461B4	DMA25_XCNT_CUR	DMA25 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310461B8	DMA25_YCNT_CUR	DMA25 Current Row Count (2D only) Register	0x00000000

Table A-41: ADSP-SC58x DMA26 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046200	DMA26_DSCPTR_NXT	DMA26 Pointer to Next Initial Descriptor Register	0x00000000
0x31046204	DMA26_ADDRSTART	DMA26 Start Address of Current Buffer Register	0x00000000
0x31046208	DMA26_CFG	DMA26 Configuration Register	0x00000000
0x3104620C	DMA26_XCNT	DMA26 Inner Loop Count Start Value Register	0x00000000
0x31046210	DMA26_XMOD	DMA26 Inner Loop Address Increment Register	0x00000000
0x31046214	DMA26_YCNT	DMA26 Outer Loop Count Start Value (2D only) Register	0x00000000

Table A-41: ADSP-SC58x DMA26 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31046218	DMA26_YMOD	DMA26 Outer Loop Address Increment (2D only) Register	0x00000000
0x31046224	DMA26_DSCPTR_CUR	DMA26 Current Descriptor Pointer Register	0x00000000
0x31046228	DMA26_DSCPTR_PRV	DMA26 Previous Initial Descriptor Pointer Register	0x00000000
0x3104622C	DMA26_ADDR_CUR	DMA26 Current Address Register	0x00000000
0x31046230	DMA26_STAT	DMA26 Status Register	0x00006000
0x31046234	DMA26_XCNT_CUR	DMA26 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31046238	DMA26_YCNT_CUR	DMA26 Current Row Count (2D only) Register	0x00000000

Table A-42: ADSP-SC58x DMA27 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046280	DMA27_DSCPTR_NXT	DMA27 Pointer to Next Initial Descriptor Register	0x00000000
0x31046284	DMA27_ADDRSTART	DMA27 Start Address of Current Buffer Register	0x00000000
0x31046288	DMA27_CFG	DMA27 Configuration Register	0x00000000
0x3104628C	DMA27_XCNT	DMA27 Inner Loop Count Start Value Register	0x00000000
0x31046290	DMA27_XMOD	DMA27 Inner Loop Address Increment Register	0x00000000
0x31046294	DMA27_YCNT	DMA27 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046298	DMA27_YMOD	DMA27 Outer Loop Address Increment (2D only) Register	0x00000000
0x310462A4	DMA27_DSCPTR_CUR	DMA27 Current Descriptor Pointer Register	0x00000000
0x310462A8	DMA27_DSCPTR_PRV	DMA27 Previous Initial Descriptor Pointer Register	0x00000000
0x310462AC	DMA27_ADDR_CUR	DMA27 Current Address Register	0x00000000
0x310462B0	DMA27_STAT	DMA27 Status Register	0x00006000
0x310462B4	DMA27_XCNT_CUR	DMA27 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310462B8	DMA27_YCNT_CUR	DMA27 Current Row Count (2D only) Register	0x00000000

Table A-43: ADSP-SC58x DMA28 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046300	DMA28_DSCPTR_NXT	DMA28 Pointer to Next Initial Descriptor Register	0x00000000

Table A-43: ADSP-SC58x DMA28 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31046304	DMA28_ADDRSTART	DMA28 Start Address of Current Buffer Register	0x00000000
0x31046308	DMA28_CFG	DMA28 Configuration Register	0x00000000
0x3104630C	DMA28_XCNT	DMA28 Inner Loop Count Start Value Register	0x00000000
0x31046310	DMA28_XMOD	DMA28 Inner Loop Address Increment Register	0x00000000
0x31046314	DMA28_YCNT	DMA28 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046318	DMA28_YMOD	DMA28 Outer Loop Address Increment (2D only) Register	0x00000000
0x31046324	DMA28_DSCPTR_CUR	DMA28 Current Descriptor Pointer Register	0x00000000
0x31046328	DMA28_DSCPTR_PRV	DMA28 Previous Initial Descriptor Pointer Register	0x00000000
0x3104632C	DMA28_ADDR_CUR	DMA28 Current Address Register	0x00000000
0x31046330	DMA28_STAT	DMA28 Status Register	0x00006000
0x31046334	DMA28_XCNT_CUR	DMA28 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31046338	DMA28_YCNT_CUR	DMA28 Current Row Count (2D only) Register	0x00000000

Table A-44: ADSP-SC58x DMA29 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31046380	DMA29_DSCPTR_NXT	DMA29 Pointer to Next Initial Descriptor Register	0x00000000
0x31046384	DMA29_ADDRSTART	DMA29 Start Address of Current Buffer Register	0x00000000
0x31046388	DMA29_CFG	DMA29 Configuration Register	0x00000000
0x3104638C	DMA29_XCNT	DMA29 Inner Loop Count Start Value Register	0x00000000
0x31046390	DMA29_XMOD	DMA29 Inner Loop Address Increment Register	0x00000000
0x31046394	DMA29_YCNT	DMA29 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31046398	DMA29_YMOD	DMA29 Outer Loop Address Increment (2D only) Register	0x00000000
0x310463A4	DMA29_DSCPTR_CUR	DMA29 Current Descriptor Pointer Register	0x00000000
0x310463A8	DMA29_DSCPTR_PRV	DMA29 Previous Initial Descriptor Pointer Register	0x00000000
0x310463AC	DMA29_ADDR_CUR	DMA29 Current Address Register	0x00000000
0x310463B0	DMA29_STAT	DMA29 Status Register	0x00006000

Table A-44: ADSP-SC58x DMA29 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310463B4	DMA29_XCNT_CUR	DMA29 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310463B8	DMA29_YCNT_CUR	DMA29 Current Row Count (2D only) Register	0x00000000

Table A-45: ADSP-SC58x DMA3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022180	DMA3_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000
0x31022184	DMA3_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x31022188	DMA3_CFG	DMA3 Configuration Register	0x00000000
0x3102218C	DMA3_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x31022190	DMA3_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x31022194	DMA3_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022198	DMA3_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x310221A4	DMA3_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000
0x310221A8	DMA3_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x310221AC	DMA3_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x310221B0	DMA3_STAT	DMA3 Status Register	0x00006000
0x310221B4	DMA3_XCNT_CUR	DMA3 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310221B8	DMA3_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000

Table A-46: ADSP-SC58x DMA30 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFF000	DMA30_DSCPTR_NXT	DMA30 Pointer to Next Initial Descriptor Register	0x00000000
0x30FFF004	DMA30_ADDRSTART	DMA30 Start Address of Current Buffer Register	0x00000000
0x30FFF008	DMA30_CFG	DMA30 Configuration Register	0x00000000
0x30FFF00C	DMA30_XCNT	DMA30 Inner Loop Count Start Value Register	0x00000000
0x30FFF010	DMA30_XMOD	DMA30 Inner Loop Address Increment Register	0x00000000
0x30FFF014	DMA30_YCNT	DMA30 Outer Loop Count Start Value (2D only) Register	0x00000000

Table A-46: ADSP-SC58x DMA30 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFF018	DMA30_YMOD	DMA30 Outer Loop Address Increment (2D only) Register	0x00000000
0x30FFF024	DMA30_DSCPTR_CUR	DMA30 Current Descriptor Pointer Register	0x00000000
0x30FFF028	DMA30_DSCPTR_PRV	DMA30 Previous Initial Descriptor Pointer Register	0x00000000
0x30FFF02C	DMA30_ADDR_CUR	DMA30 Current Address Register	0x00000000
0x30FFF030	DMA30_STAT	DMA30 Status Register	0x00006000
0x30FFF034	DMA30_XCNT_CUR	DMA30 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x30FFF038	DMA30_YCNT_CUR	DMA30 Current Row Count (2D only) Register	0x00000000

Table A-47: ADSP-SC58x DMA31 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102A100	DMA31_DSCPTR_NXT	DMA31 Pointer to Next Initial Descriptor Register	0x00000000
0x3102A104	DMA31_ADDRSTART	DMA31 Start Address of Current Buffer Register	0x00000000
0x3102A108	DMA31_CFG	DMA31 Configuration Register	0x00000000
0x3102A10C	DMA31_XCNT	DMA31 Inner Loop Count Start Value Register	0x00000000
0x3102A110	DMA31_XMOD	DMA31 Inner Loop Address Increment Register	0x00000000
0x3102A114	DMA31_YCNT	DMA31 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102A118	DMA31_YMOD	DMA31 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102A124	DMA31_DSCPTR_CUR	DMA31 Current Descriptor Pointer Register	0x00000000
0x3102A128	DMA31_DSCPTR_PRV	DMA31 Previous Initial Descriptor Pointer Register	0x00000000
0x3102A12C	DMA31_ADDR_CUR	DMA31 Current Address Register	0x00000000
0x3102A130	DMA31_STAT	DMA31 Status Register	0x00006000
0x3102A134	DMA31_XCNT_CUR	DMA31 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102A138	DMA31_YCNT_CUR	DMA31 Current Row Count (2D only) Register	0x00000000

Table A-48: ADSP-SC58x DMA32 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102A000	DMA32_DSCPTR_NXT	DMA32 Pointer to Next Initial Descriptor Register	0x00000000

Table A-48: ADSP-SC58x DMA32 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102A004	DMA32_ADDRSTART	DMA32 Start Address of Current Buffer Register	0x00000000
0x3102A008	DMA32_CFG	DMA32 Configuration Register	0x00000000
0x3102A00C	DMA32_XCNT	DMA32 Inner Loop Count Start Value Register	0x00000000
0x3102A010	DMA32_XMOD	DMA32 Inner Loop Address Increment Register	0x00000000
0x3102A014	DMA32_YCNT	DMA32 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102A018	DMA32_YMOD	DMA32 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102A024	DMA32_DSCPTR_CUR	DMA32 Current Descriptor Pointer Register	0x00000000
0x3102A028	DMA32_DSCPTR_PRV	DMA32 Previous Initial Descriptor Pointer Register	0x00000000
0x3102A02C	DMA32_ADDR_CUR	DMA32 Current Address Register	0x00000000
0x3102A030	DMA32_STAT	DMA32 Status Register	0x00006000
0x3102A034	DMA32_XCNT_CUR	DMA32 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102A038	DMA32_YCNT_CUR	DMA32 Current Row Count (2D only) Register	0x00000000

Table A-49: ADSP-SC58x DMA33 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3102A080	DMA33_DSCPTR_NXT	DMA33 Pointer to Next Initial Descriptor Register	0x00000000
0x3102A084	DMA33_ADDRSTART	DMA33 Start Address of Current Buffer Register	0x00000000
0x3102A088	DMA33_CFG	DMA33 Configuration Register	0x00000000
0x3102A08C	DMA33_XCNT	DMA33 Inner Loop Count Start Value Register	0x00000000
0x3102A090	DMA33_XMOD	DMA33 Inner Loop Address Increment Register	0x00000000
0x3102A094	DMA33_YCNT	DMA33 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3102A098	DMA33_YMOD	DMA33 Outer Loop Address Increment (2D only) Register	0x00000000
0x3102A0A4	DMA33_DSCPTR_CUR	DMA33 Current Descriptor Pointer Register	0x00000000
0x3102A0A8	DMA33_DSCPTR_PRV	DMA33 Previous Initial Descriptor Pointer Register	0x00000000
0x3102A0AC	DMA33_ADDR_CUR	DMA33 Current Address Register	0x00000000
0x3102A0B0	DMA33_STAT	DMA33 Status Register	0x00006000

Table A-49: ADSP-SC58x DMA33 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3102A0B4	DMA33_XCNT_CUR	DMA33 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3102A0B8	DMA33_YCNT_CUR	DMA33 Current Row Count (2D only) Register	0x00000000

Table A-50: ADSP-SC58x DMA34 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026180	DMA34_DSCPTR_NXT	DMA34 Pointer to Next Initial Descriptor Register	0x00000000
0x31026184	DMA34_ADDRSTART	DMA34 Start Address of Current Buffer Register	0x00000000
0x31026188	DMA34_CFG	DMA34 Configuration Register	0x00000000
0x3102618C	DMA34_XCNT	DMA34 Inner Loop Count Start Value Register	0x00000000
0x31026190	DMA34_XMOD	DMA34 Inner Loop Address Increment Register	0x00000000
0x31026194	DMA34_YCNT	DMA34 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026198	DMA34_YMOD	DMA34 Outer Loop Address Increment (2D only) Register	0x00000000
0x310261A4	DMA34_DSCPTR_CUR	DMA34 Current Descriptor Pointer Register	0x00000000
0x310261A8	DMA34_DSCPTR_PRV	DMA34 Previous Initial Descriptor Pointer Register	0x00000000
0x310261AC	DMA34_ADDR_CUR	DMA34 Current Address Register	0x00000000
0x310261B0	DMA34_STAT	DMA34 Status Register	0x00006000
0x310261B4	DMA34_XCNT_CUR	DMA34 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310261B8	DMA34_YCNT_CUR	DMA34 Current Row Count (2D only) Register	0x00000000

Table A-51: ADSP-SC58x DMA35 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026100	DMA35_DSCPTR_NXT	DMA35 Pointer to Next Initial Descriptor Register	0x00000000
0x31026104	DMA35_ADDRSTART	DMA35 Start Address of Current Buffer Register	0x00000000
0x31026108	DMA35_CFG	DMA35 Configuration Register	0x00000000
0x3102610C	DMA35_XCNT	DMA35 Inner Loop Count Start Value Register	0x00000000
0x31026110	DMA35_XMOD	DMA35 Inner Loop Address Increment Register	0x00000000
0x31026114	DMA35_YCNT	DMA35 Outer Loop Count Start Value (2D only) Register	0x00000000

Table A-51: ADSP-SC58x DMA35 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026118	DMA35_YMOD	DMA35 Outer Loop Address Increment (2D only) Register	0x00000000
0x31026124	DMA35_DSCPTR_CUR	DMA35 Current Descriptor Pointer Register	0x00000000
0x31026128	DMA35_DSCPTR_PRV	DMA35 Previous Initial Descriptor Pointer Register	0x00000000
0x3102612C	DMA35_ADDR_CUR	DMA35 Current Address Register	0x00000000
0x31026130	DMA35_STAT	DMA35 Status Register	0x00006000
0x31026134	DMA35_XCNT_CUR	DMA35 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026138	DMA35_YCNT_CUR	DMA35 Current Row Count (2D only) Register	0x00000000

Table A-52: ADSP-SC58x DMA36 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFF080	DMA36_DSCPTR_NXT	DMA36 Pointer to Next Initial Descriptor Register	0x00000000
0x30FFF084	DMA36_ADDRSTART	DMA36 Start Address of Current Buffer Register	0x00000000
0x30FFF088	DMA36_CFG	DMA36 Configuration Register	0x00000000
0x30FFF08C	DMA36_XCNT	DMA36 Inner Loop Count Start Value Register	0x00000000
0x30FFF090	DMA36_XMOD	DMA36 Inner Loop Address Increment Register	0x00000000
0x30FFF094	DMA36_YCNT	DMA36 Outer Loop Count Start Value (2D only) Register	0x00000000
0x30FFF098	DMA36_YMOD	DMA36 Outer Loop Address Increment (2D only) Register	0x00000000
0x30FFF0A4	DMA36_DSCPTR_CUR	DMA36 Current Descriptor Pointer Register	0x00000000
0x30FFF0A8	DMA36_DSCPTR_PRV	DMA36 Previous Initial Descriptor Pointer Register	0x00000000
0x30FFF0AC	DMA36_ADDR_CUR	DMA36 Current Address Register	0x00000000
0x30FFF0B0	DMA36_STAT	DMA36 Status Register	0x00006000
0x30FFF0B4	DMA36_XCNT_CUR	DMA36 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x30FFF0B8	DMA36_YCNT_CUR	DMA36 Current Row Count (2D only) Register	0x00000000

Table A-53: ADSP-SC58x DMA37 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026280	DMA37_DSCPTR_NXT	DMA37 Pointer to Next Initial Descriptor Register	0x00000000

Table A-53: ADSP-SC58x DMA37 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026284	DMA37_ADDRSTART	DMA37 Start Address of Current Buffer Register	0x00000000
0x31026288	DMA37_CFG	DMA37 Configuration Register	0x00000000
0x3102628C	DMA37_XCNT	DMA37 Inner Loop Count Start Value Register	0x00000000
0x31026290	DMA37_XMOD	DMA37 Inner Loop Address Increment Register	0x00000000
0x31026294	DMA37_YCNT	DMA37 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026298	DMA37_YMOD	DMA37 Outer Loop Address Increment (2D only) Register	0x00000000
0x310262A4	DMA37_DSCPTR_CUR	DMA37 Current Descriptor Pointer Register	0x00000000
0x310262A8	DMA37_DSCPTR_PRV	DMA37 Previous Initial Descriptor Pointer Register	0x00000000
0x310262AC	DMA37_ADDR_CUR	DMA37 Current Address Register	0x00000000
0x310262B0	DMA37_STAT	DMA37 Status Register	0x00006000
0x310262B4	DMA37_XCNT_CUR	DMA37 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310262B8	DMA37_YCNT_CUR	DMA37 Current Row Count (2D only) Register	0x00000000

Table A-54: ADSP-SC58x DMA38 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31026200	DMA38_DSCPTR_NXT	DMA38 Pointer to Next Initial Descriptor Register	0x00000000
0x31026204	DMA38_ADDRSTART	DMA38 Start Address of Current Buffer Register	0x00000000
0x31026208	DMA38_CFG	DMA38 Configuration Register	0x00000000
0x3102620C	DMA38_XCNT	DMA38 Inner Loop Count Start Value Register	0x00000000
0x31026210	DMA38_XMOD	DMA38 Inner Loop Address Increment Register	0x00000000
0x31026214	DMA38_YCNT	DMA38 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31026218	DMA38_YMOD	DMA38 Outer Loop Address Increment (2D only) Register	0x00000000
0x31026224	DMA38_DSCPTR_CUR	DMA38 Current Descriptor Pointer Register	0x00000000
0x31026228	DMA38_DSCPTR_PRV	DMA38 Previous Initial Descriptor Pointer Register	0x00000000
0x3102622C	DMA38_ADDR_CUR	DMA38 Current Address Register	0x00000000
0x31026230	DMA38_STAT	DMA38 Status Register	0x00006000

Table A-54: ADSP-SC58x DMA38 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31026234	DMA38_XCNT_CUR	DMA38 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31026238	DMA38_YCNT_CUR	DMA38 Current Row Count (2D only) Register	0x00000000

Table A-55: ADSP-SC58x DMA39 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109A000	DMA39_DSCPTR_NXT	DMA39 Pointer to Next Initial Descriptor Register	0x00000000
0x3109A004	DMA39_ADDRSTART	DMA39 Start Address of Current Buffer Register	0x00000000
0x3109A008	DMA39_CFG	DMA39 Configuration Register	0x00000220
0x3109A00C	DMA39_XCNT	DMA39 Inner Loop Count Start Value Register	0x00000000
0x3109A010	DMA39_XMOD	DMA39 Inner Loop Address Increment Register	0x00000000
0x3109A014	DMA39_YCNT	DMA39 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109A018	DMA39_YMOD	DMA39 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109A024	DMA39_DSCPTR_CUR	DMA39 Current Descriptor Pointer Register	0x00000000
0x3109A028	DMA39_DSCPTR_PRV	DMA39 Previous Initial Descriptor Pointer Register	0x00000000
0x3109A02C	DMA39_ADDR_CUR	DMA39 Current Address Register	0x00000000
0x3109A030	DMA39_STAT	DMA39 Status Register	0x00006000
0x3109A034	DMA39_XCNT_CUR	DMA39 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109A038	DMA39_YCNT_CUR	DMA39 Current Row Count (2D only) Register	0x00000000
0x3109A040	DMA39_BWLCNT	DMA39 Bandwidth Limit Count Register	0x00000000
0x3109A044	DMA39_BWLCNT_CUR	DMA39 Bandwidth Limit Count Current Register	0x00000000
0x3109A048	DMA39_BWMCNT	DMA39 Bandwidth Monitor Count Register	0x00000000
0x3109A04C	DMA39_BWMCNT_CUR	DMA39 Bandwidth Monitor Count Current Register	0x00000000

Table A-56: ADSP-SC58x DMA4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022200	DMA4_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x31022204	DMA4_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000

Table A-56: ADSP-SC58x DMA4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31022208	DMA4_CFG	DMA4 Configuration Register	0x00000000
0x3102220C	DMA4_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x31022210	DMA4_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x31022214	DMA4_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022218	DMA4_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022224	DMA4_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x31022228	DMA4_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x3102222C	DMA4_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x31022230	DMA4_STAT	DMA4 Status Register	0x00006000
0x31022234	DMA4_XCNT_CUR	DMA4 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022238	DMA4_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000

Table A-57: ADSP-SC58x DMA40 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109A080	DMA40_DSCPTR_NXT	DMA40 Pointer to Next Initial Descriptor Register	0x00000000
0x3109A084	DMA40_ADDRSTART	DMA40 Start Address of Current Buffer Register	0x00000000
0x3109A088	DMA40_CFG	DMA40 Configuration Register	0x00000222
0x3109A08C	DMA40_XCNT	DMA40 Inner Loop Count Start Value Register	0x00000000
0x3109A090	DMA40_XMOD	DMA40 Inner Loop Address Increment Register	0x00000000
0x3109A094	DMA40_YCNT	DMA40 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109A098	DMA40_YMOD	DMA40 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109A0A4	DMA40_DSCPTR_CUR	DMA40 Current Descriptor Pointer Register	0x00000000
0x3109A0A8	DMA40_DSCPTR_PRV	DMA40 Previous Initial Descriptor Pointer Register	0x00000000
0x3109A0AC	DMA40_ADDR_CUR	DMA40 Current Address Register	0x00000000
0x3109A0B0	DMA40_STAT	DMA40 Status Register	0x00006000
0x3109A0B4	DMA40_XCNT_CUR	DMA40 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109A0B8	DMA40_YCNT_CUR	DMA40 Current Row Count (2D only) Register	0x00000000
0x3109A0C0	DMA40_BWLCNT	DMA40 Bandwidth Limit Count Register	0x00000000

Table A-57: ADSP-SC58x DMA40 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109A0C4	DMA40_BWLCNT_CUR	DMA40 Bandwidth Limit Count Current Register	0x00000000
0x3109A0C8	DMA40_BWMCNT	DMA40 Bandwidth Monitor Count Register	0x00000000
0x3109A0CC	DMA40_BWMCNT_CUR	DMA40 Bandwidth Monitor Count Current Register	0x00000000

Table A-58: ADSP-SC58x DMA41 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C200	DMA41_DSCPTR_NXT	DMA41 Pointer to Next Initial Descriptor Register	0x00000000
0x3109C204	DMA41_ADDRSTART	DMA41 Start Address of Current Buffer Register	0x00000000
0x3109C208	DMA41_CFG	DMA41 Configuration Register	0x00000220
0x3109C20C	DMA41_XCNT	DMA41 Inner Loop Count Start Value Register	0x00000000
0x3109C210	DMA41_XMOD	DMA41 Inner Loop Address Increment Register	0x00000000
0x3109C214	DMA41_YCNT	DMA41 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109C218	DMA41_YMOD	DMA41 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109C224	DMA41_DSCPTR_CUR	DMA41 Current Descriptor Pointer Register	0x00000000
0x3109C228	DMA41_DSCPTR_PRV	DMA41 Previous Initial Descriptor Pointer Register	0x00000000
0x3109C22C	DMA41_ADDR_CUR	DMA41 Current Address Register	0x00000000
0x3109C230	DMA41_STAT	DMA41 Status Register	0x00006000
0x3109C234	DMA41_XCNT_CUR	DMA41 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109C238	DMA41_YCNT_CUR	DMA41 Current Row Count (2D only) Register	0x00000000
0x3109C240	DMA41_BWLCNT	DMA41 Bandwidth Limit Count Register	0x00000000
0x3109C244	DMA41_BWLCNT_CUR	DMA41 Bandwidth Limit Count Current Register	0x00000000
0x3109C248	DMA41_BWMCNT	DMA41 Bandwidth Monitor Count Register	0x00000000
0x3109C24C	DMA41_BWMCNT_CUR	DMA41 Bandwidth Monitor Count Current Register	0x00000000

Table A-59: ADSP-SC58x DMA42 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C400	DMA42_DSCPTR_NXT	DMA42 Pointer to Next Initial Descriptor Register	0x00000000
0x3109C404	DMA42_ADDRSTART	DMA42 Start Address of Current Buffer Register	0x00000000

Table A-59: ADSP-SC58x DMA42 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C408	DMA42_CFG	DMA42 Configuration Register	0x00000222
0x3109C40C	DMA42_XCNT	DMA42 Inner Loop Count Start Value Register	0x00000000
0x3109C410	DMA42_XMOD	DMA42 Inner Loop Address Increment Register	0x00000000
0x3109C414	DMA42_YCNT	DMA42 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109C418	DMA42_YMOD	DMA42 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109C424	DMA42_DSCPTR_CUR	DMA42 Current Descriptor Pointer Register	0x00000000
0x3109C428	DMA42_DSCPTR_PRV	DMA42 Previous Initial Descriptor Pointer Register	0x00000000
0x3109C42C	DMA42_ADDR_CUR	DMA42 Current Address Register	0x00000000
0x3109C430	DMA42_STAT	DMA42 Status Register	0x00006000
0x3109C434	DMA42_XCNT_CUR	DMA42 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109C438	DMA42_YCNT_CUR	DMA42 Current Row Count (2D only) Register	0x00000000
0x3109C440	DMA42_BWLCNT	DMA42 Bandwidth Limit Count Register	0x00000000
0x3109C444	DMA42_BWLCNT_CUR	DMA42 Bandwidth Limit Count Current Register	0x00000000
0x3109C448	DMA42_BWMCNT	DMA42 Bandwidth Monitor Count Register	0x00000000
0x3109C44C	DMA42_BWMCNT_CUR	DMA42 Bandwidth Monitor Count Current Register	0x00000000

Table A-60: ADSP-SC58x DMA43 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109B000	DMA43_DSCPTR_NXT	DMA43 Pointer to Next Initial Descriptor Register	0x00000000
0x3109B004	DMA43_ADDRSTART	DMA43 Start Address of Current Buffer Register	0x00000000
0x3109B008	DMA43_CFG	DMA43 Configuration Register	0x00000220
0x3109B00C	DMA43_XCNT	DMA43 Inner Loop Count Start Value Register	0x00000000
0x3109B010	DMA43_XMOD	DMA43 Inner Loop Address Increment Register	0x00000000
0x3109B014	DMA43_YCNT	DMA43 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109B018	DMA43_YMOD	DMA43 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109B024	DMA43_DSCPTR_CUR	DMA43 Current Descriptor Pointer Register	0x00000000
0x3109B028	DMA43_DSCPTR_PRV	DMA43 Previous Initial Descriptor Pointer Register	0x00000000

Table A-60: ADSP-SC58x DMA43 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109B02C	DMA43_ADDR_CUR	DMA43 Current Address Register	0x00000000
0x3109B030	DMA43_STAT	DMA43 Status Register	0x00006000
0x3109B034	DMA43_XCNT_CUR	DMA43 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109B038	DMA43_YCNT_CUR	DMA43 Current Row Count (2D only) Register	0x00000000
0x3109B040	DMA43_BWLCNT	DMA43 Bandwidth Limit Count Register	0x00000000
0x3109B044	DMA43_BWLCNT_CUR	DMA43 Bandwidth Limit Count Current Register	0x00000000
0x3109B048	DMA43_BWMCNT	DMA43 Bandwidth Monitor Count Register	0x00000000
0x3109B04C	DMA43_BWMCNT_CUR	DMA43 Bandwidth Monitor Count Current Register	0x00000000

Table A-61: ADSP-SC58x DMA44 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109B080	DMA44_DSCPTR_NXT	DMA44 Pointer to Next Initial Descriptor Register	0x00000000
0x3109B084	DMA44_ADDRSTART	DMA44 Start Address of Current Buffer Register	0x00000000
0x3109B088	DMA44_CFG	DMA44 Configuration Register	0x00000222
0x3109B08C	DMA44_XCNT	DMA44 Inner Loop Count Start Value Register	0x00000000
0x3109B090	DMA44_XMOD	DMA44 Inner Loop Address Increment Register	0x00000000
0x3109B094	DMA44_YCNT	DMA44 Outer Loop Count Start Value (2D only) Register	0x00000000
0x3109B098	DMA44_YMOD	DMA44 Outer Loop Address Increment (2D only) Register	0x00000000
0x3109B0A4	DMA44_DSCPTR_CUR	DMA44 Current Descriptor Pointer Register	0x00000000
0x3109B0A8	DMA44_DSCPTR_PRV	DMA44 Previous Initial Descriptor Pointer Register	0x00000000
0x3109B0AC	DMA44_ADDR_CUR	DMA44 Current Address Register	0x00000000
0x3109B0B0	DMA44_STAT	DMA44 Status Register	0x00006000
0x3109B0B4	DMA44_XCNT_CUR	DMA44 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x3109B0B8	DMA44_YCNT_CUR	DMA44 Current Row Count (2D only) Register	0x00000000
0x3109B0C0	DMA44_BWLCNT	DMA44 Bandwidth Limit Count Register	0x00000000
0x3109B0C4	DMA44_BWLCNT_CUR	DMA44 Bandwidth Limit Count Current Register	0x00000000
0x3109B0C8	DMA44_BWMCNT	DMA44 Bandwidth Monitor Count Register	0x00000000
0x3109B0CC	DMA44_BWMCNT_CUR	DMA44 Bandwidth Monitor Count Current Register	0x00000000

Table A-62: ADSP-SC58x DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022280	DMA5_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000
0x31022284	DMA5_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x31022288	DMA5_CFG	DMA5 Configuration Register	0x00000000
0x3102228C	DMA5_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x31022290	DMA5_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000
0x31022294	DMA5_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022298	DMA5_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x310222A4	DMA5_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x310222A8	DMA5_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x310222AC	DMA5_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x310222B0	DMA5_STAT	DMA5 Status Register	0x00006000
0x310222B4	DMA5_XCNT_CUR	DMA5 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310222B8	DMA5_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000

Table A-63: ADSP-SC58x DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022300	DMA6_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x31022304	DMA6_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x31022308	DMA6_CFG	DMA6 Configuration Register	0x00000000
0x3102230C	DMA6_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000
0x31022310	DMA6_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x31022314	DMA6_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022318	DMA6_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x31022324	DMA6_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x31022328	DMA6_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x3102232C	DMA6_ADDR_CUR	DMA6 Current Address Register	0x00000000
0x31022330	DMA6_STAT	DMA6 Status Register	0x00006000
0x31022334	DMA6_XCNT_CUR	DMA6 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31022338	DMA6_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000

Table A-64: ADSP-SC58x DMA7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31022380	DMA7_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x31022384	DMA7_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000
0x31022388	DMA7_CFG	DMA7 Configuration Register	0x00000000
0x3102238C	DMA7_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x31022390	DMA7_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x31022394	DMA7_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31022398	DMA7_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x310223A4	DMA7_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x310223A8	DMA7_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x310223AC	DMA7_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x310223B0	DMA7_STAT	DMA7 Status Register	0x00006000
0x310223B4	DMA7_XCNT_CUR	DMA7 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310223B8	DMA7_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000

Table A-65: ADSP-SC58x DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31028000	DMA8_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000
0x31028004	DMA8_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x31028008	DMA8_CFG	DMA8 Configuration Register	0x00000000
0x3102800C	DMA8_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x31028010	DMA8_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x31028014	DMA8_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31028018	DMA8_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x31028024	DMA8_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x31028028	DMA8_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x3102802C	DMA8_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x31028030	DMA8_STAT	DMA8 Status Register	0x00006000
0x31028034	DMA8_XCNT_CUR	DMA8 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x31028038	DMA8_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000

Table A-65: ADSP-SC58x DMA8 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31028040	DMA8_BWLCNT	DMA8 Bandwidth Limit Count Register	0x00000000
0x31028044	DMA8_BWLCNT_CUR	DMA8 Bandwidth Limit Count Current Register	0x00000000
0x31028048	DMA8_BWMCNT	DMA8 Bandwidth Monitor Count Register	0x00000000
0x3102804C	DMA8_BWMCNT_CUR	DMA8 Bandwidth Monitor Count Current Register	0x00000000

Table A-66: ADSP-SC58x DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31028080	DMA9_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x31028084	DMA9_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x31028088	DMA9_CFG	DMA9 Configuration Register	0x00000000
0x3102808C	DMA9_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x31028090	DMA9_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x31028094	DMA9_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x31028098	DMA9_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x310280A4	DMA9_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x310280A8	DMA9_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x310280AC	DMA9_ADDR_CUR	DMA9 Current Address Register	0x00000000
0x310280B0	DMA9_STAT	DMA9 Status Register	0x00006000
0x310280B4	DMA9_XCNT_CUR	DMA9 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x310280B8	DMA9_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000
0x310280C0	DMA9_BWLCNT	DMA9 Bandwidth Limit Count Register	0x00000000
0x310280C4	DMA9_BWLCNT_CUR	DMA9 Bandwidth Limit Count Current Register	0x00000000
0x310280C8	DMA9_BWMCNT	DMA9 Bandwidth Monitor Count Register	0x00000000
0x310280CC	DMA9_BWMCNT_CUR	DMA9 Bandwidth Monitor Count Current Register	0x00000000

Table A-67: ADSP-SC58x DMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31070004	DMC0_CTL	DMC0 Control Register	0x00000000
0x31070008	DMC0_STAT	DMC0 Status Register	0x00000001

Table A-67: ADSP-SC58x DMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3107000C	DMC0_EFFCTL	DMC0 Efficiency Control Register	0x00440000
0x31070010	DMC0_PRIO	DMC0 Priority ID Register 1	0x00000000
0x31070014	DMC0_PRIOMSK	DMC0 Priority ID Mask Register 1	0x00000000
0x31070018	DMC0_PRIO2	DMC0 Priority ID Register 2	0x00000000
0x3107001C	DMC0_PRIOMSK2	DMC0 Priority ID Mask Register 2	0x00000000
0x31070040	DMC0_CFG	DMC0 Configuration Register	0x00000000
0x31070044	DMC0_TR0	DMC0 Timing 0 Register	0x00000000
0x31070048	DMC0_TR1	DMC0 Timing 1 Register	0x00000000
0x3107004C	DMC0_TR2	DMC0 Timing 2 Register	0x00000000
0x3107005C	DMC0_MSK	DMC0 Mask (Mode Register Shadow) Register	0x00000000
0x31070060	DMC0_MR	DMC0 Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3)	0x00000000
0x31070064	DMC0_MR1	DMC0 Shadow MR1 Register (DDR3)	0x00000000
0x31070064	DMC0_EMR1	DMC0 Shadow EMR1 DDR2 Register	0x00000000
0x31070068	DMC0_EMR2	DMC0 Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)	0x00000000
0x31070068	DMC0_MR2	DMC0 Shadow MR2 Register (DDR3)	0x00000000
0x31070080	DMC0_DLLCTL	DMC0 DLL Control Register	0x0000054B
0x31070090	DMC0_DT_CALIB_ADDR	DMC0 Data Calibration Address Register	0x00000000
0x31070094	DMC0_DT_DATA_CALIB_DATA0	DMC0 Data Calibration Data 0 Register	0x00000000
0x31070098	DMC0_DT_DATA_CALIB_DATA1	DMC0 Data Calibration Data 1 Register	0x00000000
0x31070100	DMC0_RDDATABUFID1	DMC0 DMC Read Data Buffer ID Register 1	0x00000000
0x31070104	DMC0_RDDATABUFMSK1	DMC0 DMC Read Data Buffer Mask Register 1	0x00000000
0x31070108	DMC0_RDDATABUFID2	DMC0 DMC Read Data Buffer ID Register 2	0x00000000
0x3107010C	DMC0_RDDATABUFMSK2	DMC0 DMC Read Data Buffer Mask Register 2	0x00000000
0x310701C0	DMC0_CPHY_CTL	DMC0 Controller to PHY Interface Register	0x00000000

Table A-68: ADSP-SC58x DMC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31073004	DMC1_CTL	DMC1 Control Register	0x00000000

Table A-68: ADSP-SC58x DMC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31073008	DMC1_STAT	DMC1 Status Register	0x00000001
0x3107300C	DMC1_EFFCTL	DMC1 Efficiency Control Register	0x00440000
0x31073010	DMC1_PRIO	DMC1 Priority ID Register 1	0x00000000
0x31073014	DMC1_PRIOMSK	DMC1 Priority ID Mask Register 1	0x00000000
0x31073018	DMC1_PRIO2	DMC1 Priority ID Register 2	0x00000000
0x3107301C	DMC1_PRIOMSK2	DMC1 Priority ID Mask Register 2	0x00000000
0x31073040	DMC1_CFG	DMC1 Configuration Register	0x00000000
0x31073044	DMC1_TR0	DMC1 Timing 0 Register	0x00000000
0x31073048	DMC1_TR1	DMC1 Timing 1 Register	0x00000000
0x3107304C	DMC1_TR2	DMC1 Timing 2 Register	0x00000000
0x3107305C	DMC1_MSK	DMC1 Mask (Mode Register Shadow) Register	0x00000000
0x31073060	DMC1_MR	DMC1 Shadow MR Register (DDR2/LPDDR), Shadow MR0 Register (DDR3)	0x00000000
0x31073064	DMC1_MR1	DMC1 Shadow MR1 Register (DDR3)	0x00000000
0x31073064	DMC1_EMR1	DMC1 Shadow EMR1 DDR2 Register	0x00000000
0x31073068	DMC1_EMR2	DMC1 Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)	0x00000000
0x31073068	DMC1_MR2	DMC1 Shadow MR2 Register (DDR3)	0x00000000
0x31073080	DMC1_DLLCTL	DMC1 DLL Control Register	0x0000054B
0x31073090	DMC1_DT_CALIB_ADDR	DMC1 Data Calibration Address Register	0x00000000
0x31073094	DMC1_DT_DATA_CALIB_DATA0	DMC1 Data Calibration Data 0 Register	0x00000000
0x31073098	DMC1_DT_DATA_CALIB_DATA1	DMC1 Data Calibration Data 1 Register	0x00000000
0x31073100	DMC1_RDDATABUFID1	DMC1 DMC Read Data Buffer ID Register 1	0x00000000
0x31073104	DMC1_RDDATABUFMSK1	DMC1 DMC Read Data Buffer Mask Register 1	0x00000000
0x31073108	DMC1_RDDATABUFID2	DMC1 DMC Read Data Buffer ID Register 2	0x00000000
0x3107310C	DMC1_RDDATABUFMSK2	DMC1 DMC Read Data Buffer Mask Register 2	0x00000000
0x310731C0	DMC1_CPHY_CTL	DMC1 Controller to PHY Interface Register	0x00000000

Table A-69: ADSP-SC58x DMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31071000	DMC0_PHY_CTL0	DMC0 PHY Control 0 Register	0x00000000
0x31071004	DMC0_PHY_CTL1	DMC0 PHY Control 1 Register	0x00000000
0x31071008	DMC0_PHY_CTL2	DMC0 PHY Control 2 Register	0x00000000
0x3107100C	DMC0_PHY_CTL3	DMC0 PHY Control 3 Register	0x00000000
0x31071010	DMC0_PHY_CTL4	DMC0 PHY Control 4 Register	0x00000000
0x31071034	DMC0_CAL_PADCTL0	DMC0 Calibration PAD Control 0 Register	0xE0000000
0x3107103C	DMC0_CAL_PADCTL2	DMC0 Calibration PAD Control 2 Register	0x00000000

Table A-70: ADSP-SC58x DMC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31074000	DMC1_PHY_CTL0	DMC1 PHY Control 0 Register	0x00000000
0x31074004	DMC1_PHY_CTL1	DMC1 PHY Control 1 Register	0x00000000
0x31074008	DMC1_PHY_CTL2	DMC1 PHY Control 2 Register	0x00000000
0x3107400C	DMC1_PHY_CTL3	DMC1 PHY Control 3 Register	0x00000000
0x31074010	DMC1_PHY_CTL4	DMC1 PHY Control 4 Register	0x00000000
0x31074034	DMC1_CAL_PADCTL0	DMC1 Calibration PAD Control 0 Register	0xE0000000
0x3107403C	DMC1_CAL_PADCTL2	DMC1 Calibration PAD Control 2 Register	0x00000000

Table A-71: ADSP-SC58x DPM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31090000	DPM0_CTL	DPM0 Control Register	0x00000000
0x31090004	DPM0_STAT	DPM0 Status Register	0x00000001
0x31090070	DPM0_PER_DIS0	DPM0 Peripherals Disable Register 0	0x00000000
0x31090074	DPM0_PER_DIS1	DPM0 Peripherals Disable Register 1	0x00000000
0x31090084	DPM0_REVID	DPM0 Revision ID	0x00000020

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C000	EMAC0_MACCFG	EMAC0 MAC Configuration Register	0x00008000
0x3100C004	EMAC0_MACFRMFILT	EMAC0 MAC Rx Frame Filter Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C008	EMAC0_HASHTBL_HI	EMAC0 Hash Table High Register	0x00000000
0x3100C00C	EMAC0_HASHTBL_LO	EMAC0 Hash Table Low Register	0x00000000
0x3100C010	EMAC0_SMI_ADDR	EMAC0 SMI Address Register	0x00000000
0x3100C014	EMAC0_SMI_DATA	EMAC0 SMI Data Register	0x00000000
0x3100C018	EMAC0_FLOWCTL	EMAC0 FLow Control Register	0x00000000
0x3100C01C	EMAC0_VLANTAG	EMAC0 VLAN Tag Register	0x00000000
0x3100C024	EMAC0_DBG	EMAC0 Debug Register	0x00000000
0x3100C030	EMAC0_LPI_CTLSTAT	EMAC0 Low Power Idle Control and Status Register	0x00000000
0x3100C034	EMAC0_LPI_TMRSCTL	EMAC0 Low Power Idle Timeout Register	0x00000000
0x3100C038	EMAC0_ISTAT	EMAC0 Interrupt Status Register	0x00000000
0x3100C03C	EMAC0_IMSK	EMAC0 Interrupt Mask Register	0x00000000
0x3100C040	EMAC0_ADDR0_HI	EMAC0 MAC Address 0 High Register	0x8000FFFF
0x3100C044	EMAC0_ADDR0_LO	EMAC0 MAC Address 0 Low Register	0xFFFFFFFF
0x3100C048	EMAC0_ADDR1_HI	EMAC0 MAC Address 1 High Register	0x00000000
0x3100C04C	EMAC0_ADDR1_LO	EMAC0 MAC Address 1 Low Register	0x00000000
0x3100C0D8	EMAC0_GIGE_CTLSTAT	EMAC0 RGMII Control and Status Register	0x00000000
0x3100C0DC	EMAC0_WDOG_TIMOUT	EMAC0 Watchdog Timeout Register	0x00000000
0x3100C100	EMAC0_MMC_CTL	EMAC0 MMC Control Register	0x00000000
0x3100C104	EMAC0_MMC_RXINT	EMAC0 MMC Rx Interrupt Register	0x00000000
0x3100C108	EMAC0_MMC_TXINT	EMAC0 MMC Tx Interrupt Register	0x00000000
0x3100C10C	EMAC0_MMC_RXIMSK	EMAC0 MMC Rx Interrupt Mask Register	0x00000000
0x3100C110	EMAC0_MMC_TXIMSK	EMAC0 MMC TX Interrupt Mask Register	0x00000000
0x3100C114	EMAC0_TXOCTCNT_GB	EMAC0 Tx OCT Count (Good/Bad) Register	0x00000000
0x3100C118	EMAC0_TXFRMCNT_GB	EMAC0 Tx Frame Count (Good/Bad) Register	0x00000000
0x3100C11C	EMAC0_TXBCASTFRM_G	EMAC0 Tx Broadcast Frames (Good) Register	0x00000000
0x3100C120	EMAC0_TXMCASTFRM_G	EMAC0 Tx Multicast Frames (Good) Register	0x00000000
0x3100C124	EMAC0_TX64_GB	EMAC0 Tx 64-Byte Frames (Good/Bad) Register	0x00000000
0x3100C128	EMAC0_TX65TO127_GB	EMAC0 Tx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x3100C12C	EMAC0_TX128TO255_GB	EMAC0 Tx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C130	EMAC0_TX256TO511_GB	EMAC0 Tx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000
0x3100C134	EMAC0_TX512TO1023_GB	EMAC0 Tx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x3100C138	EMAC0_TX1024TOMAX_GB	EMAC0 Tx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000
0x3100C13C	EMAC0_TXUCASTFRM_GB	EMAC0 Tx Unicast Frames (Good/Bad) Register	0x00000000
0x3100C140	EMAC0_TXMCASTFRM_GB	EMAC0 Tx Multicast Frames (Good/Bad) Register	0x00000000
0x3100C144	EMAC0_TXBCASTFRM_GB	EMAC0 Tx Broadcast Frames (Good/Bad) Register	0x00000000
0x3100C148	EMAC0_TXUNDR_ERR	EMAC0 Tx Underflow Error Register	0x00000000
0x3100C14C	EMAC0_TXSNGCOL_G	EMAC0 Tx Single Collision (Good) Register	0x00000000
0x3100C150	EMAC0_TXMULTCOL_G	EMAC0 Tx Multiple Collision (Good) Register	0x00000000
0x3100C154	EMAC0_TXDEFERRED	EMAC0 Tx Deferred Register	0x00000000
0x3100C158	EMAC0_TXLATECOL	EMAC0 Tx Late Collision Register	0x00000000
0x3100C15C	EMAC0_TXEXCESSCOL	EMAC0 Tx Excess Collision Register	0x00000000
0x3100C160	EMAC0_TXCARR_ERR	EMAC0 Tx Carrier Error Register	0x00000000
0x3100C164	EMAC0_TXOCTCNT_G	EMAC0 Tx Octet Count (Good) Register	0x00000000
0x3100C168	EMAC0_TXFRMCNT_G	EMAC0 Tx Frame Count (Good) Register	0x00000000
0x3100C16C	EMAC0_TXEXCESSDEF	EMAC0 Tx Excess Deferral Register	0x00000000
0x3100C170	EMAC0_TXPAUSEFRM	EMAC0 Tx Pause Frame Register	0x00000000
0x3100C174	EMAC0_TXVLANFRM_G	EMAC0 Tx VLAN Frames (Good) Register	0x00000000
0x3100C178	EMAC0_TXOVRSIZE_G	EMAC0 Number of Tx Frames (Good) greater than maxsize	0x00000000
0x3100C180	EMAC0_RXFRMCNT_GB	EMAC0 Rx Frame Count (Good/Bad) Register	0x00000000
0x3100C184	EMAC0_RXOCTCNT_GB	EMAC0 Rx Octet Count (Good/Bad) Register	0x00000000
0x3100C188	EMAC0_RXOCTCNT_G	EMAC0 Rx Octet Count (Good) Register	0x00000000
0x3100C18C	EMAC0_RXBCASTFRM_G	EMAC0 Rx Broadcast Frames (Good) Register	0x00000000
0x3100C190	EMAC0_RXMCASTFRM_G	EMAC0 Rx Multicast Frames (Good) Register	0x00000000
0x3100C194	EMAC0_RXCRC_ERR	EMAC0 Rx CRC Error Register	0x00000000
0x3100C198	EMAC0_RXALIGN_ERR	EMAC0 Rx alignment Error Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C19C	EMAC0_RXRUNT_ERR	EMAC0 Rx Runt Error Register	0x00000000
0x3100C1A0	EMAC0_RXJAB_ERR	EMAC0 Rx Jab Error Register	0x00000000
0x3100C1A4	EMAC0_RXUSIZE_G	EMAC0 Rx Undersize (Good) Register	0x00000000
0x3100C1A8	EMAC0_RXOSIZE_G	EMAC0 Rx Oversize (Good) Register	0x00000000
0x3100C1AC	EMAC0_RX64_GB	EMAC0 Rx 64-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1B0	EMAC0_RX65TO127_GB	EMAC0 Rx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1B4	EMAC0_RX128TO255_GB	EMAC0 Rx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1B8	EMAC0_RX256TO511_GB	EMAC0 Rx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1BC	EMAC0_RX512TO1023_GB	EMAC0 Rx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1C0	EMAC0_RX1024TOMAX_GB	EMAC0 Rx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000
0x3100C1C4	EMAC0_RXUCASTFRM_G	EMAC0 Rx Unicast Frames (Good) Register	0x00000000
0x3100C1C8	EMAC0_RXLEN_ERR	EMAC0 Rx Length Error Register	0x00000000
0x3100C1CC	EMAC0_RXOORTYPE	EMAC0 Rx Out Of Range Type Register	0x00000000
0x3100C1D0	EMAC0_RXPAUSEFRM	EMAC0 Rx Pause Frames Register	0x00000000
0x3100C1D4	EMAC0_RXFIFO_OVF	EMAC0 Rx FIFO Overflow Register	0x00000000
0x3100C1D8	EMAC0_RXVLANFRM_GB	EMAC0 Rx VLAN Frames (Good/Bad) Register	0x00000000
0x3100C1DC	EMAC0_RXWDOG_ERR	EMAC0 Rx Watch Dog Error Register	0x00000000
0x3100C1E0	EMAC0_RXRCV_ERR	EMAC0 Rx Error Frames Received Register	0x00000000
0x3100C1E4	EMAC0_RXCTLFRM_G	EMAC0 Rx Good Control Frames Register	0x00000000
0x3100C200	EMAC0_IPC_RXIMSK	EMAC0 MMC IPC Rx Interrupt Mask Register	0x00000000
0x3100C208	EMAC0_IPC_RXINT	EMAC0 MMC IPC Rx Interrupt Register	0x00000000
0x3100C210	EMAC0_RXIPV4_GD_FRM	EMAC0 Rx IPv4 Datagrams (Good) Register	0x00000000
0x3100C214	EMAC0_RXIPV4_HDR_ERR_FRM	EMAC0 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x3100C218	EMAC0_RXIPV4_NOPAY_FRM	EMAC0 Rx IPv4 Datagrams No Payload Frame Register	0x00000000
0x3100C21C	EMAC0_RXIPV4_FRAG_FRM	EMAC0 Rx IPv4 Datagrams Fragmented Frames Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C220	EMAC0_RXIPV4_UDSBL_FRM	EMAC0 Rx IPv4 UDP Disabled Frames Register	0x00000000
0x3100C224	EMAC0_RXIPV6_GD_FRM	EMAC0 Rx IPv6 Datagrams Good Frames Register	0x00000000
0x3100C228	EMAC0_RXIPV6_HDR_ERR_FRM	EMAC0 Rx IPv6 Datagrams Header Error Frames Register	0x00000000
0x3100C22C	EMAC0_RXIPV6_NOPAY_FRM	EMAC0 Rx IPv6 Datagrams No Payload Frames Register	0x00000000
0x3100C230	EMAC0_RXUDP_GD_FRM	EMAC0 Rx UDP Good Frames Register	0x00000000
0x3100C234	EMAC0_RXUDP_ERR_FRM	EMAC0 Rx UDP Error Frames Register	0x00000000
0x3100C238	EMAC0_RXTCP_GD_FRM	EMAC0 Rx TCP Good Frames Register	0x00000000
0x3100C23C	EMAC0_RXTCP_ERR_FRM	EMAC0 Rx TCP Error Frames Register	0x00000000
0x3100C240	EMAC0_RXICMP_GD_FRM	EMAC0 Rx ICMP Good Frames Register	0x00000000
0x3100C244	EMAC0_RXICMP_ERR_FRM	EMAC0 Rx ICMP Error Frames Register	0x00000000
0x3100C250	EMAC0_RXIPV4_GD_OCT	EMAC0 Rx IPv4 Datagrams Good Octets Register	0x00000000
0x3100C254	EMAC0_RXIPV4_HDR_ERR_OCT	EMAC0 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x3100C258	EMAC0_RXIPV4_NOPAY_OCT	EMAC0 Rx IPv4 Datagrams No Payload Octets Register	0x00000000
0x3100C25C	EMAC0_RXIPV4_FRAG_OCT	EMAC0 Rx IPv4 Datagrams Fragmented Octets Register	0x00000000
0x3100C260	EMAC0_RXIPV4_UDSBL_OCT	EMAC0 Rx IPv4 UDP Disabled Octets Register	0x00000000
0x3100C264	EMAC0_RXIPV6_GD_OCT	EMAC0 Rx IPv6 Good Octets Register	0x00000000
0x3100C268	EMAC0_RXIPV6_HDR_ERR_OCT	EMAC0 Rx IPv6 Header Errors Register	0x00000000
0x3100C26C	EMAC0_RXIPV6_NOPAY_OCT	EMAC0 Rx IPv6 No Payload Octets Register	0x00000000
0x3100C270	EMAC0_RXUDP_GD_OCT	EMAC0 Rx UDP Good Octets Register	0x00000000
0x3100C274	EMAC0_RXUDP_ERR_OCT	EMAC0 Rx UDP Error Octets Register	0x00000000
0x3100C278	EMAC0_RXTCP_GD_OCT	EMAC0 Rx TCP Good Octets Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C27C	EMAC0_RXTCP_ERR_OCT	EMAC0 Rx TCP Error Octets Register	0x00000000
0x3100C280	EMAC0_RXICMP_GD_OCT	EMAC0 Rx ICMP Good Octets Register	0x00000000
0x3100C284	EMAC0_RXICMP_ERR_OCT	EMAC0 Rx ICMP Error Octets Register	0x00000000
0x3100C400	EMAC0_L3L4_CTL	EMAC0 Layer3 and Layer4 Control Register	0x00000000
0x3100C404	EMAC0_L4_ADDR	EMAC0 Layer 4 Address Register	0x00000000
0x3100C410	EMAC0_L3_ADDR0	EMAC0 Layer 3 Address0 Register	0x00000000
0x3100C414	EMAC0_L3_ADDR1	EMAC0 Layer 3 Address1 Register	0x00000000
0x3100C418	EMAC0_L3_ADDR2	EMAC0 Layer 3 Address2 Register	0x00000000
0x3100C41C	EMAC0_L3_ADDR3	EMAC0 Layer 3 Address3 Register	0x00000000
0x3100C584	EMAC0_VLAN_INCL	EMAC0 VLAN Tag Inclusion or Replacement Register	0x00000000
0x3100C588	EMAC0_VLAN_HSHTBL	EMAC0 VLAN Hash Table Register	0x00000000
0x3100C700	EMAC0_TM_CTL	EMAC0 Time Stamp Control Register	0x00002000
0x3100C704	EMAC0_TM_SUBSEC	EMAC0 Time Stamp Sub Second Increment Register	0x00000000
0x3100C708	EMAC0_TM_SEC	EMAC0 Time Stamp Low Seconds Register	0x00000000
0x3100C70C	EMAC0_TM_NSEC	EMAC0 Time Stamp Nanoseconds Register	0x00000000
0x3100C710	EMAC0_TM_SECUPDT	EMAC0 Time Stamp Seconds Update Register	0x00000000
0x3100C714	EMAC0_TM_NSECUPDT	EMAC0 Time Stamp Nanoseconds Update Register	0x00000000
0x3100C718	EMAC0_TM_ADDEND	EMAC0 Time Stamp Addend Register	0x00000000
0x3100C71C	EMAC0_TM_PPS0TGTM	EMAC0 Time Stamp Target Time Seconds Register	0x00000000
0x3100C720	EMAC0_TM_PPS0NTGTM	EMAC0 Time Stamp Target Time Nanoseconds Register	0x00000000
0x3100C724	EMAC0_TM_HISEC	EMAC0 Time Stamp High Second Register	0x00000000
0x3100C728	EMAC0_TM_STMPSTAT	EMAC0 Time Stamp Status Register	0x00000000
0x3100C72C	EMAC0_TM_PPSCTL	EMAC0 PPS Control Register	0x00000000
0x3100C730	EMAC0_TM_AUXSTMP_NSEC	EMAC0 Time Stamp Auxiliary TS Nano Seconds Register	0x00000000
0x3100C734	EMAC0_TM_AUXSTMP_SEC	EMAC0 Time Stamp Auxiliary TM Seconds Register	0x00000000
0x3100C738	EMAC0_MAC_AVCTL	EMAC0 AV MAC Control Register	0x00000000
0x3100C760	EMAC0_TM_PPS0INTVL	EMAC0 Time Stamp PPS Interval Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100C764	EMAC0_TM_PPS0WIDTH	EMAC0 PPS Width Register	0x00000000
0x3100C780	EMAC0_TM_PPS1TGTM	EMAC0 PPS 1 Target Time Seconds Register	0x00000000
0x3100C784	EMAC0_TM_PPS1NTGTM	EMAC0 PPS 1 Target Time Nanoseconds Register	0x00000000
0x3100C788	EMAC0_TM_PPS1INTVL	EMAC0 PPS 1 Interval Register	0x00000000
0x3100C78C	EMAC0_TM_PPS1WIDTH	EMAC0 PPS 1 Width Register	0x00000000
0x3100C7A0	EMAC0_TM_PPS2TGTM	EMAC0 PPS 2 Target Time Seconds Register	0x00000000
0x3100C7A4	EMAC0_TM_PPS2NTGTM	EMAC0 PPS 2 Target Time Nanoseconds Register	0x00000000
0x3100C7A8	EMAC0_TM_PPS2INTVL	EMAC0 PPS 2 Interval Register	0x00000000
0x3100C7AC	EMAC0_TM_PPS2WIDTH	EMAC0 PPS 2 Width Register	0x00000000
0x3100C7C0	EMAC0_TM_PPS3TGTM	EMAC0 PPS 3 Target Time Seconds Register	0x00000000
0x3100C7C4	EMAC0_TM_PPS3NTGTM	EMAC0 PPS 3 Target Time Nanoseconds Register	0x00000000
0x3100C7C8	EMAC0_TM_PPS3INTVL	EMAC0 PPS 3 Interval Register	0x00000000
0x3100C7CC	EMAC0_TM_PPS3WIDTH	EMAC0 PPS 3 Width Register	0x00000000
0x3100D000	EMAC0_DMA0_BUSMODE	EMAC0 DMA Bus Mode Register	0x00020101
0x3100D004	EMAC0_DMA0_TXPOLL	EMAC0 DMA Tx Poll Demand Register	0x00000000
0x3100D008	EMAC0_DMA0_RXPOLL	EMAC0 DMA Rx Poll Demand register	0x00000000
0x3100D00C	EMAC0_DMA0_RXDSC_ADDR	EMAC0 DMA Rx Descriptor List Address Register	0x00000000
0x3100D010	EMAC0_DMA0_TXDSC_ADDR	EMAC0 DMA Tx Descriptor List Address Register	0x00000000
0x3100D014	EMAC0_DMA0_STAT	EMAC0 DMA Status Register	0x00000000
0x3100D018	EMAC0_DMA0_OPMODE	EMAC0 DMA Operation Mode Register	0x00000000
0x3100D01C	EMAC0_DMA0_IEN	EMAC0 DMA Interrupt Enable Register	0x00000000
0x3100D020	EMAC0_DMA0_MISS_FRM	EMAC0 DMA Missed Frame Register	0x00000000
0x3100D024	EMAC0_DMA0_RXIWDOG	EMAC0 DMA Rx Interrupt Watch Dog Register	0x00000000
0x3100D028	EMAC0_DMA0_BMMODE	EMAC0 DMA SCB Bus Mode Register	0xC0110001
0x3100D02C	EMAC0_DMA0_BMSTAT	EMAC0 DMA SCB Status Register	0x00000000
0x3100D048	EMAC0_DMA0_TXDSC_CUR	EMAC0 DMA Tx Descriptor Current Register	0x00000000
0x3100D04C	EMAC0_DMA0_RXDSC_CUR	EMAC0 DMA Rx Descriptor Current Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100D050	EMAC0_DMA0_TXBUF_CUR	EMAC0 DMA Tx Buffer Current Register	0x00000000
0x3100D054	EMAC0_DMA0_RXBUF_CUR	EMAC0 DMA Rx Buffer Current Register	0x00000000
0x3100D100	EMAC0_DMA1_BUSMODE	EMAC0 DMA Bus Mode Register	0x00020100
0x3100D104	EMAC0_DMA1_TXPOLL	EMAC0 DMA Tx Poll Demand Register	0x00000000
0x3100D108	EMAC0_DMA1_RXPOLL	EMAC0 DMA Rx Poll Demand Register	0x00000000
0x3100D10C	EMAC0_DMA1_RXDSC_ADDR	EMAC0 DMA Rx Descriptor List Address Register	0x00000000
0x3100D110	EMAC0_DMA1_TXDSC_ADDR	EMAC0 DMA Tx Descriptor List Address Register	0x00000000
0x3100D114	EMAC0_DMA1_STAT	EMAC0 DMA Status Register	0x00000000
0x3100D118	EMAC0_DMA1_OPMODE	EMAC0 DMA Operation Mode Register	0x00000000
0x3100D11C	EMAC0_DMA1_IEN	EMAC0 DMA Interrupt Enable Register	0x00000000
0x3100D120	EMAC0_DMA1_MISS_FRM	EMAC0 DMA Missed Frame Register	0x00000000
0x3100D124	EMAC0_DMA1_RXIWDOG	EMAC0 DMA Rx Interrupt Watch Dog Register	0x00000000
0x3100D130	EMAC0_DMA1_CHSFCS	EMAC0 Channel 1 Control Bits for Slot Function Register	0x00000000
0x3100D148	EMAC0_DMA1_TXDSC_CUR	EMAC0 DMA Tx Descriptor Current Register	0x00000000
0x3100D14C	EMAC0_DMA1_RXDSC_CUR	EMAC0 DMA Rx Descriptor Current Register	0x00000000
0x3100D150	EMAC0_DMA1_TXBUF_CUR	EMAC0 DMA Tx Buffer Current Register	0x00000000
0x3100D154	EMAC0_DMA1_RXBUF_CUR	EMAC0 DMA Rx Buffer Current Register	0x00000000
0x3100D160	EMAC0_DMA1_CHCBSCTL	EMAC0 Channel 1 Credit Shaping Control Register	0x00000000
0x3100D164	EMAC0_DMA1_CHCBSSTAT	EMAC0 Channel 1 Average Traffic Transmitted Register	0x00000000
0x3100D168	EMAC0_DMA1_CHISC	EMAC0 Channel 1 Idle Slope Credit Value Register	0x00000000
0x3100D16C	EMAC0_DMA1_CHSSC	EMAC0 Channel 1 Send Slope Credit Value Register	0x00000000
0x3100D170	EMAC0_DMA1_CHHIC	EMAC0 Channel 1 High Credit Value Register	0x00000000

Table A-72: ADSP-SC58x EMAC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100D174	EMAC0_DMA1_CHLOC	EMAC0 Channel 1 Low Credit Value Register	0x00000000
0x3100D200	EMAC0_DMA2_BUSMODE	EMAC0 DMA Bus Mode Register	0x00020101
0x3100D204	EMAC0_DMA2_TXPOLL	EMAC0 DMA Tx Poll Demand Register	0x00000000
0x3100D208	EMAC0_DMA2_RXPOLL	EMAC0 DMA Rx Poll Demand register	0x00000000
0x3100D20C	EMAC0_DMA2_RXDSC_ADDR	EMAC0 DMA Rx Descriptor List Address Register	0x00000000
0x3100D210	EMAC0_DMA2_TXDSC_ADDR	EMAC0 DMA Tx Descriptor List Address Register	0x00000000
0x3100D214	EMAC0_DMA2_STAT	EMAC0 DMA Status Register	0x00000000
0x3100D218	EMAC0_DMA2_OPMODE	EMAC0 DMA Operation Mode Register	0x00000000
0x3100D21C	EMAC0_DMA2_IEN	EMAC0 DMA Interrupt Enable Register	0x00000000
0x3100D220	EMAC0_DMA2_MISS_FRM	EMAC0 DMA Missed Frame Register	0x00000000
0x3100D224	EMAC0_DMA2_RXIWDOG	EMAC0 DMA Rx Interrupt Watch Dog Register	0x00000000
0x3100D230	EMAC0_DMA2_CHSFCS	EMAC0 Channel 2 Control Bits for Slot Function Register	0x00000000
0x3100D248	EMAC0_DMA2_TXDSC_CUR	EMAC0 DMA Tx Descriptor Current Register	0x00000000
0x3100D24C	EMAC0_DMA2_RXDSC_CUR	EMAC0 DMA Rx Descriptor Current Register	0x00000000
0x3100D250	EMAC0_DMA2_TXBUF_CUR	EMAC0 DMA Tx Buffer Current Register	0x00000000
0x3100D254	EMAC0_DMA2_RXBUF_CUR	EMAC0 DMA Rx Buffer Current Register	0x00000000
0x3100D260	EMAC0_DMA2_CHCBSCTL	EMAC0 Channel 2 Credit Shaping Control Register	0x00000000
0x3100D264	EMAC0_DMA2_CHCBSSTAT	EMAC0 Channel 2 Avg Traffic Transmitted Status Register	0x00000000
0x3100D268	EMAC0_DMA2_CHISC	EMAC0 Channel 2 Idle Slope Credit Value Register	0x00000000
0x3100D26C	EMAC0_DMA2_CHSSC	EMAC0 Channel 2 Send Slope Credit Value Register	0x00000000
0x3100D270	EMAC0_DMA2_CHHIC	EMAC0 Channel 2 High Credit Value Register	0x00000000
0x3100D274	EMAC0_DMA2_CHLOC	EMAC0 Channel 2 Low Credit Value Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100E000	EMAC1_MACCFG	EMAC1 MAC Configuration Register	0x00000000
0x3100E004	EMAC1_MACFRMFILT	EMAC1 MAC Rx Frame Filter Register	0x00000000
0x3100E008	EMAC1_HASHTBL_HI	EMAC1 Hash Table High Register	0x00000000
0x3100E00C	EMAC1_HASHTBL_LO	EMAC1 Hash Table Low Register	0x00000000
0x3100E010	EMAC1_SMI_ADDR	EMAC1 SMI Address Register	0x00000000
0x3100E014	EMAC1_SMI_DATA	EMAC1 SMI Data Register	0x00000000
0x3100E018	EMAC1_FLOWCTL	EMAC1 FLow Control Register	0x00000000
0x3100E01C	EMAC1_VLANTAG	EMAC1 VLAN Tag Register	0x00000000
0x3100E024	EMAC1_DBG	EMAC1 Debug Register	0x00000000
0x3100E038	EMAC1_ISTAT	EMAC1 Interrupt Status Register	0x00000000
0x3100E03C	EMAC1_IMSK	EMAC1 Interrupt Mask Register	0x00000000
0x3100E040	EMAC1_ADDR0_HI	EMAC1 MAC Address 0 High Register	0x8000FFFF
0x3100E044	EMAC1_ADDR0_LO	EMAC1 MAC Address 0 Low Register	0xFFFFFFFF
0x3100E048	EMAC1_ADDR1_HI	EMAC1 MAC Address 1 High Register	0x00000000
0x3100E04C	EMAC1_ADDR1_LO	EMAC1 MAC Address 1 Low Register	0x00000000
0x3100E0DC	EMAC1_WDOG_TIMOUT	EMAC1 Watchdog Timeout Register	0x00000000
0x3100E100	EMAC1_MMC_CTL	EMAC1 MMC Control Register	0x00000000
0x3100E104	EMAC1_MMC_RXINT	EMAC1 MMC Rx Interrupt Register	0x00000000
0x3100E108	EMAC1_MMC_TXINT	EMAC1 MMC Tx Interrupt Register	0x00000000
0x3100E10C	EMAC1_MMC_RXIMSK	EMAC1 MMC Rx Interrupt Mask Register	0x00000000
0x3100E110	EMAC1_MMC_TXIMSK	EMAC1 MMC TX Interrupt Mask Register	0x00000000
0x3100E114	EMAC1_TXOCTCNT_GB	EMAC1 Tx OCT Count (Good/Bad) Register	0x00000000
0x3100E118	EMAC1_TXFRMCNT_GB	EMAC1 Tx Frame Count (Good/Bad) Register	0x00000000
0x3100E11C	EMAC1_TXBCASTFRM_G	EMAC1 Tx Broadcast Frames (Good) Register	0x00000000
0x3100E120	EMAC1_TXMCASTFRM_G	EMAC1 Tx Multicast Frames (Good) Register	0x00000000
0x3100E124	EMAC1_TX64_GB	EMAC1 Tx 64-Byte Frames (Good/Bad) Register	0x00000000
0x3100E128	EMAC1_TX65TO127_GB	EMAC1 Tx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x3100E12C	EMAC1_TX128TO255_GB	EMAC1 Tx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000
0x3100E130	EMAC1_TX256TO511_GB	EMAC1 Tx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100E134	EMAC1_TX512TO1023_GB	EMAC1 Tx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x3100E138	EMAC1_TX1024TOMAX_GB	EMAC1 Tx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000
0x3100E13C	EMAC1_TXUCASTFRM_GB	EMAC1 Tx Unicast Frames (Good/Bad) Register	0x00000000
0x3100E140	EMAC1_TXMCASTFRM_GB	EMAC1 Tx Multicast Frames (Good/Bad) Register	0x00000000
0x3100E144	EMAC1_TXBCASTFRM_GB	EMAC1 Tx Broadcast Frames (Good/Bad) Register	0x00000000
0x3100E148	EMAC1_TXUNDR_ERR	EMAC1 Tx Underflow Error Register	0x00000000
0x3100E14C	EMAC1_TXSNGCOL_G	EMAC1 Tx Single Collision (Good) Register	0x00000000
0x3100E150	EMAC1_TXMULTCOL_G	EMAC1 Tx Multiple Collision (Good) Register	0x00000000
0x3100E154	EMAC1_TXDEFERRED	EMAC1 Tx Deferred Register	0x00000000
0x3100E158	EMAC1_TXLATECOL	EMAC1 Tx Late Collision Register	0x00000000
0x3100E15C	EMAC1_TXEXCESSCOL	EMAC1 Tx Excess Collision Register	0x00000000
0x3100E160	EMAC1_TXCARR_ERR	EMAC1 Tx Carrier Error Register	0x00000000
0x3100E164	EMAC1_TXOCTCNT_G	EMAC1 Tx Octet Count (Good) Register	0x00000000
0x3100E168	EMAC1_TXFRMCNT_G	EMAC1 Tx Frame Count (Good) Register	0x00000000
0x3100E16C	EMAC1_TXEXCESSDEF	EMAC1 Tx Excess Deferral Register	0x00000000
0x3100E170	EMAC1_TXPAUSEFRM	EMAC1 Tx Pause Frame Register	0x00000000
0x3100E174	EMAC1_TXVLANFRM_G	EMAC1 Tx VLAN Frames (Good) Register	0x00000000
0x3100E178	EMAC1_TXOVRSIZE_G	EMAC1 Number of Tx Frames (Good) greater than maxsize	0x00000000
0x3100E180	EMAC1_RXFRMCNT_GB	EMAC1 Rx Frame Count (Good/Bad) Register	0x00000000
0x3100E184	EMAC1_RXOCTCNT_GB	EMAC1 Rx Octet Count (Good/Bad) Register	0x00000000
0x3100E188	EMAC1_RXOCTCNT_G	EMAC1 Rx Octet Count (Good) Register	0x00000000
0x3100E18C	EMAC1_RXBCASTFRM_G	EMAC1 Rx Broadcast Frames (Good) Register	0x00000000
0x3100E190	EMAC1_RXMCASTFRM_G	EMAC1 Rx Multicast Frames (Good) Register	0x00000000
0x3100E194	EMAC1_RXCRC_ERR	EMAC1 Rx CRC Error Register	0x00000000
0x3100E198	EMAC1_RXALIGN_ERR	EMAC1 Rx alignment Error Register	0x00000000
0x3100E19C	EMAC1_RXRUNT_ERR	EMAC1 Rx Runt Error Register	0x00000000
0x3100E1A0	EMAC1_RXJAB_ERR	EMAC1 Rx Jab Error Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100E1A4	EMAC1_RXUSIZE_G	EMAC1 Rx Undersize (Good) Register	0x00000000
0x3100E1A8	EMAC1_RXOSIZE_G	EMAC1 Rx Oversize (Good) Register	0x00000000
0x3100E1AC	EMAC1_RX64_GB	EMAC1 Rx 64-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1B0	EMAC1_RX65TO127_GB	EMAC1 Rx 65- to 127-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1B4	EMAC1_RX128TO255_GB	EMAC1 Rx 128- to 255-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1B8	EMAC1_RX256TO511_GB	EMAC1 Rx 256- to 511-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1BC	EMAC1_RX512TO1023_GB	EMAC1 Rx 512- to 1023-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1C0	EMAC1_RX1024TOMAX_GB	EMAC1 Rx 1024- to Max-Byte Frames (Good/Bad) Register	0x00000000
0x3100E1C4	EMAC1_RXUCASTFRM_G	EMAC1 Rx Unicast Frames (Good) Register	0x00000000
0x3100E1C8	EMAC1_RXLEN_ERR	EMAC1 Rx Length Error Register	0x00000000
0x3100E1CC	EMAC1_RXOORTYPE	EMAC1 Rx Out Of Range Type Register	0x00000000
0x3100E1D0	EMAC1_RXPAUSEFRM	EMAC1 Rx Pause Frames Register	0x00000000
0x3100E1D4	EMAC1_RXFIFO_OVF	EMAC1 Rx FIFO Overflow Register	0x00000000
0x3100E1D8	EMAC1_RXVLANFRM_GB	EMAC1 Rx VLAN Frames (Good/Bad) Register	0x00000000
0x3100E1DC	EMAC1_RXWDOG_ERR	EMAC1 Rx Watch Dog Error Register	0x00000000
0x3100E1E0	EMAC1_RXRCV_ERR	EMAC1 Rx Error Frames Received Register	0x00000000
0x3100E1E4	EMAC1_RXCTLFRM_G	EMAC1 Rx Good Control Frames Register	0x00000000
0x3100E200	EMAC1_IPC_RXIMSK	EMAC1 MMC IPC Rx Interrupt Mask Register	0x00000000
0x3100E208	EMAC1_IPC_RXINT	EMAC1 MMC IPC Rx Interrupt Register	0x00000000
0x3100E210	EMAC1_RXIPV4_GD_FRM	EMAC1 Rx IPv4 Datagrams (Good) Register	0x00000000
0x3100E214	EMAC1_RXIPV4_HDR_ERR_FRM	EMAC1 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x3100E218	EMAC1_RXIPV4_NOPAY_FRM	EMAC1 Rx IPv4 Datagrams No Payload Frame Register	0x00000000
0x3100E21C	EMAC1_RXIPV4_FRAG_FRM	EMAC1 Rx IPv4 Datagrams Fragmented Frames Register	0x00000000
0x3100E220	EMAC1_RXIPV4_UDSBL_FRM	EMAC1 Rx IPv4 UDP Disabled Frames Register	0x00000000
0x3100E224	EMAC1_RXIPV6_GD_FRM	EMAC1 Rx IPv6 Datagrams Good Frames Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100E228	EMAC1_RXIPV6_HDR_ERR_FRM	EMAC1 Rx IPv6 Datagrams Header Error Frames Register	0x00000000
0x3100E22C	EMAC1_RXIPV6_NOPAY_FRM	EMAC1 Rx IPv6 Datagrams No Payload Frames Register	0x00000000
0x3100E230	EMAC1_RXUDP_GD_FRM	EMAC1 Rx UDP Good Frames Register	0x00000000
0x3100E234	EMAC1_RXUDP_ERR_FRM	EMAC1 Rx UDP Error Frames Register	0x00000000
0x3100E238	EMAC1_RXTCP_GD_FRM	EMAC1 Rx TCP Good Frames Register	0x00000000
0x3100E23C	EMAC1_RXTCP_ERR_FRM	EMAC1 Rx TCP Error Frames Register	0x00000000
0x3100E240	EMAC1_RXICMP_GD_FRM	EMAC1 Rx ICMP Good Frames Register	0x00000000
0x3100E244	EMAC1_RXICMP_ERR_FRM	EMAC1 Rx ICMP Error Frames Register	0x00000000
0x3100E250	EMAC1_RXIPV4_GD_OCT	EMAC1 Rx IPv4 Datagrams Good Octets Register	0x00000000
0x3100E254	EMAC1_RXIPV4_HDR_ERR_OCT	EMAC1 Rx IPv4 Datagrams Header Errors Register	0x00000000
0x3100E258	EMAC1_RXIPV4_NOPAY_OCT	EMAC1 Rx IPv4 Datagrams No Payload Octets Register	0x00000000
0x3100E25C	EMAC1_RXIPV4_FRAG_OCT	EMAC1 Rx IPv4 Datagrams Fragmented Octets Register	0x00000000
0x3100E260	EMAC1_RXIPV4_UDSBL_OCT	EMAC1 Rx IPv4 UDP Disabled Octets Register	0x00000000
0x3100E264	EMAC1_RXIPV6_GD_OCT	EMAC1 Rx IPv6 Good Octets Register	0x00000000
0x3100E268	EMAC1_RXIPV6_HDR_ERR_OCT	EMAC1 Rx IPv6 Header Errors Register	0x00000000
0x3100E26C	EMAC1_RXIPV6_NOPAY_OCT	EMAC1 Rx IPv6 No Payload Octets Register	0x00000000
0x3100E270	EMAC1_RXUDP_GD_OCT	EMAC1 Rx UDP Good Octets Register	0x00000000
0x3100E274	EMAC1_RXUDP_ERR_OCT	EMAC1 Rx UDP Error Octets Register	0x00000000
0x3100E278	EMAC1_RXTCP_GD_OCT	EMAC1 Rx TCP Good Octets Register	0x00000000
0x3100E27C	EMAC1_RXTCP_ERR_OCT	EMAC1 Rx TCP Error Octets Register	0x00000000
0x3100E280	EMAC1_RXICMP_GD_OCT	EMAC1 Rx ICMP Good Octets Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100E284	EMAC1_RXICMP_ERR_OCT	EMAC1 Rx ICMP Error Octets Register	0x00000000
0x3100E400	EMAC1_L3L4_CTL	EMAC1 Layer3 and Layer4 Control Register	0x00000000
0x3100E404	EMAC1_L4_ADDR	EMAC1 Layer 4 Address Register	0x00000000
0x3100E410	EMAC1_L3_ADDR0	EMAC1 Layer 3 Address0 Register	0x00000000
0x3100E414	EMAC1_L3_ADDR1	EMAC1 Layer 3 Address1 Register	0x00000000
0x3100E418	EMAC1_L3_ADDR2	EMAC1 Layer 3 Address2 Register	0x00000000
0x3100E41C	EMAC1_L3_ADDR3	EMAC1 Layer 3 Address3 Register	0x00000000
0x3100E584	EMAC1_VLAN_INCL	EMAC1 VLAN Tag Inclusion or Replacement Register	0x00000000
0x3100E588	EMAC1_VLAN_HSHTBL	EMAC1 VLAN Hash Table Register	0x00000000
0x3100E760	EMAC1_TM_PPS0INTVL	EMAC1 Time Stamp PPS Interval Register	0x00000000
0x3100E764	EMAC1_TM_PPS0WIDTH	EMAC1 PPS Width Register	0x00000000
0x3100F000	EMAC1_DMA0_BUSMODE	EMAC1 DMA Bus Mode Register	0x00020101
0x3100F004	EMAC1_DMA0_TXPOLL	EMAC1 DMA Tx Poll Demand Register	0x00000000
0x3100F008	EMAC1_DMA0_RXPOLL	EMAC1 DMA Rx Poll Demand register	0x00000000
0x3100F00C	EMAC1_DMA0_RXDSC_ADDR	EMAC1 DMA Rx Descriptor List Address Register	0x00000000
0x3100F010	EMAC1_DMA0_TXDSC_ADDR	EMAC1 DMA Tx Descriptor List Address Register	0x00000000
0x3100F014	EMAC1_DMA0_STAT	EMAC1 DMA Status Register	0x00000000
0x3100F018	EMAC1_DMA0_OPMODE	EMAC1 DMA Operation Mode Register	0x00000000
0x3100F01C	EMAC1_DMA0_IEN	EMAC1 DMA Interrupt Enable Register	0x00000000
0x3100F020	EMAC1_DMA0_MISS_FRM	EMAC1 DMA Missed Frame Register	0x00000000
0x3100F024	EMAC1_DMA0_RXIWDOG	EMAC1 DMA Rx Interrupt Watch Dog Register	0x00000000
0x3100F028	EMAC1_DMA0_BMMODE	EMAC1 DMA SCB Bus Mode Register	0x40110001
0x3100F02C	EMAC1_DMA0_BMSTAT	EMAC1 DMA SCB Status Register	0x00000000
0x3100F048	EMAC1_DMA0_TXDSC_CUR	EMAC1 DMA Tx Descriptor Current Register	0x00000000
0x3100F04C	EMAC1_DMA0_RXDSC_CUR	EMAC1 DMA Rx Descriptor Current Register	0x00000000

Table A-73: ADSP-SC58x EMAC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100F050	EMAC1_DMA0_TXBUF_CUR	EMAC1 DMA Tx Buffer Current Register	0x00000000
0x3100F054	EMAC1_DMA0_RXBUF_CUR	EMAC1 DMA Rx Buffer Current Register	0x00000000

Table A-74: ADSP-SC58x EMDMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C602C	EMDMA0_CTL	EMDMA0 External Memory DMA Control Register	0x00000000
0x310C6080	EMDMA0_INDX1	EMDMA0 External Index Register	0x00000000
0x310C6084	EMDMA0_MOD1	EMDMA0 External Modifier Register	0x00000000
0x310C6088	EMDMA0_CNT1	EMDMA0 External Count Register	0x00000000
0x310C608C	EMDMA0_INDX0	EMDMA0 Internal Index Register	0x00000000
0x310C6090	EMDMA0_MOD0	EMDMA0 Internal Modifier Register	0x00000000
0x310C6094	EMDMA0_CNT0	EMDMA0 Internal Count Register	0x00000000
0x310C6098	EMDMA0_CHNPTR	EMDMA0 Chain Pointer Register	0x00000000
0x310C609C	EMDMA0_BASE	EMDMA0 External Base Address Register	0x00000000
0x310C60A0	EMDMA0_TPTR	EMDMA0 Tap List Pointer Register	0x00000000
0x310C60A4	EMDMA0_BUFLEN	EMDMA0 Circular Buffer Length Register	0x00000000
0x310C60AC	EMDMA0_TCNT	EMDMA0 Delay Line Tap Count Register	0x00000000

Table A-75: ADSP-SC58x EMDMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C6030	EMDMA1_CTL	EMDMA1 External Memory DMA Control Register	0x00000000
0x310C60C0	EMDMA1_INDX1	EMDMA1 External Index Register	0x00000000
0x310C60C4	EMDMA1_MOD1	EMDMA1 External Modifier Register	0x00000000
0x310C60C8	EMDMA1_CNT1	EMDMA1 External Count Register	0x00000000
0x310C60CC	EMDMA1_INDX0	EMDMA1 Internal Index Register	0x00000000
0x310C60D0	EMDMA1_MOD0	EMDMA1 Internal Modifier Register	0x00000000
0x310C60D4	EMDMA1_CNT0	EMDMA1 Internal Count Register	0x00000000
0x310C60D8	EMDMA1_CHNPTR	EMDMA1 Chain Pointer Register	0x00000000
0x310C60DC	EMDMA1_BASE	EMDMA1 External Base Address Register	0x00000000

Table A-75: ADSP-SC58x EMDMA1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C60E0	EMDMA1_TPTR	EMDMA1 Tap List Pointer Register	0x00000000
0x310C60E4	EMDMA1_BUFLEN	EMDMA1 Circular Buffer Length Register	0x00000000
0x310C60EC	EMDMA1_TCNT	EMDMA1 Delay Line Tap Count Register	0x00000000

Table A-76: ADSP-SC58x EPPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31040000	EPPI0_STAT	EPPI0 Status Register	0x00000000
0x31040004	EPPI0_HCNT	EPPI0 Horizontal Transfer Count Register	0x00000000
0x31040008	EPPI0_HDLY	EPPI0 Horizontal Delay Count Register	0x00000000
0x3104000C	EPPI0_VCNT	EPPI0 Vertical Transfer Count Register	0x00000000
0x31040010	EPPI0_VDLY	EPPI0 Vertical Delay Count Register	0x00000000
0x31040014	EPPI0_FRAME	EPPI0 Lines Per Frame Register	0x00000000
0x31040018	EPPI0_LINE	EPPI0 Samples Per Line Register	0x00000000
0x3104001C	EPPI0_CLKDIV	EPPI0 Clock Divide Register	0x00000000
0x31040020	EPPI0_CTL	EPPI0 Control Register	0x00000000
0x31040024	EPPI0_FS1_WLHB	EPPI0 FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register	0x00000000
0x31040028	EPPI0_FS1_PASPL	EPPI0 FS1 Period Register / EPPI Active Samples Per Line Register	0x00000000
0x3104002C	EPPI0_FS2_WLVB	EPPI0 FS2 Width Register / EPPI Lines Of Vertical Blanking Register	0x00000000
0x31040030	EPPI0_FS2_PALPF	EPPI0 FS2 Period Register / EPPI Active Lines Per Field Register	0x00000000
0x31040034	EPPI0_IMSK	EPPI0 Interrupt Mask Register	0x00000000
0x3104003C	EPPI0_ODDCLIP	EPPI0 Clipping Register for ODD (Chroma) Data Register	0xFFFF0000
0x31040040	EPPI0_EVENCLIP	EPPI0 Clipping Register for EVEN (Luma) Data Register	0xFFFF0000
0x31040044	EPPI0_FS1_DLY	EPPI0 Frame Sync 1 Delay Value Register	0x00000000
0x31040048	EPPI0_FS2_DLY	EPPI0 Frame Sync 2 Delay Value Register	0x00000000
0x3104004C	EPPI0_CTL2	EPPI0 Control Register 2 Register	0x00000000

Table A-77: ADSP-SC58x FFTA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C000	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C004	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C008	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C00C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C010	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C014	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C018	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C01C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C020	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C024	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C028	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C02C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C030	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C034	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C038	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C03C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C040	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C044	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C048	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C04C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C050	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C054	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C058	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C05C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C060	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C064	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C068	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C06C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C070	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C074	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C078	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000

Table A-77: ADSP-SC58x FFTA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C07C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C080	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C084	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C088	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C08C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C090	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C094	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C098	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C09C	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0A0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0A4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0A8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0AC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0B0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0B4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0B8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0BC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0C0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0C4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0C8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0CC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0D0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0D4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0D8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0DC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0E0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0E4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0E8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0EC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0F0	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0F4	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000

Table A-77: ADSP-SC58x FFTA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109C0F8	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C0FC	FFTA0_INST[nn]	FFTA0 Instruction Memory Register	0x00000000
0x3109C100	FFTA0_CTL	FFTA0 Control Register	0x00000000
0x3109C104	FFTA0_STAT	FFTA0 Status Register	0x00000000
0x3109C108	FFTA0_PC	FFTA0 Program Counter Register	0x00000000
0x3109C10C	FFTA0_THREADOFFSET	FFTA0 Thread Count Offset Register	0x00000000
0x3109C110	FFTA0_SCALE	FFTA0 FFT/IFFT Scale Factor Register	0x3F800000
0x3109C114	FFTA0_XFRLEFT[nn]	FFTA0 Load/Dump Transfer Left Register	0x00000000
0x3109C118	FFTA0_XFRLEFT[nn]	FFTA0 Load/Dump Transfer Left Register	0x00000000
0x3109C11C	FFTA0_XFRLEFT[nn]	FFTA0 Load/Dump Transfer Left Register	0x00000000
0x3109C120	FFTA0_LC[nn]	FFTA0 Loop Counter Value Register	0x00000000
0x3109C124	FFTA0_LC[nn]	FFTA0 Loop Counter Value Register	0x00000000
0x3109C600	FFTA0_WCTL	FFTA0 Wrapper Control Register	0x00000000

Table A-78: ADSP-SC58x FIR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C3000	FIR0_CTL1	FIR0 FIR Global Control Register	0x00000000
0x310C3004	FIR0_DMASTAT	FIR0 FIR DMA Status Register	0x00000000
0x310C3008	FIR0_MACSTAT	FIR0 FIR MAC Status Register	0x00000000
0x310C3010	FIR0_DBG_CTL	FIR0 FIR Debug Control Register	0x00000000
0x310C3014	FIR0_DBG_ADDR	FIR0 Debug Address Register	0x00000000
0x310C3018	FIR0_DBG_WRDAT	FIR0 FIR Debug Data Write Register	0x00000000
0x310C301C	FIR0_DBG_RDDAT	FIR0 FIR Debug Data Read Register	0x00000000
0x310C3040	FIR0_CTL2	FIR0 FIR Channel Control Register	0x00000000
0x310C3044	FIR0_INIDX	FIR0 FIR Input Data Index Register	0x00000000
0x310C3048	FIR0_INMOD	FIR0 FIR Input Data Modifier Register	0x00000000
0x310C304C	FIR0_INCNT	FIR0 FIR Input Data Count Register	0x00000000
0x310C3050	FIR0_INBASE	FIR0 FIR Input Data Base Register	0x00000000
0x310C3054	FIR0_OUTIDX	FIR0 FIR Output Data Index Register	0x00000000
0x310C3058	FIR0_OUTMOD	FIR0 FIR Output Data Modifier Register	0x00000000

Table A-78: ADSP-SC58x FIR0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C305C	FIR0_OUTCNT	FIR0 FIR Output Data Count Register	0x00000000
0x310C3060	FIR0_OUTBASE	FIR0 FIR Output Data Base Register	0x00000000
0x310C3064	FIR0_COEFIDX	FIR0 FIR Coefficient Index Register	0x00000000
0x310C3068	FIR0_COEFMOD	FIR0 FIR Coefficient Modifier Register	0x00000000
0x310C306C	FIR0_COEFCNT	FIR0 FIR Coefficient Count Register	0x00000000
0x310C3070	FIR0_CHNPTR	FIR0 FIR Chain Pointer Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2000	GICDST0_EN	GICDST0 GIC Port 0 Enable	0x00000000
0x310B2080	GICDST0_SGI_SECURITY	GICDST0 Software Generated Interrupt Security Register	0x00000000
0x310B2084	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B2088	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B208C	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B2090	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B2094	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B2098	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B209C	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B20A0	GICDST0_SPI_SECURITY[n]	GICDST0 Shared Peripheral Interrupt Security Register	0x00000000
0x310B2104	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2108	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B210C	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2110	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2114	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2118	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B211C	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2120	GICDST0_SPI_EN_SET[n]	GICDST0 Shared Peripheral Interrupt Enable Set Register	0x00000000
0x310B2184	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B2188	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B218C	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B2190	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B2194	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B2198	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B219C	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B21A0	GICDST0_SPI_EN_CLR[n]	GICDST0 Shared Peripheral Interrupt Enable Clear Register	0x00000000
0x310B2200	GICDST0_SGI_PND_SET	GICDST0 Software Generated Interrupt Pending Set Register	0x00000000
0x310B2204	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B2208	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B220C	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B2210	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B2214	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B2218	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B221C	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000
0x310B2220	GICDST0_SPI_PND_SET[n]	GICDST0 Shared Peripheral Interrupt Pending Set Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2280	GICDST0_SGI_PND_CLR	GICDST0 Software Generated Interrupt Clear-Pending Register	0x00000000
0x310B2284	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B2288	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B228C	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B2290	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B2294	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B2298	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B229C	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B22A0	GICDST0_SPI_PND_CLR[n]	GICDST0 Shared Peripheral Interrupt Pending Clear Register	0x00000000
0x310B2300	GICDST0_SGI_ACTIVE	GICDST0 Software Generated Interrupt Active Register	0x00000000
0x310B2304	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2308	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B230C	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2310	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2314	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2318	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B231C	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2320	GICDST0_SPI_ACTIVE[n]	GICDST0 Shared Peripheral Interrupt Active Register	0x00000000
0x310B2400	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2401	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2402	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2403	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2404	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2405	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2406	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2407	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2408	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2409	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240A	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240B	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240C	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240D	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240E	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B240F	GICDST0_SGI_PRIO[n]	GICDST0 Software Generated Interrupt Priority Register	0x00000000
0x310B2420	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2421	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2422	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2423	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2424	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2425	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2426	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2427	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2428	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2429	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B242F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2430	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2431	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2432	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2433	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2434	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2435	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2436	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2437	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2438	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2439	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B243F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2440	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2441	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2442	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2443	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2444	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2445	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2446	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2447	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2448	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2449	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B244F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2450	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2451	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2452	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2453	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2454	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2455	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2456	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2457	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2458	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2459	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B245F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2460	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2461	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2462	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2463	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2464	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2465	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2466	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2467	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2468	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2469	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B246F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2470	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2471	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2472	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2473	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2474	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2475	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2476	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2477	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2478	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2479	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B247F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2480	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2481	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2482	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2483	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2484	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2485	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2486	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2487	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2488	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2489	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B248F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2490	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2491	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2492	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2493	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2494	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2495	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2496	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2497	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2498	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2499	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B249F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24A9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AD	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24AF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B24B2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24B9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BD	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24BF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24C9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CD	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24CF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B24D1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24D9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DD	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24DF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24E9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24EA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24EB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24EC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24ED	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24EE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24EF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B24F0	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F1	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F2	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F3	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F4	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F5	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F6	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F7	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F8	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24F9	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FA	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FB	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FC	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FD	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FE	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B24FF	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2500	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2501	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2502	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2503	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2504	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2505	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2506	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2507	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2508	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2509	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B250A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B250B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B250C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B250D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B250E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B250F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2510	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2511	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2512	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2513	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2514	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2515	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2516	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2517	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2518	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2519	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251A	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251B	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251C	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251D	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251E	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B251F	GICDST0_SPI_PRIO[n]	GICDST0 Shared Peripheral Interrupt Priority Register	0x00000000
0x310B2820	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2821	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2822	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2823	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2824	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2825	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2826	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2827	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2828	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2829	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B282F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2830	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2831	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2832	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2833	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2834	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2835	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2836	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2837	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2838	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2839	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B283A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B283B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B283C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B283D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B283E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B283F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2840	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2841	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2842	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2843	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2844	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2845	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2846	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2847	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2848	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2849	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B284A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B284B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B284C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B284D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B284E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B284F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2850	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2851	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2852	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2853	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2854	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2855	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2856	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2857	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2858	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2859	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B285A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B285B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B285C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B285D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B285E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B285F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2860	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2861	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2862	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2863	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2864	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2865	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2866	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2867	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2868	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2869	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B286F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2870	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2871	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2872	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2873	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2874	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2875	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2876	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2877	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2878	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2879	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B287F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2880	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2881	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2882	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2883	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2884	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2885	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2886	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2887	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2888	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2889	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B288F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2890	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2891	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2892	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2893	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2894	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2895	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2896	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2897	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2898	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2899	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B289F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B28A6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28A9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AD	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28AF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B28B8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28B9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BD	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28BF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28C9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B28CA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28CB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28CC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28CD	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28CE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28CF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28D9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28DA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28DB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B28DC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28DD	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28DE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28DF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28E9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28EA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28EB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28EC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28ED	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B28EE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28EF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F0	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F1	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F2	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F3	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F4	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F5	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F6	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F7	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F8	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28F9	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FA	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FB	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FC	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FD	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FE	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B28FF	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2900	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2901	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2902	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2903	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2904	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2905	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2906	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2907	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2908	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2909	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B290F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2910	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2911	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2912	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2913	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2914	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2915	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2916	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2917	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2918	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2919	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291A	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291B	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291C	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291D	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291E	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B291F	GICDST0_SPI_TRGT[n]	GICDST0 Shared Peripheral Interrupt Processor Targets Register	0x00000000
0x310B2C08	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C0C	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C10	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C14	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000

Table A-79: ADSP-SC58x GICDST0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B2C18	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C1C	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C20	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C24	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C28	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C2C	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C30	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C34	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C38	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C3C	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C40	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2C44	GICDST0_SPI_CFG[n]	GICDST0 Shared Peripheral Interrupt Configuration Register	0x00000000
0x310B2D04	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D08	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D0C	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D10	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D14	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D18	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D1C	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2D20	GICDST0_SPI[n]	GICDST0 Shared Peripheral Interrupt Register	0x00000000
0x310B2F00	GICDST0_SGI_CTL	GICDST0 Software Generated Interrupt Control Register	0x00000000

Table A-80: ADSP-SC58x GICCPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310B4000	GICCPU1_CTL	GICCPU1 CPU Interface Control Register (ICCICR)	0x00000000
0x310B4004	GICCPU1_PRIO_MSK	GICCPU1 Priority Mask Register (ICCIPMR)	0x00000000
0x310B4008	GICCPU1_BIN_PT	GICCPU1 Binary Point Register (ICCBPR)	0x00000000
0x310B400C	GICCPU1_INT_ACK	GICCPU1 Interrupt Acknowledge Register (ICCIAR)	0x00000000
0x310B4010	GICCPU1_EOI	GICCPU1 End of Interrupt Register (ICCEOIR)	0x00000000
0x310B4014	GICCPU1_RUN_PRIO	GICCPU1 Running Priority Register (ICCRPR)	0x00000000
0x310B4018	GICCPU1_PND_HI	GICCPU1 Highest Pending Interrupt Register (ICCHPIR)	0x00000000
0x310B401C	GICCPU1_BIN_PT_ALIAS	GICCPU1 Aliased Binary Point Register (ICCABPR)	0x00000000

Table A-81: ADSP-SC58x HADC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31017000	HADC0_CTL	HADC0 Control Register	0x000001AA
0x31017004	HADC0_CHAN_MSK	HADC0 Channel Mask Register	0x0000FF00
0x31017008	HADC0_IMSK	HADC0 Interrupt Mask Register	0x00000000
0x3101700C	HADC0_STAT	HADC0 Status Register	0x00000000
0x31017010	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017014	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017018	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x3101701C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017020	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017024	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017028	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x3101702C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017030	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017034	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017038	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x3101703C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017040	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017044	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x31017048	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000

Table A-81: ADSP-SC58x HADC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3101704C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000

Table A-82: ADSP-SC58x HAE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310180A0	HAE0_RUN	HAE0 Run Register	0x00000000
0x31018B00	HAE0_CFG0	HAE0 Configuration 0 Register	0x00000000
0x31018B04	HAE0_CFG1	HAE0 Configuration 1 Register	0x00000000
0x31018B08	HAE0_CFG2	HAE0 Configuration 2 Register	0x00000000
0x31018B0C	HAE0_CFG3	HAE0 Configuration 3 Register	0x00000000
0x31018B20	HAE0_STAT	HAE0 Status Register	0x00000000
0x31018B40	HAE0_ISAMPLE	HAE0 I (Current) Sample Register	0x00000000
0x31018B44	HAE0_VSAMPLE	HAE0 V (Voltage) Sample Register	0x00000000
0x31018B80	HAE0_IWAVEFORM	HAE0 I (Current) Waveform Register	0x00000000
0x31018B84	HAE0_VWAVEFORM	HAE0 V (Voltage) Waveform Register	0x00000000
0x31019E00	HAE0_CFG4	HAE0 Configuration 4 Register	0x00000000
0x31019E04	HAE0_DIDT_GAIN	HAE0 DIDT Gain Register	0x00000000
0x31019E08	HAE0_DIDT_COEF	HAE0 DIDT Coefficient Register	0x00000000
0x31019E10	HAE0_VLEVEL	HAE0 Voltage Level Register	0x00000000
0x31019E2C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E30	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E34	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E38	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E3C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E40	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E44	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E48	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E4C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E50	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E54	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x31019E58	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000

Table A-83: ADSP-SC58x IIR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C4000	IIR0_CTL1	IIR0 Global Control Register	0x00000000
0x310C4004	IIR0_DMASTAT	IIR0 DMA Status Register	0x00000000
0x310C4008	IIR0_MACSTAT	IIR0 MAC Status Register	0x00000000
0x310C400C	IIR0_DBG_CTL	IIR0 IIR Debug Control Register	0x00000000
0x310C4010	IIR0_DBG_ADDR	IIR0 IIR Debug Address Register	0x00000000
0x310C4014	IIR0_DBG_WRDAT_LO	IIR0 IIR Debug Write Data Low Register	0x00000000
0x310C4018	IIR0_DBG_WRDAT_HI	IIR0 IIR Debug Write Data High Register	0x00000000
0x310C401C	IIR0_DBG_RDDAT_LO	IIR0 IIR Debug Read Data Low Register	0x00000000
0x310C4020	IIR0_DBG_RDDAT_HI	IIR0 IIR Debug Read Data High Register	0x00000000
0x310C4040	IIR0_CTL2	IIR0 Channel Control Register	0x00000000
0x310C4044	IIR0_INIDX	IIR0 Input Data Index Register	0x00000000
0x310C4048	IIR0_INMOD	IIR0 Input Data Index Modifier Register	0x00000000
0x310C404C	IIR0_INLEN	IIR0 Input Data Buffer Length Register	0x00000000
0x310C4050	IIR0_INBASE	IIR0 Input Buffer Base Register	0x00000000
0x310C4054	IIR0_OUTIDX	IIR0 Output Data Buffer Index Register	0x00000000
0x310C4058	IIR0_OUTMOD	IIR0 IIR Output Data Index Modifier Register	0x00000000
0x310C405C	IIR0_OUTLEN	IIR0 IIR Output Data Buffer Length Register	0x00000000
0x310C4060	IIR0_OUTBASE	IIR0 Output Buffer Base Register	0x00000000
0x310C4064	IIR0_COEFIDX	IIR0 Coefficient Buffer Index Register	0x00000000
0x310C4068	IIR0_COEFMOD	IIR0 Coefficient Index Modifier Register	0x00000000
0x310C406C	IIR0_COEFLEN	IIR0 Coefficient Buffer Length Register	0x00000000
0x310C4070	IIR0_CHNPTR	IIR0 Chain Pointer Register	0x00000000

Table A-84: ADSP-SC58x L2CTL0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31080000	L2CTL0_CTL	L2CTL0 Control Register	0x00000000
0x31080010	L2CTL0_STAT	L2CTL0 Status Register	0x01000000
0x31080014	L2CTL0_RPCR	L2CTL0 Read Priority Count Register	0x00000F0F
0x31080018	L2CTL0_WPCR	L2CTL0 Write Priority Count Register	0x00000F0F
0x31080024	L2CTL0_RFA	L2CTL0 Refresh Address Register	0x20080000

Table A-84: ADSP-SC58x L2CTL0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31080040	L2CTL0_ERRADDR0	L2CTL0 ECC Error Address 0 Register	0x20080000
0x31080044	L2CTL0_ERRADDR1	L2CTL0 ECC Error Address 1 Register	0x20080000
0x31080048	L2CTL0_ERRADDR2	L2CTL0 ECC Error Address 2 Register	0x20080000
0x3108004C	L2CTL0_ERRADDR3	L2CTL0 ECC Error Address 3 Register	0x20080000
0x31080050	L2CTL0_ERRADDR4	L2CTL0 ECC Error Address 4 Register	0x20080000
0x31080054	L2CTL0_ERRADDR5	L2CTL0 ECC Error Address 5 Register	0x20080000
0x31080058	L2CTL0_ERRADDR6	L2CTL0 ECC Error Address 6 Register	0x20080000
0x3108005C	L2CTL0_ERRADDR7	L2CTL0 ECC Error Address 7 Register	0x20080000
0x31080080	L2CTL0_ET0	L2CTL0 Error Type 0 Register	0x00000000
0x31080084	L2CTL0_EADDR0	L2CTL0 Error Type 0 Address Register	0x20000000
0x31080088	L2CTL0_ET1	L2CTL0 Error Type 1 Register	0x00000000
0x3108008C	L2CTL0_EADDR1	L2CTL0 Error Type 1 Address Register	0x20000000

Table A-85: ADSP-SC58x L2CTL1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31081000	L2CTL1_CTL	L2CTL1 Control Register	0x00000000
0x31081010	L2CTL1_STAT	L2CTL1 Status Register	0x01000000
0x31081014	L2CTL1_RPCR	L2CTL1 Read Priority Count Register	0x00000F0F
0x31081018	L2CTL1_WPCR	L2CTL1 Write Priority Count Register	0x00000F0F
0x31081024	L2CTL1_RFA	L2CTL1 Refresh Address Register	0x20180000
0x31081040	L2CTL1_ERRADDR0	L2CTL1 ECC Error Address 0 Register	0x20180000
0x31081044	L2CTL1_ERRADDR1	L2CTL1 ECC Error Address 1 Register	0x20180000
0x31081048	L2CTL1_ERRADDR2	L2CTL1 ECC Error Address 2 Register	0x20180000
0x3108104C	L2CTL1_ERRADDR3	L2CTL1 ECC Error Address 3 Register	0x20180000
0x31081050	L2CTL1_ERRADDR4	L2CTL1 ECC Error Address 4 Register	0x20180000
0x31081054	L2CTL1_ERRADDR5	L2CTL1 ECC Error Address 5 Register	0x20180000
0x31081058	L2CTL1_ERRADDR6	L2CTL1 ECC Error Address 6 Register	0x20180000
0x3108105C	L2CTL1_ERRADDR7	L2CTL1 ECC Error Address 7 Register	0x20180000
0x31081080	L2CTL1_ET0	L2CTL1 Error Type 0 Register	0x00000000
0x31081084	L2CTL1_EADDR0	L2CTL1 Error Type 0 Address Register	0x20100000

Table A-85: ADSP-SC58x L2CTL1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31081088	L2CTL1_ET1	L2CTL1 Error Type 1 Register	0x00000000
0x3108108C	L2CTL1_EADDR1	L2CTL1 Error Type 1 Address Register	0x20100000

Table A-86: ADSP-SC58x L2CTL2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31082000	L2CTL2_CTL	L2CTL2 Control Register	0x00000000
0x31082010	L2CTL2_STAT	L2CTL2 Status Register	0x01000000
0x31082014	L2CTL2_RPCR	L2CTL2 Read Priority Count Register	0x00000F0F
0x31082018	L2CTL2_WPCR	L2CTL2 Write Priority Count Register	0x00000F0F
0x31082024	L2CTL2_RFA	L2CTL2 Refresh Address Register	0x20280000
0x31082040	L2CTL2_ERRADDR0	L2CTL2 ECC Error Address 0 Register	0x20280000
0x31082044	L2CTL2_ERRADDR1	L2CTL2 ECC Error Address 1 Register	0x20280000
0x31082048	L2CTL2_ERRADDR2	L2CTL2 ECC Error Address 2 Register	0x20280000
0x3108204C	L2CTL2_ERRADDR3	L2CTL2 ECC Error Address 3 Register	0x20280000
0x31082050	L2CTL2_ERRADDR4	L2CTL2 ECC Error Address 4 Register	0x20280000
0x31082054	L2CTL2_ERRADDR5	L2CTL2 ECC Error Address 5 Register	0x20280000
0x31082058	L2CTL2_ERRADDR6	L2CTL2 ECC Error Address 6 Register	0x20280000
0x3108205C	L2CTL2_ERRADDR7	L2CTL2 ECC Error Address 7 Register	0x20280000
0x31082080	L2CTL2_ET0	L2CTL2 Error Type 0 Register	0x00000000
0x31082084	L2CTL2_EADDR0	L2CTL2 Error Type 0 Address Register	0x20200000
0x31082088	L2CTL2_ET1	L2CTL2 Error Type 1 Register	0x00000000
0x3108208C	L2CTL2_EADDR1	L2CTL2 Error Type 1 Address Register	0x20200000

Table A-87: ADSP-SC58x LP0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFE000	LP0_CTL	LP0 Control Register	0x00000000
0x30FFE004	LP0_STAT	LP0 Status Register	0x00000000
0x30FFE008	LP0_DIV	LP0 Clock Divider Value Register	0x00000000
0x30FFE010	LP0_TX	LP0 Transmit Buffer Register	0x00000000
0x30FFE014	LP0_RX	LP0 Receive Buffer Register	0x00000000

Table A-87: ADSP-SC58x LP0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFE018	LP0_TXIN_SHDW	LP0 Shadow Input Transmit Buffer Register	0x00000000
0x30FFE01C	LP0_TXOUT_SHDW	LP0 Shadow Output Transmit Buffer Register	0x00000000

Table A-88: ADSP-SC58x LP1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x30FFE100	LP1_CTL	LP1 Control Register	0x00000000
0x30FFE104	LP1_STAT	LP1 Status Register	0x00000000
0x30FFE108	LP1_DIV	LP1 Clock Divider Value Register	0x00000000
0x30FFE110	LP1_TX	LP1 Transmit Buffer Register	0x00000000
0x30FFE114	LP1_RX	LP1 Receive Buffer Register	0x00000000
0x30FFE118	LP1_TXIN_SHDW	LP1 Shadow Input Transmit Buffer Register	0x00000000
0x30FFE11C	LP1_TXOUT_SHDW	LP1 Shadow Output Transmit Buffer Register	0x00000000

Table A-89: ADSP-SC58x M6CPH0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C5000	M6CPH0_CTL	M6CPH0 M6 Wrapper Control register	0x00000000
0x310C5038	M6CPH0_WRMOD	M6CPH0 Write DMA Modifier	0x00000000
0x310C503C	M6CPH0_RDMOD	M6CPH0 Read DMA Modifier	0x00000000
0x310C5040	M6CPH0_WRINDX	M6CPH0 Memory Write Index	0x00000000
0x310C5044	M6CPH0_RDINDX	M6CPH0 Memory Read Index	0x00000000
0x310C5058	M6CPH0_WRCNT	M6CPH0 Write DMA Counter	0x00000000
0x310C505C	M6CPH0_RDCNT	M6CPH0 Read DMA Counter	0x00000000
0x310C5080	M6CPH0_DATA	M6CPH0 M6 Cipher Engine Data Register	0x00000000
0x310C5084	M6CPH0_KEY_HI	M6CPH0 M6 Cipher Engine Key High Register	0x00000000
0x310C5088	M6CPH0_KEY_LO	M6CPH0 M6 Cipher Engine Key Low Register	0x00000000
0x310C508C	M6CPH0_ENGINE_CTL	M6CPH0 M6 Cipher Engine Control Register	0x00000000
0x310C5090	M6CPH0_ENCRSZ	M6CPH0 M6 Cipher Engine Encryption size register	0x00000000
0x310C5094	M6CPH0_STAT	M6CPH0 M6 Cipher Engine status register	0x00000000

Table A-90: ADSP-SC58x MLB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109D000	MLB0_CTL0	MLB0 MediaLB Control 0 Register	0x00000000
0x3109D008	MLB0_PCTL0	MLB0 MediaLB 6-pin Control 0 Register	0x00000000
0x3109D00C	MLB0_MS0	MLB0 Channel Status 0 Register	0x00000000
0x3109D014	MLB0_MS1	MLB0 Channel Status 1 Register	0x00000000
0x3109D020	MLB0_MSS	MLB0 System Status Register	0x00000000
0x3109D024	MLB0_MSD	MLB0 System Data Register	0x00000000
0x3109D02C	MLB0_MIEN	MLB0 Interrupt Enable Register	0x00000000
0x3109D034	MLB0_GCTL	MLB0 MLB Global Control Register	0x00000000
0x3109D03C	MLB0_CTL1	MLB0 Control 1 Register	0x00000000
0x3109D080	MLB0_HCTL	MLB0 HBI Control Register	0x00000000
0x3109D088	MLB0_HCMR0	MLB0 HBI Channel Mask 0 Register	0x00000000
0x3109D08C	MLB0_HCMR1	MLB0 HBI Channel Mask 1 Register	0x00000000
0x3109D090	MLB0_HCER0	MLB0 HBI Channel Error 0 Register	0x00000000
0x3109D094	MLB0_HCER1	MLB0 HBI Channel Error 1 Register	0x00000000
0x3109D098	MLB0_HCBR0	MLB0 HBI Channel Busy 0 Register	0x00000000
0x3109D09C	MLB0_HCBR1	MLB0 HBI Channel Busy 1 Register	0x00000000
0x3109D0C0	MLB0_MDAT0	MLB0 Memory Interface Control Data 0 Register	0x00000000
0x3109D0C4	MLB0_MDAT1	MLB0 Memory Interface Control Data 1 Register	0x00000000
0x3109D0C8	MLB0_MDAT2	MLB0 Memory Interface Control Data 2 Register	0x00000000
0x3109D0CC	MLB0_MDAT3	MLB0 Memory Interface Control Data 3 Register	0x00000000
0x3109D0D0	MLB0_MDWE0	MLB0 Memory Interface Control Data Write Enable 0 Register	0x00000000
0x3109D0D4	MLB0_MDWE1	MLB0 Memory Interface Control Data Write Enable 1 Register	0x00000000
0x3109D0D8	MLB0_MDWE2	MLB0 Memory Interface Control Data Write Enable 2 Register	0x00000000
0x3109D0DC	MLB0_MDWE3	MLB0 Memory Interface Control Data Write Enable 3 Register	0x00000000
0x3109D0E0	MLB0_MCTL	MLB0 Memory Interface Control Register	0x00000000
0x3109D0E4	MLB0_MADR	MLB0 Memory Interface Address Register	0x00000000
0x3109D3C0	MLB0_ACTL	MLB0 Bus Control Register	0x00000000
0x3109D3D0	MLB0_ACSR0	MLB0 Peripheral Channel Status 0 Register	0x00000000

Table A-90: ADSP-SC58x MLB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109D3D4	MLB0_ACSR1	MLB0 Peripheral Channel Status 1 Register	0x00000000
0x3109D3D8	MLB0_ACMR0	MLB0 Peripheral Channel Mask 0 Register	0x00000000
0x3109D3DC	MLB0_ACMR1	MLB0 Peripheral Channel Mask 1 Register	0x00000000

Table A-91: ADSP-SC58x SCB3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x60205020	SCB3_MST00_SYNC	SCB3 SYNC Mode	0x00000004
0x60205040	SCB3_DCLK0_WR_TIDE	SCB3 DCLK0 Interface Block APB WR Tidemark	0x00000004

Table A-92: ADSP-SC58x MSI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31010000	MSI0_CTL	MSI0 Control Register	0x00000000
0x31010008	MSI0_CLKDIV	MSI0 Clock Divider Register	0x00000000
0x31010010	MSI0_CLKEN	MSI0 Clock Enable Register	0x00000000
0x31010014	MSI0_TMOUT	MSI0 Timeout Register	0xFFFFFF40
0x31010018	MSI0_CTYPE	MSI0 Card Type Register	0x00000000
0x3101001C	MSI0_BLKSIZ	MSI0 Block Size Register	0x00000200
0x31010020	MSI0_BYTCNT	MSI0 Byte Count Register	0x00000200
0x31010024	MSI0_IMSK	MSI0 Interrupt Mask Register	0x00000000
0x31010028	MSI0_CMDARG	MSI0 Command Argument Register	0x00000000
0x3101002C	MSI0_CMD	MSI0 Command Register	0x20000000
0x31010030	MSI0_RESP0	MSI0 Response Register 0	0x00000000
0x31010034	MSI0_RESP1	MSI0 Response Register 1	0x00000000
0x31010038	MSI0_RESP2	MSI0 Response Register 2	0x00000000
0x3101003C	MSI0_RESP3	MSI0 Response Register 3	0x00000000
0x31010040	MSI0_MSKISTAT	MSI0 Masked Interrupt Status Register	0x00000000
0x31010044	MSI0_ISTAT	MSI0 Raw Interrupt Status Register	0x00000000
0x31010048	MSI0_STAT	MSI0 Status Register	0x00000106
0x3101004C	MSI0_FIFOTH	MSI0 FIFO Threshold Watermark Register	0x00FF0000
0x31010050	MSI0_CDETECT	MSI0 Card Detect Register	0x00000001

Table A-92: ADSP-SC58x MSI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3101005C	MSI0_TCBCNT	MSI0 Transferred CIU Card Byte Count Register	0x00000000
0x31010060	MSI0_TBBCNT	MSI0 Transferred Host to BIU-FIFO Byte Count Register	0x00000000
0x31010064	MSI0_DEBNCE	MSI0 Debounce Count Register	0x00FFFFFF
0x31010080	MSI0_BUSMODE	MSI0 Bus Mode Register	0x00000000
0x31010084	MSI0_PLDMND	MSI0 Poll Demand Register	0x00000000
0x31010088	MSI0_DBADDR	MSI0 Descriptor List Base Address Register	0x00000000
0x3101008C	MSI0_IDSTS	MSI0 Internal DMA Status Register	0x00000000
0x31010090	MSI0_IDINTEN	MSI0 Internal DMA Interrupt Enable Register	0x00000000
0x31010094	MSI0_DSCADDR	MSI0 Current Host Descriptor Address Register	0x00000000
0x31010098	MSI0_BUFADDR	MSI0 Current Buffer Descriptor Address Register	0x00000000
0x31010100	MSI0_CDTHRCTL	MSI0 Card Threshold Control Register	0x00000000
0x31010110	MSI0_ENSHIFT	MSI0 Enable Phase Shift Register	0x00000000

Table A-93: ADSP-SC58x OTPC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31011004	OTPC0_STAT	OTPC0 OTP Status Register	0x00000000
0x3101102C	OTPC0_SECU_STATE	OTPC0 OTP Security State Register	0x00000001

Table A-94: ADSP-SC58x PADS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004404	PADS0_PCFG0	PADS0 Peripheral PAD Configuration0 Register	0x00010009
0x31004460	PADS0_DAI0_IE	PADS0 DAI0 Port Input Enable Control Register	0x00000000
0x31004464	PADS0_DAI1_IE	PADS0 DAI1 Port Input Enable Control Register	0x00000000

Table A-95: ADSP-SC58x PCG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9300	PCG0_CTLA0	PCG0 Precision Clock A Control 0 Register	0x00000000
0x310C9304	PCG0_CTLA1	PCG0 Precision Clock A Control 1 Register	0x00000000
0x310C9308	PCG0_CTLB0	PCG0 Precision Clock B Control 0 Register	0x00000000
0x310C930C	PCG0_CTLB1	PCG0 Precision Clock B Control 1 Register	0x00000000

Table A-95: ADSP-SC58x PCG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9310	PCG0_PW1	PCG0 Precision Clock Pulse Width Control 1 Register	0x00000000
0x310C9314	PCG0_SYNC1	PCG0 Precision Clock Frame Sync Synchronization 1 Register	0x00000000
0x310CB300	PCG0_CTLC0	PCG0 Precision Clock C Control 0 Register	0x00000000
0x310CB304	PCG0_CTLC1	PCG0 Precision Clock C Control 1 Register	0x00000000
0x310CB308	PCG0_CTLD0	PCG0 Precision Clock D Control 0 Register	0x00000000
0x310CB30C	PCG0_CTLD1	PCG0 Precision Clock D Control 1 Register	0x00000000
0x310CB310	PCG0_PW2	PCG0 Precision Clock Pulse Width Control 2 Register	0x00000000
0x310CB314	PCG0_SYNC2	PCG0 Precision Clock Frame Sync Synchronization 2 Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8000	PCIE0_EP_ID_[n]	PCIE0 Device ID and Vendor ID Register	0xC58911D4
0x310B8000	PCIE0_RC_ID_[n]	PCIE0 Device ID and Vendor ID Register	0xC58911D4
0x310B8004	PCIE0_RC_STATCMD_[n]	PCIE0 Command and Status Register	0x00100000
0x310B8004	PCIE0_EP_STATCMD_[n]	PCIE0 Command and Status Register	0x00100000
0x310B8008	PCIE0_EP_CCRID_[n]	PCIE0 Class Code and Revision ID Register	0x00000001
0x310B8008	PCIE0_RC_CCRID_[n]	PCIE0 Class Code and Revision ID Register	0x00000001
0x310B800C	PCIE0_EP_CFG_[n]	PCIE0 End Point Configuration Register	0x00000000
0x310B800C	PCIE0_RC_CFG_[n]	PCIE0 Root Complex Configuration Register	0x00010000
0x310B8010	PCIE0_EP_BAR0_[n]	PCIE0 Endpoint Base Address Register 0	0x00000008
0x310B8014	PCIE0_EP_BAR1_[n]	PCIE0 Endpoint Base Address Register 1	0x00000008
0x310B8018	PCIE0_RC_TMRLAT_[n]	PCIE0 Latency Timer Register	0x00000000
0x310B8018	PCIE0_EP_BAR2_[n]	PCIE0 Endpoint Base Address Register 2	0x00000008
0x310B801C	PCIE0_EP_BAR3_[n]	PCIE0 Endpoint Base Address Register 3	0x00000008
0x310B801C	PCIE0_RC_SECSTAT_[n]	PCIE0 Secondary Status and I/O Base and Limit Register	0x00000101
0x310B8020	PCIE0_EP_BAR4_[n]	PCIE0 Endpoint Base Address Register 4	0x00000008
0x310B8020	PCIE0_RC_MBL_[n]	PCIE0 Memory Base and Memory Limit Register	0x00000000
0x310B8024	PCIE0_RC_PREFMBL_[n]	PCIE0 Prefetchable Memory Base and Limit Register	0x00010001
0x310B8024	PCIE0_EP_BAR5_[n]	PCIE0 End Point Base Address Register 5	0x00000008

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8028	PCIE0_RC_PREF_BUPP_[n]	PCIE0 Prefetchable Base Upper 32 Bits Register	0x00000000
0x310B8028	PCIE0_EP_CRDBPTR_[n]	PCIE0 CardBus CIS Pointer Register	0x00000000
0x310B802C	PCIE0_RC_PREF_LMT_UP_[n]	PCIE0 Prefetchable Limit Upper 32 Bits Register	0x00000000
0x310B802C	PCIE0_EP_SSVID_[n]	PCIE0 Subsystem ID and Subsystem Vendor ID Register	0x00000000
0x310B8030	PCIE0_RC_IOBL_[n]	PCIE0 Root Complex I/O Base and Limit Upper 16 bits Register	0x00000000
0x310B8030	PCIE0_EP_ROMCFG_[n]	PCIE0 End Point Expansion ROM Base Address Register	0x00000000
0x310B8034	PCIE0_EP_CAPBPTR_[n]	PCIE0 Capability Pointer Register	0x00000040
0x310B8034	PCIE0_RC_CAPBPTR_[n]	PCIE0 Capability Pointer Register.	0x00000040
0x310B8038	PCIE0_RC_ROMCFG_[n]	PCIE0 Root Complex Expansion ROM Base Address Register	0x00000000
0x310B803C	PCIE0_RC_BRDG_ILPCTL_[n]	PCIE0 Interrupt Bridge Line and Pin Control Register	0x000001FF
0x310B803C	PCIE0_EP_PINLN_INT_[n]	PCIE0 Interrupt Line and Pin Register	0x000001FF
0x310B8040	PCIE0_PMT_CAPB_[n]	PCIE0 Power Management Capabilities Register	0xDBC35001
0x310B8044	PCIE0_PMT_CTLSTAT_[n]	PCIE0 Power Management Control and Status Register	0x00000000
0x310B8050	PCIE0_MSI_CAPB_NPTR_[n]	PCIE0 MSI Capability ID, Next Pointer and Control Register	0x008A7005
0x310B8054	PCIE0_MSI_CAPB_OFF4_[n]	PCIE0 MSI Capability Offset Register	0x00000000
0x310B8058	PCIE0_MSI_CAPB_OFF8_[n]	PCIE0 MSI Capability Offset Register	0x00000000
0x310B8064	PCIE0_MSI_CAPB_OFFC_[n]	PCIE0 MSI Capability Offset Register	0x00000000
0x310B8070	PCIE0_CAPB_NPTR_[n]	PCIE0 PCIe Capabilities, ID, Next Pointer Register	0x00020010
0x310B8074	PCIE0_EP_DEVCAPB_[n]	PCIE0 Device Capabilities Register	0x00008FC1
0x310B8074	PCIE0_RC_DEVCAPB_[n]	PCIE0 Device Capabilities Register	0x00008001
0x310B8078	PCIE0_DEV_CTLSTAT_[n]	PCIE0 Device Control and Status Register	0x00002810
0x310B807C	PCIE0_LNK_CAPB_[n]	PCIE0 Link Capabilities Register	0x0040AC12
0x310B8080	PCIE0_LNK_CTLSTAT_[n]	PCIE0 Link Control and Status Register	0x00110010
0x310B808C	PCIE0_RC_CTLCAPB_[n]	PCIE0 Root Control and Capabilities Register	0x00000000
0x310B8090	PCIE0_RC_STAT_[n]	PCIE0 Root Status Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8094	PCIE0_DEV_CAPB2_[n]	PCIE0 Device Capabilities 2 Register	0x00000000
0x310B8098	PCIE0_DEV_CTLSTAT2_[n]	PCIE0 Device Control 2 and Status 2 Register	0x00000000
0x310B809C	PCIE0_LNK_CAPB2_[n]	PCIE0 Link Capabilities 2 Register	0x00000104
0x310B80A0	PCIE0_LNK_CTLSTAT2_[n]	PCIE0 Link Control 2 and Status 2 Register	0x00010002
0x310B8100	PCIE0_ADVERR_EXTCAPB_[n]	PCIE0 Advanced Error Reporting Extended Capability Header Register	0x00020001
0x310B8104	PCIE0_UNCOR_ERRSTAT_[n]	PCIE0 Uncorrectable Error Status Register	0x00000000
0x310B8108	PCIE0_UNCOR_ERRMSK_[n]	PCIE0 Uncorrectable Error Mask Register	0x00400000
0x310B810C	PCIE0_UNCOR_ERRSEV_[n]	PCIE0 Uncorrectable Error Severity Register	0x00462030
0x310B8110	PCIE0_CORERR_STAT_[n]	PCIE0 Correctable Error Status Register	0x00000000
0x310B8114	PCIE0_CORERR_MSK_[n]	PCIE0 Correctable Error Mask Register	0x00006000
0x310B8118	PCIE0_ADVERR_CAPBCTL_[n]	PCIE0 Advanced Error Capabilities and Control Register	0x00000000
0x310B811C	PCIE0_HDRLOG0_[n]	PCIE0 Header Log Register 0	0x00000000
0x310B8120	PCIE0_HDRLOG1_[n]	PCIE0 Header Log Register 1	0x00000000
0x310B8124	PCIE0_HDRLOG2_[n]	PCIE0 Header Log Register 2	0x00000000
0x310B8128	PCIE0_HDRLOG3_[n]	PCIE0 Header Log Register 3	0x00000000
0x310B812C	PCIE0_RC_ERRCMD_[n]	PCIE0 Root Complex Error Command Register	0x00000000
0x310B8130	PCIE0_RC_ERRSTAT_[n]	PCIE0 Root Error Status Register	0x00000000
0x310B8134	PCIE0_ERRSRC_ID_[n]	PCIE0 Error Source Identification Register	0x00000000
0x310B8700	PCIE0_ACK_TMR_[n]	PCIE0 Acknowledge Latency Timer and Replay Timer Register	0x0C23040B
0x310B8704	PCIE0_VEND_DLLP_[n]	PCIE0 Vendor Specific DLLP Register	0xFFFFFFFF
0x310B8708	PCIE0_PL_FRC_[n]	PCIE0 Port Force Link Register	0x07000004
0x310B870C	PCIE0_ACK_ASPMCTL_[n]	PCIE0 Acknowledge Frequency and L0-L1 ASPM Control Register	0x1B2C2C00
0x310B8710	PCIE0_PLCTL_[n]	PCIE0 Port Logic Link Control Register	0x00000000
0x310B8714	PCIE0_LANE_SKEW_[n]	PCIE0 Lane Skew Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8718	PCIE0_TMRCTL_MFN_[n]	PCIE0 Timer Control and Max Function Number Register	0x00008000
0x310B871C	PCIE0_SYM_TMRFILT1_[n]	PCIE0 Symbol Timer Filter 1 Off	0x00000140
0x310B8720	PCIE0_FILTMSK2_[n]	PCIE0 Filter Mask 2 Register	0x00000000
0x310B8724	PCIE0_BUS_MULOB_DECOMP_[n]	PCIE0 Bus Multiple Outbound Decomposition SubReq Control Register	0x00000001
0x310B8728	PCIE0_PLDBG0_[n]	PCIE0 Port Logic Debug0 Register	0x00000000
0x310B872C	PCIE0_PLDBG1_[n]	PCIE0 Port Logic Debug1 Register	0x00000000
0x310B8730	PCIE0_TX_PST_FCCSTAT_[n]	PCIE0 Transmit Posted FC Credit Status Register	0x00000000
0x310B8734	PCIE0_TX_NPST_FCCSTAT_[n]	PCIE0 Transmit Non-Posted FC Credit Status Register	0x00000000
0x310B8738	PCIE0_TXCOMP_FCCSTAT_[n]	PCIE0 Transmit Completion FC Credit Status Register	0x00000000
0x310B873C	PCIE0_QSTAT_[n]	PCIE0 Queue Status Register	0x00000000
0x310B8740	PCIE0_VC_TXARB1_[n]	PCIE0 Virtual Channel Transmit Arbitration Register 1	0x0000000F
0x310B8744	PCIE0_VC_TXARB2_[n]	PCIE0 Virtual Channel Transmit Arbitration Register 2	0x00000000
0x310B8748	PCIE0_VC0_PST_RXQCTL_[n]	PCIE0 Segmented-Buffer VC0 Posted Receive Queue Control Register	0x4021B050
0x310B874C	PCIE0_VC0_NPST_RXQCTL_[n]	PCIE0 Segmented-Buffer VC0 Non-Posted Receive Queue Control Register	0x0021B006
0x310B8750	PCIE0_VC0_COMP_RXQCTL_[n]	PCIE0 Segmented-Buffer VC0 Completion Receive Queue Control Register	0x00800000
0x310B880C	PCIE0_GEN2_CTL_[n]	PCIE0 Link Width and Speed Change Control Register	0x0002012C
0x310B8810	PCIE0_PL_PHYSTAT_[n]	PCIE0 Port Control PHY Status Register	0x00000000
0x310B8814	PCIE0_PL_PHYCTL_[n]	PCIE0 Port Control PHY Control Register	0x00000000
0x310B8820	PCIE0_MSI_CTL_LADDR_[n]	PCIE0 MSI Controller Lower Address Register	0x00000000
0x310B8824	PCIE0_MSI_CTL_UADDR_[n]	PCIE0 MSI Controller Upper Address Register	0x00000000
0x310B8828	PCIE0_MSI_IEN0_[n]	PCIE0 MSI Controller Interrupt 0 Enable Register	0x00000000
0x310B882C	PCIE0_MSI_IMSK0_[n]	PCIE0 MSI Controller Interrupt 0 Mask Register	0x00000000
0x310B8830	PCIE0_MSI_ISTAT0_[n]	PCIE0 MSI Controller Interrupt 0 Status Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8834	PCIE0_MSI_IEN1_[n]	PCIE0 MSI Controller Interrupt 1 Enable Register	0x00000000
0x310B8838	PCIE0_MSI_IMSK1_[n]	PCIE0 MSI Controller Interrupt 1 Mask Register	0x00000000
0x310B883C	PCIE0_MSI_ISTAT1_[n]	PCIE0 MSI Controller Interrupt 1 Status Register	0x00000000
0x310B8840	PCIE0_MSI_IEN2_[n]	PCIE0 MSI Controller Interrupt 2 Enable Register	0x00000000
0x310B8844	PCIE0_MSI_IMSK2_[n]	PCIE0 MSI Controller Interrupt 2 Mask Register	0x00000000
0x310B8848	PCIE0_MSI_ISTAT2_[n]	PCIE0 MSI Controller Interrupt 2 Status Register	0x00000000
0x310B884C	PCIE0_MSI_IEN3_[n]	PCIE0 MSI Controller Interrupt 3 Enable Register	0x00000000
0x310B8850	PCIE0_MSI_IMSK3_[n]	PCIE0 MSI Controller Interrupt 3 Mask Register	0x00000000
0x310B8854	PCIE0_MSI_ISTAT3_[n]	PCIE0 MSI Controller Interrupt 3 Status Register	0x00000000
0x310B8858	PCIE0_MSI_IEN4_[n]	PCIE0 MSI Controller Interrupt 4 Enable Register	0x00000000
0x310B885C	PCIE0_MSI_IMSK4_[n]	PCIE0 MSI Controller Interrupt 4 Mask Register	0x00000000
0x310B8860	PCIE0_MSI_ISTAT4_[n]	PCIE0 MSI Controller Interrupt 4 Status Register	0x00000000
0x310B8864	PCIE0_MSI_IEN5_[n]	PCIE0 MSI Controller Interrupt 5 Enable Register	0x00000000
0x310B8868	PCIE0_MSI_IMSK5_[n]	PCIE0 MSI Controller Interrupt 5 Mask Register	0x00000000
0x310B886C	PCIE0_MSI_ISTAT5_[n]	PCIE0 MSI Controller Interrupt 5 Status Register	0x00000000
0x310B8870	PCIE0_MSI_IEN6_[n]	PCIE0 MSI Controller Interrupt 6 Enable Register	0x00000000
0x310B8874	PCIE0_MSI_IMSK6_[n]	PCIE0 MSI Controller Interrupt 6 Mask Register	0x00000000
0x310B8878	PCIE0_MSI_ISTAT6_[n]	PCIE0 MSI Controller Interrupt 6 Status Register	0x00000000
0x310B887C	PCIE0_MSI_IEN7_[n]	PCIE0 MSI Controller Interrupt 7 Enable Register	0x00000000
0x310B8880	PCIE0_MSI_IMSK7_[n]	PCIE0 MSI Controller Interrupt 7 Mask Register	0x00000000
0x310B8884	PCIE0_MSI_ISTAT7_[n]	PCIE0 MSI Controller Interrupt 7 Status Register	0x00000000
0x310B8888	PCIE0_MSI_GPIO_IO_[n]	PCIE0 MSI Controller General-Purpose IO Register	0x00000000
0x310B88BC	PCIE0_MISC_CTL1_[n]	PCIE0 DBI Read-Only Write Enable Register	0x00000001
0x310B88D0	PCIE0_BERR_RESP_[n]	PCIE0 Peripheral and SCB Bridge Slave Error Response Register	0x00000000
0x310B88DC	PCIE0_BUSMGR_WDOG_[n]	PCIE0 Bus Order Manager Watchdog Off Register	0x80000FFF
0x310B88E0	PCIE0_CACHE_CTL1_[n]	PCIE0 ACE Cache Coherency Control Register 1	0x00000000
0x310B88E8	PCIE0_CACHE_CTL3_[n]	PCIE0 ACE Cache Coherency Control Register 3	0x00000000
0x310B8900	PCIE0_IATU_VWPRT_[n]	PCIE0 IATU View Port Register	0x00000000
0x310B8904	PCIE0_IATU_CTL1_OUTB_[n]	PCIE0 iATU Region Control 1 Outbound Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8904	PCIE0_IATU_CTL1_INB_[n]	PCIE0 iATU Region Control 1 Inbound Register	0x00000000
0x310B8908	PCIE0_IATU_CTL2_OUTB_[n]	PCIE0 iATU Region Control 2 Register Outbound	0x00000000
0x310B8908	PCIE0_IATU_CTL2_INB_[n]	PCIE0 iATU Region Control 2 Inbound Register	0x00000000
0x310B890C	PCIE0_IATU_LBADDR_INB_[n]	PCIE0 IATU Lower Base Inbound Address Register	0x00000000
0x310B890C	PCIE0_IATU_LBADDR_OUTB_[n]	PCIE0 IATU Lower Base Outbound Address Register	0x00000000
0x310B8910	PCIE0_IATU_UBADDR_OUTB_[n]	PCIE0 IATU Upper Base Address Outbound Register	0x00000000
0x310B8910	PCIE0_IATU_UBADDR_INB_[n]	PCIE0 IATU Upper Base Address Inbound Register	0x00000000
0x310B8914	PCIE0_IATU_LMTADDR_OUTB_[n]	PCIE0 IATU Outbound Limit Address Register	0xFFFFFFFF
0x310B8914	PCIE0_IATU_LMTADDR_INB_[n]	PCIE0 IATU Inbound Limit Address Register	0xFFFFFFFF
0x310B8918	PCIE0_IATU_LTADDR_OUTB_[n]	PCIE0 IATU Lower Target Address Outbound Register	0x00000000
0x310B8918	PCIE0_IATU_LTADDR_INB_[n]	PCIE0 IATU Lower Target Address Inbound Register	0x00000000
0x310B891C	PCIE0_IATU_UTADDR_OUTB_[n]	PCIE0 iATU Upper Target Address Outbound Register	0x00000000
0x310B891C	PCIE0_IATU_UTADDR_INB_[n]	PCIE0 iATU Upper Target Address Inbound Register	0x00000000
0x310B8920	PCIE0_IATU_CTL3_INB_[n]	PCIE0 iATU Region Control 3 Register	0x00000000
0x310B8920	PCIE0_IATU_CTL3_OUTB_[n]	PCIE0 iATU Region Control 3 Register	0x00000000
0x310B8978	PCIE0_DMA_CTL_[n]	PCIE0 DMA Number of Channels Register	0x00010001
0x310B897C	PCIE0_DMAWR_EN_[n]	PCIE0 DMA Write Engine Enable Register	0x00000000
0x310B8980	PCIE0_DMAWR_DRBL_[n]	PCIE0 DMA Write Doorbell Register	0x00000000
0x310B8988	PCIE0_DMAWR_ARBWGT_LO_[n]	PCIE0 DMA Write Channel Arbitration Weight Low Off Register	0x00008421

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8998	PCIE0_DMAWR_PREQ_TMR_[n]	PCIE0 DMA Write Posted Request Deadlock Timer Register	0x0000000E
0x310B899C	PCIE0_DMARD_EN_[n]	PCIE0 DMA Read Engine Enable Register	0x00000000
0x310B89A0	PCIE0_DMARD_DRBL_[n]	PCIE0 DMA Read Doorbell Register	0x00000000
0x310B89A8	PCIE0_DMARD_ARBWGT_LO_[n]	PCIE0 DMA Read Arbitration Weight Low Off Register	0x00008421
0x310B89BC	PCIE0_DMAWR_ISTAT_[n]	PCIE0 DMA Write Interrupt Status Register	0x00000000
0x310B89C4	PCIE0_DMAWR_IMSK_[n]	PCIE0 DMA Write Interrupt Mask Register	0x00FF00FF
0x310B89C8	PCIE0_DMAWR_ICLR_[n]	PCIE0 DMA Write Interrupt Clear Register	0x00000000
0x310B89CC	PCIE0_DMAWR_ERRSTAT_[n]	PCIE0 DMA Write Error Status Register	0x00000000
0x310B89D0	PCIE0_DMAWR_DONE_IMWR_LO_[n]	PCIE0 DMA Write Done IMWr Address Low Register	0x00000000
0x310B89D4	PCIE0_DMAWR_DONE_IMWR_HI_[n]	PCIE0 DMA Write Done IMWr Interrupt Address High Register	0x00000000
0x310B89D8	PCIE0_DMAWR_ABRT_IMWR_LO[n]	PCIE0 DMA Write Abort IMWr Address Low Register	0x00000000
0x310B89DC	PCIE0_DMAWR_ABRT_IMWR_HI[n]	PCIE0 DMA Write Abort IMWr Address High Register	0x00000000
0x310B89E0	PCIE0_DMAWR_CH01_IMWR_[n]	PCIE0 DMA Write Channel 1 and 0 IMWr Data Register	0x00000000
0x310B8A00	PCIE0_DMAWR_LLSTERR_EN_[n]	PCIE0 DMA Write Linked List Error Enable Register	0x00000000
0x310B8A10	PCIE0_DMARD_ISTAT_[n]	PCIE0 DMA Read Interrupt Status Register	0x00000000
0x310B8A18	PCIE0_DMARD_IMSK_[n]	PCIE0 DMA Read Interrupt Mask Register	0x00FF00FF
0x310B8A1C	PCIE0_DMARD_ICLR_[n]	PCIE0 DMA Read Interrupt Clear Register	0x00000000
0x310B8A24	PCIE0_DMARD_ERRSTAT_LO_[n]	PCIE0 DMA Read Error Status Low Register	0x00000000
0x310B8A28	PCIE0_DMARD_ERRSTAT_HI_[n]	PCIE0 DMA Read Error Status High Register	0x00000000
0x310B8A34	PCIE0_DMARD_LLSTERR_EN_[n]	PCIE0 DMA Read Linked List Error Enable Register	0x00000000
0x310B8A3C	PCIE0_DMARD_DONE_IMWR_LO_[n]	PCIE0 DMA Read Done IMWr Address Low Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8A40	PCIE0_DMARD_DONE_IMWR_HI_[n]	PCIE0 DMA Read Done IMWr Address High Register	0x00000000
0x310B8A44	PCIE0_DMARD_IMWRABRT_LO_[n]	PCIE0 DMA Read Abort IMWr Address Low Register	0x00000000
0x310B8A48	PCIE0_DMARD_IMWRABRT_HI_[n]	PCIE0 DMA Read Abort IMWr Address High Register	0x00000000
0x310B8A4C	PCIE0_DMARD_CH01_IMWR_[n]	PCIE0 DMA Read Channel 1 and 0 IMWr Data Register	0x00000000
0x310B8A6C	PCIE0_DMA_VWPRT_SEL_[n]	PCIE0 DMA Channel Context Index Register	0x00000000
0x310B8A70	PCIE0_DMAWR_CTL1_[n]	PCIE0 DMA Control 1 Write Channel Register	0x00000000
0x310B8A70	PCIE0_DMARD_CTL1_[n]	PCIE0 DMA Control 1 Read Channel Register	0x00000000
0x310B8A78	PCIE0_DMARD_XFRSZ_[n]	PCIE0 DMA Transfer Size Read Channel Register	0x00000000
0x310B8A78	PCIE0_DMAWR_XFRSZ_0[n]	PCIE0 DMA Transfer Size Write Channel Register	0x00000000
0x310B8A7C	PCIE0_DMARD_SAR_LO_[n]	PCIE0 DMA SAR Low Read Channel Register	0x00000000
0x310B8A7C	PCIE0_DMAWR_SAR_LO_[n]	PCIE0 DMA SAR Low Write Channel Register	0x00000000
0x310B8A80	PCIE0_DMARD_SAR_HI_[n]	PCIE0 DMA SAR High Read Channel Register	0x00000000
0x310B8A80	PCIE0_DMAWR_SAR_HI_[n]	PCIE0 DMA SAR High Write Channel Register	0x00000000
0x310B8A84	PCIE0_DMARD_DAR_LO_[n]	PCIE0 DMA Destination Address Low Read Channel Register	0x00000000
0x310B8A84	PCIE0_DMAWR_DAR_LO_[n]	PCIE0 DMA DAR Low Write Channel Register	0x00000000
0x310B8A88	PCIE0_DMAWR_DAR_HI_[n]	PCIE0 DMA Dar High Write Channel Register	0x00000000
0x310B8A88	PCIE0_DMARD_DAR_HI_[n]	PCIE0 DMA Destination Address High Read Channel Register	0x00000000
0x310B8A8C	PCIE0_DMAWR_LLP_LO_[n]	PCIE0 DMA LLP Low Write Channel Register	0x00000000
0x310B8A8C	PCIE0_DMARD_LLP_LO_[n]	PCIE0 DMA LLP Low Read Channel Register	0x00000000

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310B8A90	PCIE0_DMARD_LLP_HI_[n]	PCIE0 DMA LLP High Read Channel Register	0x00000000
0x310B8A90	PCIE0_DMAWR_LLP_HI_[n]	PCIE0 DMA LLP High Write Channel Register	0x00000000
0x310B8B40	PCIE0_AUX_CLKFREQ_[n]	PCIE0 Auxiliary Clock Frequency Control Register	0x0000000A
0x310B9010	PCIE0_EP_BAR0_MASK[n]	PCIE0 Endpoint Base Address Mask Register 0	0x00000001
0x310B9014	PCIE0_EP_BAR1_MASK[n]	PCIE0 Endpoint Base Address Mask Register 1	0x00000001
0x310B9018	PCIE0_EP_BAR2_MASK[n]	PCIE0 Endpoint Base Address Mask Register 2	0x00000001
0x310B901C	PCIE0_EP_BAR3_MASK[n]	PCIE0 Endpoint Base Address Mask Register 3	0x00000001
0x310B9020	PCIE0_EP_BAR4_MASK[n]	PCIE0 Endpoint Base Address Mask Register 4	0x00000001
0x310B9024	PCIE0_EP_BAR5_MASK[n]	PCIE0 Endpoint Base Address Mask Register 5	0x00000001
0x310BAC00	PCIE0_APP_CTL	PCIE0 Application Control Register	0x00000000
0x310BAC04	PCIE0_APP_STAT	PCIE0 Application Status Register	0x00000000
0x310BAC08	PCIE0_APP_INTSTAT	PCIE0 Application Interrupts Status Register	0x00000000
0x310BAC14	PCIE0_APP_VNDR_MSG_HDR0	PCIE0 Vendor Message Header Bytes 8 to 11 Register	0x00000000
0x310BAC18	PCIE0_APP_VNDR_MSG_HDR1	PCIE0 Vendor Message Header Bytes 12 to 15 Register	0x00000000
0x310BAC1C	PCIE0_APP_VNDR_MSG	PCIE0 Vendor Message Requester ID Register	0x00000000
0x310BAC28	PCIE0_APP_UNLK_MSG	PCIE0 Unlock Message Requester ID Register	0x00000000
0x310BAC34	PCIE0_APP_CORERR_MSG	PCIE0 Correctable Error Message Requester ID Register	0x00000000
0x310BAC40	PCIE0_APP_NFERR_MSG	PCIE0 Non Fatal Error Message Requester ID Register	0x00000000
0x310BAC4C	PCIE0_APP_FERR_MSG	PCIE0 Fatal Error Message Requester ID Register	0x00000000
0x310BAC58	PCIE0_APP_PMEVT_MSG	PCIE0 Power Management Event Message Requester ID Register	0x00000000
0x310BAC64	PCIE0_APP_PMACK_MSG	PCIE0 Power Management Acknowledge Message Requester ID Register	0x00000000
0x310BAC70	PCIE0_APP_PMTOFF_MSG	PCIE0 Power Management Turn Off Message Requester ID Register	0x00000000
0x310BAC74	PCIE0_APP_DIAGCTL	PCIE0 Diagnostic Control Register	0x00000000
0x310BAC78	PCIE0_APP_DIAGSTAT	PCIE0 Diagnostic Status Register	0x00000000
0x310BB000	PCIE0_RSCK_CTL	PCIE0 Reset and Clock Control Register	0x00000003

Table A-96: ADSP-SC58x PCIE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310BB004	PCIE0_RSCK_STAT	PCIE0 Reset and Clock Status	0x0000001F
0x310BB008	PCIE0_PHY_TXDEEMPH	PCIE0 TX De-emphasis Parameters Register	0x00021618
0x310BB00C	PCIE0_PHY_TXSWING	PCIE0 TX Launch Amplitude Register	0x000039F3

Table A-97: ADSP-SC58x PINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005000	PINT0_MSK_SET	PINT0 PINT Mask Set Register	0x00000000
0x31005004	PINT0_MSK_CLR	PINT0 PINT Mask Clear Register	0x00000000
0x31005008	PINT0_REQ	PINT0 PINT Request Register	0x00000000
0x3100500C	PINT0_ASSIGN	PINT0 PINT Assign Register	0x00000101
0x31005010	PINT0_EDGE_SET	PINT0 PINT Edge Set Register	0x00000000
0x31005014	PINT0_EDGE_CLR	PINT0 PINT Edge Clear Register	0x00000000
0x31005018	PINT0_INV_SET	PINT0 PINT Invert Set Register	0x00000000
0x3100501C	PINT0_INV_CLR	PINT0 PINT Invert Clear Register	0x00000000
0x31005020	PINT0_PINSTATE	PINT0 PINT Pin State Register	0x00000000
0x31005024	PINT0_LATCH	PINT0 PINT Latch Register	0x00000000

Table A-98: ADSP-SC58x PINT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005100	PINT1_MSK_SET	PINT1 PINT Mask Set Register	0x00000000
0x31005104	PINT1_MSK_CLR	PINT1 PINT Mask Clear Register	0x00000000
0x31005108	PINT1_REQ	PINT1 PINT Request Register	0x00000000
0x3100510C	PINT1_ASSIGN	PINT1 PINT Assign Register	0x00000101
0x31005110	PINT1_EDGE_SET	PINT1 PINT Edge Set Register	0x00000000
0x31005114	PINT1_EDGE_CLR	PINT1 PINT Edge Clear Register	0x00000000
0x31005118	PINT1_INV_SET	PINT1 PINT Invert Set Register	0x00000000
0x3100511C	PINT1_INV_CLR	PINT1 PINT Invert Clear Register	0x00000000
0x31005120	PINT1_PINSTATE	PINT1 PINT Pin State Register	0x00000000
0x31005124	PINT1_LATCH	PINT1 PINT Latch Register	0x00000000

Table A-99: ADSP-SC58x PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005200	PINT2_MSK_SET	PINT2 PINT Mask Set Register	0x00000000
0x31005204	PINT2_MSK_CLR	PINT2 PINT Mask Clear Register	0x00000000
0x31005208	PINT2_REQ	PINT2 PINT Request Register	0x00000000
0x3100520C	PINT2_ASSIGN	PINT2 PINT Assign Register	0x00000101
0x31005210	PINT2_EDGE_SET	PINT2 PINT Edge Set Register	0x00000000
0x31005214	PINT2_EDGE_CLR	PINT2 PINT Edge Clear Register	0x00000000
0x31005218	PINT2_INV_SET	PINT2 PINT Invert Set Register	0x00000000
0x3100521C	PINT2_INV_CLR	PINT2 PINT Invert Clear Register	0x00000000
0x31005220	PINT2_PINSTATE	PINT2 PINT Pin State Register	0x00000000
0x31005224	PINT2_LATCH	PINT2 PINT Latch Register	0x00000000

Table A-100: ADSP-SC58x PINT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005300	PINT3_MSK_SET	PINT3 PINT Mask Set Register	0x00000000
0x31005304	PINT3_MSK_CLR	PINT3 PINT Mask Clear Register	0x00000000
0x31005308	PINT3_REQ	PINT3 PINT Request Register	0x00000000
0x3100530C	PINT3_ASSIGN	PINT3 PINT Assign Register	0x00000101
0x31005310	PINT3_EDGE_SET	PINT3 PINT Edge Set Register	0x00000000
0x31005314	PINT3_EDGE_CLR	PINT3 PINT Edge Clear Register	0x00000000
0x31005318	PINT3_INV_SET	PINT3 PINT Invert Set Register	0x00000000
0x3100531C	PINT3_INV_CLR	PINT3 PINT Invert Clear Register	0x00000000
0x31005320	PINT3_PINSTATE	PINT3 PINT Pin State Register	0x00000000
0x31005324	PINT3_LATCH	PINT3 PINT Latch Register	0x00000000

Table A-101: ADSP-SC58x PINT4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005400	PINT4_MSK_SET	PINT4 PINT Mask Set Register	0x00000000
0x31005404	PINT4_MSK_CLR	PINT4 PINT Mask Clear Register	0x00000000
0x31005408	PINT4_REQ	PINT4 PINT Request Register	0x00000000
0x3100540C	PINT4_ASSIGN	PINT4 PINT Assign Register	0x00000101

Table A-101: ADSP-SC58x PINT4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31005410	PINT4_EDGE_SET	PINT4 PINT Edge Set Register	0x00000000
0x31005414	PINT4_EDGE_CLR	PINT4 PINT Edge Clear Register	0x00000000
0x31005418	PINT4_INV_SET	PINT4 PINT Invert Set Register	0x00000000
0x3100541C	PINT4_INV_CLR	PINT4 PINT Invert Clear Register	0x00000000
0x31005420	PINT4_PINSTATE	PINT4 PINT Pin State Register	0x00000000
0x31005424	PINT4_LATCH	PINT4 PINT Latch Register	0x00000000

Table A-102: ADSP-SC58x PINT5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31005500	PINT5_MSK_SET	PINT5 PINT Mask Set Register	0x00000000
0x31005504	PINT5_MSK_CLR	PINT5 PINT Mask Clear Register	0x00000000
0x31005508	PINT5_REQ	PINT5 PINT Request Register	0x00000000
0x3100550C	PINT5_ASSIGN	PINT5 PINT Assign Register	0x00000101
0x31005510	PINT5_EDGE_SET	PINT5 PINT Edge Set Register	0x00000000
0x31005514	PINT5_EDGE_CLR	PINT5 PINT Edge Clear Register	0x00000000
0x31005518	PINT5_INV_SET	PINT5 PINT Invert Set Register	0x00000000
0x3100551C	PINT5_INV_CLR	PINT5 PINT Invert Clear Register	0x00000000
0x31005520	PINT5_PINSTATE	PINT5 PINT Pin State Register	0x00000000
0x31005524	PINT5_LATCH	PINT5 PINT Latch Register	0x00000000

Table A-103: ADSP-SC58x PKA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D4000	PKA0_APTR	PKA0 PKA Vector_A Address	0x00000000
0x310D4004	PKA0_BPTR	PKA0 PKA Vector_B Address	0x00000000
0x310D4008	PKA0_CPTR	PKA0 PKA Vector_C Address	0x00000000
0x310D400C	PKA0_DPTR	PKA0 PKA Vector_D Address	0x00000000
0x310D4010	PKA0_ALEN	PKA0 PKA Vector_A Length	0x00000000
0x310D4014	PKA0_BLEN	PKA0 PKA Vector_B Length	0x00000000
0x310D4018	PKA0_SHIFT	PKA0 PKA Bit Shift Value	0x00000000
0x310D401C	PKA0_FUNC	PKA0 PKA Function	0x00000000

Table A-103: ADSP-SC58x PKA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310D4020	PKA0_COMPARE	PKA0 PKA Compare Result	0x00000001
0x310D4024	PKA0_RESULTMSW	PKA0 PKA Most-Significant-Word of Result Vector	0x00008000
0x310D4028	PKA0_DIVMSW	PKA0 PKA Most-Significant-Word of Divide Remainder	0x00008000
0x310D6000	PKA0_RAM	PKA0 Start of PKA RAM space	0x00000000

Table A-104: ADSP-SC58x PKIC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D8000	PKIC0_POL_CTL	PKIC0 Polarity Control Register	0x00000000
0x310D8004	PKIC0_TYPE_CTL	PKIC0 Type Control Register	0x00000000
0x310D8008	PKIC0_EN_CTL	PKIC0 Enable Control Register	0x00000000
0x310D800C	PKIC0_RAW_STAT	PKIC0 Raw Status Register	0x00000000
0x310D800C	PKIC0_EN_SET	PKIC0 Enable Set Register	0x00000000
0x310D8010	PKIC0_ACK	PKIC0 Acknowledge Register	0x00000000
0x310D8010	PKIC0_EN_STAT	PKIC0 Enabled Status Register	0x00000000
0x310D8014	PKIC0_EN_CLR	PKIC0 Enable Clear Register	0x00000000

Table A-105: ADSP-SC58x PKTE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD000	PKTE0_CTL_STAT	PKTE0 Packet Engine Control Register	0x00000002
0x310CD004	PKTE0_SRC_ADDR	PKTE0 Packet Engine Source Address	0x00000000
0x310CD008	PKTE0_DEST_ADDR	PKTE0 Packet Engine Destination Address	0x00000000
0x310CD00C	PKTE0_SA_ADDR	PKTE0 Packet Engine SA Address	0x00000000
0x310CD010	PKTE0_STATE_ADDR	PKTE0 Packet Engine State Record Address	0x00000000
0x310CD014	PKTE0_ARC4STATE_ADDR	PKTE0 Packet Engine ARC4 State Record Address	0x00000000
0x310CD018	PKTE0_USERID	PKTE0 Packet Engine User ID	0x00000000
0x310CD01C	PKTE0_LEN	PKTE0 Packet Engine Length Register	0x00800000
0x310CD080	PKTE0_CDRBASE_ADDR	PKTE0 Packet Engine Command Descriptor Ring Base Address	0x00000000
0x310CD084	PKTE0_RDRBASE_ADDR	PKTE0 Packet Engine Result Descriptor Ring Base Address	0x00000000

Table A-105: ADSP-SC58x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD088	PKTE0_RING_CFG	PKTE0 Packet Engine Ring Configuration	0x00000000
0x310CD08C	PKTE0_RING_THRESH	PKTE0 Packet Engine Ring Threshold Registers	0x00000000
0x310CD090	PKTE0_CDSC_INCR	PKTE0 Packet Engine Command Descriptor Count Increment Register	0x00000000
0x310CD090	PKTE0_CDSC_CNT	PKTE0 Packet Engine Command Descriptor Count Register	0x00000000
0x310CD094	PKTE0_RDSC_CNT	PKTE0 Packet Engine Result Descriptor Count Registers	0x00000000
0x310CD094	PKTE0_RDSC_DECR	PKTE0 Packet Engine Result Descriptor Count Decrement Registers	0x00000000
0x310CD098	PKTE0_RING_PTR	PKTE0 Packet Engine Ring Pointer Status	0x00000000
0x310CD09C	PKTE0_RING_STAT	PKTE0 Packet Engine Ring Status	0x00000000
0x310CD100	PKTE0_CFG	PKTE0 Packet Engine Configuration Register	0x00000000
0x310CD104	PKTE0_STAT	PKTE0 Packet Engine Status Register	0x00040402
0x310CD10C	PKTE0_BUF_THRESH	PKTE0 Packet Engine Buffer Threshold Register	0x00800080
0x310CD110	PKTE0_INBUF_INCR	PKTE0 Packet Engine Input Buffer Count Increment Register	0x00000000
0x310CD110	PKTE0_INBUF_CNT	PKTE0 Packet Engine Input Buffer Count Register	0x00000000
0x310CD114	PKTE0_OUTBUF_CNT	PKTE0 Packet Engine Output Buffer Count Register	0x00000000
0x310CD114	PKTE0_OUTBUF_DECR	PKTE0 Packet Engine Output Buffer Count Decrement Register	0x00000000
0x310CD118	PKTE0_BUF_PTR	PKTE0 Packet Engine Buffer Pointer Register	0x00000000
0x310CD120	PKTE0_DMA_CFG	PKTE0 Packet Engine DMA Configuration Register	0x00180006
0x310CD1D0	PKTE0_ENDIAN_CFG	PKTE0 Packet Engine Endian Configuration Register	0x00E400E4
0x310CD1E0	PKTE0_HLT_CTL	PKTE0 Packet Engine Halt Control Register	0x00000000
0x310CD1E0	PKTE0_HLT_STAT	PKTE0 Packet Engine Halt Status Register	0x00000000
0x310CD1E4	PKTE0_CONT	PKTE0 PKTE Continue Register	0x00000000
0x310CD1E8	PKTE0_CLK_CTL	PKTE0 PE Clock Control Register	0x0000001F
0x310CD200	PKTE0_IUMSK_STAT	PKTE0 Interrupt Unmasked Status Register	0x00000000
0x310CD204	PKTE0_IMSK_STAT	PKTE0 Interrupt Masked Status Register	0x00000000
0x310CD204	PKTE0_INT_CLR	PKTE0 Interrupt Clear Register	0x00000000
0x310CD208	PKTE0_INT_EN	PKTE0 Interrupt Enable Register	0x00000000
0x310CD20C	PKTE0_INT_CFG	PKTE0 Interrupt Configuration Register	0x00000000

Table A-105: ADSP-SC58x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD210	PKTE0_IMSK_EN	PKTE0 Interrupt Mask Enable Register	0x00000000
0x310CD214	PKTE0_IMSK_DIS	PKTE0 Interrupt Mask Disable Register	0x00000000
0x310CD400	PKTE0_SA_CMD0	PKTE0 SA Command 0	0x00000000
0x310CD404	PKTE0_SA_CMD1	PKTE0 SA Command 1	0x00000000
0x310CD408	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD40C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD410	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD414	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD418	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD41C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD420	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD424	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x310CD428	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD42C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD430	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD434	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD438	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD43C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD440	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD444	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x310CD448	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD44C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD450	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD454	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD458	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD45C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD460	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD464	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x310CD468	PKTE0_SA_SPI	PKTE0 SA SPI Register	0x00000000
0x310CD46C	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000
0x310CD470	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000

Table A-105: ADSP-SC58x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CD474	PKTE0_SA_SEQNUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x310CD478	PKTE0_SA_SEQNUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x310CD47C	PKTE0_SA_ARC4IJPTR	PKTE0 ARC4 i and j Pointer Register	0x00000000
0x310CD47C	PKTE0_SA_RDY	PKTE0 SA Ready Indicator	0x00000000
0x310CD47C	PKTE0_SA_NONCE	PKTE0 SA Initialization Vector Register	0x00000000
0x310CD500	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD504	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD508	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD50C	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x310CD510	PKTE0_STATE_BYTE_CNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x310CD514	PKTE0_STATE_BYTE_CNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x310CD518	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD51C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD520	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD524	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD528	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD52C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD530	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD534	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x310CD700	PKTE0_ARC4STATE_BUF	PKTE0 Starting Entry of 256-byte ARC4 State Buffer	0x00000000
0x310CD800	PKTE0_DATAIO_BUF	PKTE0 Starting Entry of 256-byte Data Input/Output Buffer	0x00000000

Table A-106: ADSP-SC58x PORTA MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004000	PORTA_FER	PORTA Port x Function Enable Register	0x00000000
0x31004004	PORTA_FER_SET	PORTA Port x Function Enable Set Register	0x00000000
0x31004008	PORTA_FER_CLR	PORTA Port x Function Enable Clear Register	0x00000000

Table A-106: ADSP-SC58x PORTA MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100400C	PORTA_DATA	PORTA Port x GPIO Data Register	0x00000000
0x31004010	PORTA_DATA_SET	PORTA Port x GPIO Data Set Register	0x00000000
0x31004014	PORTA_DATA_CLR	PORTA Port x GPIO Data Clear Register	0x00000000
0x31004018	PORTA_DIR	PORTA Port x GPIO Direction Register	0x00000000
0x3100401C	PORTA_DIR_SET	PORTA Port x GPIO Direction Set Register	0x00000000
0x31004020	PORTA_DIR_CLR	PORTA Port x GPIO Direction Clear Register	0x00000000
0x31004024	PORTA_INEN	PORTA Port x GPIO Input Enable Register	0x00000000
0x31004028	PORTA_INEN_SET	PORTA Port x GPIO Input Enable Set Register	0x00000000
0x3100402C	PORTA_INEN_CLR	PORTA Port x GPIO Input Enable Clear Register	0x00000000
0x31004030	PORTA_MUX	PORTA Port x Multiplexer Control Register	0x00000000
0x31004034	PORTA_DATA_TGL	PORTA Port x GPIO Output Toggle Register	0x00000000
0x31004038	PORTA_POL	PORTA Port x GPIO Polarity Invert Register	0x00000000
0x3100403C	PORTA_POL_SET	PORTA Port x GPIO Polarity Invert Set Register	0x00000000
0x31004040	PORTA_POL_CLR	PORTA Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004044	PORTA_LOCK	PORTA Port x GPIO Lock Register	0x00000000

Table A-107: ADSP-SC58x PORTB MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004080	PORTB_FER	PORTB Port x Function Enable Register	0x00000000
0x31004084	PORTB_FER_SET	PORTB Port x Function Enable Set Register	0x00000000
0x31004088	PORTB_FER_CLR	PORTB Port x Function Enable Clear Register	0x00000000
0x3100408C	PORTB_DATA	PORTB Port x GPIO Data Register	0x00000000
0x31004090	PORTB_DATA_SET	PORTB Port x GPIO Data Set Register	0x00000000
0x31004094	PORTB_DATA_CLR	PORTB Port x GPIO Data Clear Register	0x00000000
0x31004098	PORTB_DIR	PORTB Port x GPIO Direction Register	0x00000000
0x3100409C	PORTB_DIR_SET	PORTB Port x GPIO Direction Set Register	0x00000000
0x310040A0	PORTB_DIR_CLR	PORTB Port x GPIO Direction Clear Register	0x00000000
0x310040A4	PORTB_INEN	PORTB Port x GPIO Input Enable Register	0x00000000
0x310040A8	PORTB_INEN_SET	PORTB Port x GPIO Input Enable Set Register	0x00000000
0x310040AC	PORTB_INEN_CLR	PORTB Port x GPIO Input Enable Clear Register	0x00000000

Table A-107: ADSP-SC58x PORTB MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310040B0	PORTB_MUX	PORTB Port x Multiplexer Control Register	0x00000000
0x310040B4	PORTB_DATA_TGL	PORTB Port x GPIO Output Toggle Register	0x00000000
0x310040B8	PORTB_POL	PORTB Port x GPIO Polarity Invert Register	0x00000000
0x310040BC	PORTB_POL_SET	PORTB Port x GPIO Polarity Invert Set Register	0x00000000
0x310040C0	PORTB_POL_CLR	PORTB Port x GPIO Polarity Invert Clear Register	0x00000000
0x310040C4	PORTB_LOCK	PORTB Port x GPIO Lock Register	0x00000000

Table A-108: ADSP-SC58x PORTC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004100	PORTC_FER	PORTC Port x Function Enable Register	0x00000000
0x31004104	PORTC_FER_SET	PORTC Port x Function Enable Set Register	0x00000000
0x31004108	PORTC_FER_CLR	PORTC Port x Function Enable Clear Register	0x00000000
0x3100410C	PORTC_DATA	PORTC Port x GPIO Data Register	0x00000000
0x31004110	PORTC_DATA_SET	PORTC Port x GPIO Data Set Register	0x00000000
0x31004114	PORTC_DATA_CLR	PORTC Port x GPIO Data Clear Register	0x00000000
0x31004118	PORTC_DIR	PORTC Port x GPIO Direction Register	0x00000000
0x3100411C	PORTC_DIR_SET	PORTC Port x GPIO Direction Set Register	0x00000000
0x31004120	PORTC_DIR_CLR	PORTC Port x GPIO Direction Clear Register	0x00000000
0x31004124	PORTC_INEN	PORTC Port x GPIO Input Enable Register	0x00000000
0x31004128	PORTC_INEN_SET	PORTC Port x GPIO Input Enable Set Register	0x00000000
0x3100412C	PORTC_INEN_CLR	PORTC Port x GPIO Input Enable Clear Register	0x00000000
0x31004130	PORTC_MUX	PORTC Port x Multiplexer Control Register	0x00000000
0x31004134	PORTC_DATA_TGL	PORTC Port x GPIO Output Toggle Register	0x00000000
0x31004138	PORTC_POL	PORTC Port x GPIO Polarity Invert Register	0x00000000
0x3100413C	PORTC_POL_SET	PORTC Port x GPIO Polarity Invert Set Register	0x00000000
0x31004140	PORTC_POL_CLR	PORTC Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004144	PORTC_LOCK	PORTC Port x GPIO Lock Register	0x00000000

Table A-109: ADSP-SC58x PORTD MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004180	PORTD_FER	PORTD Port x Function Enable Register	0x00000000
0x31004184	PORTD_FER_SET	PORTD Port x Function Enable Set Register	0x00000000
0x31004188	PORTD_FER_CLR	PORTD Port x Function Enable Clear Register	0x00000000
0x3100418C	PORTD_DATA	PORTD Port x GPIO Data Register	0x00000000
0x31004190	PORTD_DATA_SET	PORTD Port x GPIO Data Set Register	0x00000000
0x31004194	PORTD_DATA_CLR	PORTD Port x GPIO Data Clear Register	0x00000000
0x31004198	PORTD_DIR	PORTD Port x GPIO Direction Register	0x00000000
0x3100419C	PORTD_DIR_SET	PORTD Port x GPIO Direction Set Register	0x00000000
0x310041A0	PORTD_DIR_CLR	PORTD Port x GPIO Direction Clear Register	0x00000000
0x310041A4	PORTD_INEN	PORTD Port x GPIO Input Enable Register	0x00000000
0x310041A8	PORTD_INEN_SET	PORTD Port x GPIO Input Enable Set Register	0x00000000
0x310041AC	PORTD_INEN_CLR	PORTD Port x GPIO Input Enable Clear Register	0x00000000
0x310041B0	PORTD_MUX	PORTD Port x Multiplexer Control Register	0x00000000
0x310041B4	PORTD_DATA_TGL	PORTD Port x GPIO Output Toggle Register	0x00000000
0x310041B8	PORTD_POL	PORTD Port x GPIO Polarity Invert Register	0x00000000
0x310041BC	PORTD_POL_SET	PORTD Port x GPIO Polarity Invert Set Register	0x00000000
0x310041C0	PORTD_POL_CLR	PORTD Port x GPIO Polarity Invert Clear Register	0x00000000
0x310041C4	PORTD_LOCK	PORTD Port x GPIO Lock Register	0x00000000

Table A-110: ADSP-SC58x PORTE MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004200	PORTE_FER	PORTE Port x Function Enable Register	0x00000000
0x31004204	PORTE_FER_SET	PORTE Port x Function Enable Set Register	0x00000000
0x31004208	PORTE_FER_CLR	PORTE Port x Function Enable Clear Register	0x00000000
0x3100420C	PORTE_DATA	PORTE Port x GPIO Data Register	0x00000000
0x31004210	PORTE_DATA_SET	PORTE Port x GPIO Data Set Register	0x00000000
0x31004214	PORTE_DATA_CLR	PORTE Port x GPIO Data Clear Register	0x00000000
0x31004218	PORTE_DIR	PORTE Port x GPIO Direction Register	0x00000000
0x3100421C	PORTE_DIR_SET	PORTE Port x GPIO Direction Set Register	0x00000000
0x31004220	PORTE_DIR_CLR	PORTE Port x GPIO Direction Clear Register	0x00000000

Table A-110: ADSP-SC58x PORTE MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31004224	PORTE_INEN	PORTE Port x GPIO Input Enable Register	0x00000000
0x31004228	PORTE_INEN_SET	PORTE Port x GPIO Input Enable Set Register	0x00000000
0x3100422C	PORTE_INEN_CLR	PORTE Port x GPIO Input Enable Clear Register	0x00000000
0x31004230	PORTE_MUX	PORTE Port x Multiplexer Control Register	0x00000000
0x31004234	PORTE_DATA_TGL	PORTE Port x GPIO Output Toggle Register	0x00000000
0x31004238	PORTE_POL	PORTE Port x GPIO Polarity Invert Register	0x00000000
0x3100423C	PORTE_POL_SET	PORTE Port x GPIO Polarity Invert Set Register	0x00000000
0x31004240	PORTE_POL_CLR	PORTE Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004244	PORTE_LOCK	PORTE Port x GPIO Lock Register	0x00000000

Table A-111: ADSP-SC58x PORTF MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004280	PORTF_FER	PORTF Port x Function Enable Register	0x00000000
0x31004284	PORTF_FER_SET	PORTF Port x Function Enable Set Register	0x00000000
0x31004288	PORTF_FER_CLR	PORTF Port x Function Enable Clear Register	0x00000000
0x3100428C	PORTF_DATA	PORTF Port x GPIO Data Register	0x00000000
0x31004290	PORTF_DATA_SET	PORTF Port x GPIO Data Set Register	0x00000000
0x31004294	PORTF_DATA_CLR	PORTF Port x GPIO Data Clear Register	0x00000000
0x31004298	PORTF_DIR	PORTF Port x GPIO Direction Register	0x00000000
0x3100429C	PORTF_DIR_SET	PORTF Port x GPIO Direction Set Register	0x00000000
0x310042A0	PORTF_DIR_CLR	PORTF Port x GPIO Direction Clear Register	0x00000000
0x310042A4	PORTF_INEN	PORTF Port x GPIO Input Enable Register	0x00000000
0x310042A8	PORTF_INEN_SET	PORTF Port x GPIO Input Enable Set Register	0x00000000
0x310042AC	PORTF_INEN_CLR	PORTF Port x GPIO Input Enable Clear Register	0x00000000
0x310042B0	PORTF_MUX	PORTF Port x Multiplexer Control Register	0x00000000
0x310042B4	PORTF_DATA_TGL	PORTF Port x GPIO Output Toggle Register	0x00000000
0x310042B8	PORTF_POL	PORTF Port x GPIO Polarity Invert Register	0x00000000
0x310042BC	PORTF_POL_SET	PORTF Port x GPIO Polarity Invert Set Register	0x00000000
0x310042C0	PORTF_POL_CLR	PORTF Port x GPIO Polarity Invert Clear Register	0x00000000
0x310042C4	PORTF_LOCK	PORTF Port x GPIO Lock Register	0x00000000

Table A-112: ADSP-SC58x PORTG MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31004300	PORTG_FER	PORTG Port x Function Enable Register	0x00000000
0x31004304	PORTG_FER_SET	PORTG Port x Function Enable Set Register	0x00000000
0x31004308	PORTG_FER_CLR	PORTG Port x Function Enable Clear Register	0x00000000
0x3100430C	PORTG_DATA	PORTG Port x GPIO Data Register	0x00000000
0x31004310	PORTG_DATA_SET	PORTG Port x GPIO Data Set Register	0x00000000
0x31004314	PORTG_DATA_CLR	PORTG Port x GPIO Data Clear Register	0x00000000
0x31004318	PORTG_DIR	PORTG Port x GPIO Direction Register	0x00000000
0x3100431C	PORTG_DIR_SET	PORTG Port x GPIO Direction Set Register	0x00000000
0x31004320	PORTG_DIR_CLR	PORTG Port x GPIO Direction Clear Register	0x00000000
0x31004324	PORTG_INEN	PORTG Port x GPIO Input Enable Register	0x00000000
0x31004328	PORTG_INEN_SET	PORTG Port x GPIO Input Enable Set Register	0x00000000
0x3100432C	PORTG_INEN_CLR	PORTG Port x GPIO Input Enable Clear Register	0x00000000
0x31004330	PORTG_MUX	PORTG Port x Multiplexer Control Register	0x00000000
0x31004334	PORTG_DATA_TGL	PORTG Port x GPIO Output Toggle Register	0x00000000
0x31004338	PORTG_POL	PORTG Port x GPIO Polarity Invert Register	0x00000000
0x3100433C	PORTG_POL_SET	PORTG Port x GPIO Polarity Invert Set Register	0x00000000
0x31004340	PORTG_POL_CLR	PORTG Port x GPIO Polarity Invert Clear Register	0x00000000
0x31004344	PORTG_LOCK	PORTG Port x GPIO Lock Register	0x00000000

Table A-113: ADSP-SC58x PWM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A000	PWM0_CTL	PWM0 Control Register	0x00020000
0x3100A004	PWM0_CHANCFG	PWM0 Channel Configuration Register	0x00000000
0x3100A008	PWM0_TRIPCFG	PWM0 Trip Configuration Register	0x00000000
0x3100A00C	PWM0_STAT	PWM0 Status Register	0x00000000
0x3100A010	PWM0_IMSK	PWM0 Interrupt Mask Register	0x00000000
0x3100A014	PWM0_ILAT	PWM0 Interrupt Latch Register	0x00000000
0x3100A018	PWM0_CHOPCFG	PWM0 Chop Configuration Register	0x00000000
0x3100A020	PWM0_SYNC_WID	PWM0 Sync Pulse Width Register	0x000003FF
0x3100A024	PWM0_TM0	PWM0 Timer 0 Period Register	0x00000000

Table A-113: ADSP-SC58x PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A028	PWM0_TM1	PWM0 Timer 1 Period Register	0x00000000
0x3100A02C	PWM0_TM2	PWM0 Timer 2 Period Register	0x00000000
0x3100A030	PWM0_TM3	PWM0 Timer 3 Period Register	0x00000000
0x3100A034	PWM0_TM4	PWM0 Timer 4 Period Register	0x00000000
0x3100A038	PWM0_DLYA	PWM0 Channel A Delay Register	0x00000000
0x3100A03C	PWM0_DLYB	PWM0 Channel B Delay Register	0x00000000
0x3100A040	PWM0_DLYC	PWM0 Channel C Delay Register	0x00000000
0x3100A044	PWM0_DLYD	PWM0 Channel D Delay Register	0x00000000
0x3100A048	PWM0_ACTL	PWM0 Channel A Control Register	0x00000000
0x3100A04C	PWM0_AH0	PWM0 Channel A-High Duty-0 Register	0x00000000
0x3100A050	PWM0_AH1	PWM0 Channel A-High Duty-1 Register	0x00000000
0x3100A054	PWM0_AH0_HP	PWM0 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A058	PWM0_AH1_HP	PWM0 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A05C	PWM0_AL0	PWM0 Channel A-Low Duty-0 Register	0x00000000
0x3100A060	PWM0_AL1	PWM0 Channel A-Low Duty-1 Register	0x00000000
0x3100A064	PWM0_AL0_HP	PWM0 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A068	PWM0_AL1_HP	PWM0 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A06C	PWM0_BCTL	PWM0 Channel B Control Register	0x00000000
0x3100A070	PWM0_BH0	PWM0 Channel B-High Duty-0 Register	0x00000000
0x3100A074	PWM0_BH1	PWM0 Channel B-High Duty-1 Register	0x00000000
0x3100A078	PWM0_BH0_HP	PWM0 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A07C	PWM0_BH1_HP	PWM0 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A080	PWM0_BL0	PWM0 Channel B-Low Duty-0 Register	0x00000000
0x3100A084	PWM0_BL1	PWM0 Channel B-Low Duty-1 Register	0x00000000
0x3100A088	PWM0_BL0_HP	PWM0 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000

Table A-113: ADSP-SC58x PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A08C	PWM0_BL1_HP	PWM0 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A090	PWM0_CCTL	PWM0 Channel C Control Register	0x00000000
0x3100A094	PWM0_CH0	PWM0 Channel C-High Pulse Duty Register 0	0x00000000
0x3100A098	PWM0_CH1	PWM0 Channel C-High Pulse Duty Register 1	0x00000000
0x3100A09C	PWM0_CH0_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A0A0	PWM0_CH1_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A0A4	PWM0_CL0	PWM0 Channel C-Low Pulse Duty Register 0	0x00000000
0x3100A0A8	PWM0_CL1	PWM0 Channel C-Low Duty-1 Register	0x00000000
0x3100A0AC	PWM0_CL0_HP	PWM0 Channel C-Low Pulse Duty Register 1	0x00000000
0x3100A0B0	PWM0_CL1_HP	PWM0 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A0B4	PWM0_DCTL	PWM0 Channel D Control Register	0x00000000
0x3100A0B8	PWM0_DH0	PWM0 Channel D-High Duty-0 Register	0x00000000
0x3100A0BC	PWM0_DH1	PWM0 Channel D-High Pulse Duty Register 1	0x00000000
0x3100A0C0	PWM0_DH0_HP	PWM0 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A0C4	PWM0_DH1_HP	PWM0 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A0C8	PWM0_DL0	PWM0 Channel D-Low Pulse Duty Register 0	0x00000000
0x3100A0CC	PWM0_DL1	PWM0 Channel D-Low Pulse Duty Register 1	0x00000000
0x3100A0D0	PWM0_DL0_HP	PWM0 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A0D4	PWM0_DL1_HP	PWM0 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A0D8	PWM0_AH_DUTY0	PWM0 Channel A-High Full Duty0 Register	0x00000000
0x3100A0DC	PWM0_AH_DUTY1	PWM0 Channel A-High Full Duty1 Register	0x00000000
0x3100A0E0	PWM0_AL_DUTY0	PWM0 Channel A-Low Full Duty0 Register	0x00000000
0x3100A0E4	PWM0_AL_DUTY1	PWM0 Channel A-Low Full Duty1 Register	0x00000000
0x3100A0E8	PWM0_BH_DUTY0	PWM0 Channel B-High Full Duty0 Register	0x00000000
0x3100A0EC	PWM0_BH_DUTY1	PWM0 Channel B-High Full Duty1 Register	0x00000000

Table A-113: ADSP-SC58x PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A0F0	PWM0_BL_DUTY0	PWM0 Channel B-Low Full Duty0 Register	0x00000000
0x3100A0F4	PWM0_BL_DUTY1	PWM0 Channel B-Low Full Duty1 Register	0x00000000
0x3100A0F8	PWM0_CH_DUTY0	PWM0 Channel C-High Full Duty0 Register	0x00000000
0x3100A0FC	PWM0_CH_DUTY1	PWM0 Channel C-High Full Duty1 Register	0x00000000
0x3100A100	PWM0_CL_DUTY0	PWM0 Channel C-Low Full Duty0 Register	0x00000000
0x3100A104	PWM0_CL_DUTY1	PWM0 Channel C-Low Full Duty1 Register	0x00000000
0x3100A108	PWM0_DH_DUTY0	PWM0 Channel D-High Full Duty0 Register	0x00000000
0x3100A10C	PWM0_DH_DUTY1	PWM0 Channel D-High Full Duty1 Register	0x00000000
0x3100A110	PWM0_DL_DUTY0	PWM0 Channel D-Low Full Duty0 Register	0x00000000
0x3100A114	PWM0_DL_DUTY1	PWM0 Channel D-Low Full Duty1 Register	0x00000000
0x3100A118	PWM0_CHA_DT	PWM0 Channel A Dead-time Register	0x00000000
0x3100A11C	PWM0_CHB_DT	PWM0 Channel B Dead-time Register	0x00000000
0x3100A120	PWM0_CHC_DT	PWM0 Channel C Dead-time Register	0x00000000
0x3100A124	PWM0_CHD_DT	PWM0 Channel D Dead-time Register	0x00000000

Table A-114: ADSP-SC58x PWM1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A400	PWM1_CTL	PWM1 Control Register	0x00020000
0x3100A404	PWM1_CHANCFG	PWM1 Channel Configuration Register	0x00000000
0x3100A408	PWM1_TRIPCFG	PWM1 Trip Configuration Register	0x00000000
0x3100A40C	PWM1_STAT	PWM1 Status Register	0x00000000
0x3100A410	PWM1_IMSK	PWM1 Interrupt Mask Register	0x00000000
0x3100A414	PWM1_ILAT	PWM1 Interrupt Latch Register	0x00000000
0x3100A418	PWM1_CHOPCFG	PWM1 Chop Configuration Register	0x00000000
0x3100A420	PWM1_SYNC_WID	PWM1 Sync Pulse Width Register	0x000003FF
0x3100A424	PWM1_TM0	PWM1 Timer 0 Period Register	0x00000000
0x3100A428	PWM1_TM1	PWM1 Timer 1 Period Register	0x00000000
0x3100A42C	PWM1_TM2	PWM1 Timer 2 Period Register	0x00000000
0x3100A430	PWM1_TM3	PWM1 Timer 3 Period Register	0x00000000
0x3100A434	PWM1_TM4	PWM1 Timer 4 Period Register	0x00000000

Table A-114: ADSP-SC58x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A438	PWM1_DLYA	PWM1 Channel A Delay Register	0x00000000
0x3100A43C	PWM1_DLYB	PWM1 Channel B Delay Register	0x00000000
0x3100A440	PWM1_DLYC	PWM1 Channel C Delay Register	0x00000000
0x3100A444	PWM1_DLYD	PWM1 Channel D Delay Register	0x00000000
0x3100A448	PWM1_ACTL	PWM1 Channel A Control Register	0x00000000
0x3100A44C	PWM1_AH0	PWM1 Channel A-High Duty-0 Register	0x00000000
0x3100A450	PWM1_AH1	PWM1 Channel A-High Duty-1 Register	0x00000000
0x3100A454	PWM1_AH0_HP	PWM1 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A458	PWM1_AH1_HP	PWM1 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A45C	PWM1_AL0	PWM1 Channel A-Low Duty-0 Register	0x00000000
0x3100A460	PWM1_AL1	PWM1 Channel A-Low Duty-1 Register	0x00000000
0x3100A464	PWM1_AL0_HP	PWM1 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A468	PWM1_AL1_HP	PWM1 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A46C	PWM1_BCTL	PWM1 Channel B Control Register	0x00000000
0x3100A470	PWM1_BH0	PWM1 Channel B-High Duty-0 Register	0x00000000
0x3100A474	PWM1_BH1	PWM1 Channel B-High Duty-1 Register	0x00000000
0x3100A478	PWM1_BH0_HP	PWM1 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A47C	PWM1_BH1_HP	PWM1 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A480	PWM1_BL0	PWM1 Channel B-Low Duty-0 Register	0x00000000
0x3100A484	PWM1_BL1	PWM1 Channel B-Low Duty-1 Register	0x00000000
0x3100A488	PWM1_BL0_HP	PWM1 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A48C	PWM1_BL1_HP	PWM1 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A490	PWM1_CCTL	PWM1 Channel C Control Register	0x00000000
0x3100A494	PWM1_CH0	PWM1 Channel C-High Pulse Duty Register 0	0x00000000
0x3100A498	PWM1_CH1	PWM1 Channel C-High Pulse Duty Register 1	0x00000000

Table A-114: ADSP-SC58x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A49C	PWM1_CH0_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A4A0	PWM1_CH1_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A4A4	PWM1_CL0	PWM1 Channel C-Low Pulse Duty Register 0	0x00000000
0x3100A4A8	PWM1_CL1	PWM1 Channel C-Low Duty-1 Register	0x00000000
0x3100A4AC	PWM1_CL0_HP	PWM1 Channel C-Low Pulse Duty Register 1	0x00000000
0x3100A4B0	PWM1_CL1_HP	PWM1 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A4B4	PWM1_DCTL	PWM1 Channel D Control Register	0x00000000
0x3100A4B8	PWM1_DH0	PWM1 Channel D-High Duty-0 Register	0x00000000
0x3100A4BC	PWM1_DH1	PWM1 Channel D-High Pulse Duty Register 1	0x00000000
0x3100A4C0	PWM1_DH0_HP	PWM1 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A4C4	PWM1_DH1_HP	PWM1 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A4C8	PWM1_DL0	PWM1 Channel D-Low Pulse Duty Register 0	0x00000000
0x3100A4CC	PWM1_DL1	PWM1 Channel D-Low Pulse Duty Register 1	0x00000000
0x3100A4D0	PWM1_DL0_HP	PWM1 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A4D4	PWM1_DL1_HP	PWM1 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A4D8	PWM1_AH_DUTY0	PWM1 Channel A-High Full Duty0 Register	0x00000000
0x3100A4DC	PWM1_AH_DUTY1	PWM1 Channel A-High Full Duty1 Register	0x00000000
0x3100A4E0	PWM1_AL_DUTY0	PWM1 Channel A-Low Full Duty0 Register	0x00000000
0x3100A4E4	PWM1_AL_DUTY1	PWM1 Channel A-Low Full Duty1 Register	0x00000000
0x3100A4E8	PWM1_BH_DUTY0	PWM1 Channel B-High Full Duty0 Register	0x00000000
0x3100A4EC	PWM1_BH_DUTY1	PWM1 Channel B-High Full Duty1 Register	0x00000000
0x3100A4F0	PWM1_BL_DUTY0	PWM1 Channel B-Low Full Duty0 Register	0x00000000
0x3100A4F4	PWM1_BL_DUTY1	PWM1 Channel B-Low Full Duty1 Register	0x00000000
0x3100A4F8	PWM1_CH_DUTY0	PWM1 Channel C-High Full Duty0 Register	0x00000000
0x3100A4FC	PWM1_CH_DUTY1	PWM1 Channel C-High Full Duty1 Register	0x00000000
0x3100A500	PWM1_CL_DUTY0	PWM1 Channel C-Low Full Duty0 Register	0x00000000

Table A-114: ADSP-SC58x PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A504	PWM1_CL_DUTY1	PWM1 Channel C-Low Full Duty1 Register	0x00000000
0x3100A508	PWM1_DH_DUTY0	PWM1 Channel D-High Full Duty0 Register	0x00000000
0x3100A50C	PWM1_DH_DUTY1	PWM1 Channel D-High Full Duty1 Register	0x00000000
0x3100A510	PWM1_DL_DUTY0	PWM1 Channel D-Low Full Duty0 Register	0x00000000
0x3100A514	PWM1_DL_DUTY1	PWM1 Channel D-Low Full Duty1 Register	0x00000000
0x3100A518	PWM1_CHA_DT	PWM1 Channel A Dead-time Register	0x00000000
0x3100A51C	PWM1_CHB_DT	PWM1 Channel B Dead-time Register	0x00000000
0x3100A520	PWM1_CHC_DT	PWM1 Channel C Dead-time Register	0x00000000
0x3100A524	PWM1_CHD_DT	PWM1 Channel D Dead-time Register	0x00000000

Table A-115: ADSP-SC58x PWM2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A800	PWM2_CTL	PWM2 Control Register	0x00020000
0x3100A804	PWM2_CHANCFG	PWM2 Channel Configuration Register	0x00000000
0x3100A808	PWM2_TRIPCFG	PWM2 Trip Configuration Register	0x00000000
0x3100A80C	PWM2_STAT	PWM2 Status Register	0x00000000
0x3100A810	PWM2_IMSK	PWM2 Interrupt Mask Register	0x00000000
0x3100A814	PWM2_ILAT	PWM2 Interrupt Latch Register	0x00000000
0x3100A818	PWM2_CHOPCFG	PWM2 Chop Configuration Register	0x00000000
0x3100A820	PWM2_SYNC_WID	PWM2 Sync Pulse Width Register	0x000003FF
0x3100A824	PWM2_TM0	PWM2 Timer 0 Period Register	0x00000000
0x3100A828	PWM2_TM1	PWM2 Timer 1 Period Register	0x00000000
0x3100A82C	PWM2_TM2	PWM2 Timer 2 Period Register	0x00000000
0x3100A830	PWM2_TM3	PWM2 Timer 3 Period Register	0x00000000
0x3100A834	PWM2_TM4	PWM2 Timer 4 Period Register	0x00000000
0x3100A838	PWM2_DLYA	PWM2 Channel A Delay Register	0x00000000
0x3100A83C	PWM2_DLYB	PWM2 Channel B Delay Register	0x00000000
0x3100A840	PWM2_DLYC	PWM2 Channel C Delay Register	0x00000000
0x3100A844	PWM2_DLYD	PWM2 Channel D Delay Register	0x00000000
0x3100A848	PWM2_ACTL	PWM2 Channel A Control Register	0x00000000

Table A-115: ADSP-SC58x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A84C	PWM2_AH0	PWM2 Channel A-High Duty-0 Register	0x00000000
0x3100A850	PWM2_AH1	PWM2 Channel A-High Duty-1 Register	0x00000000
0x3100A854	PWM2_AH0_HP	PWM2 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A858	PWM2_AH1_HP	PWM2 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A85C	PWM2_AL0	PWM2 Channel A-Low Duty-0 Register	0x00000000
0x3100A860	PWM2_AL1	PWM2 Channel A-Low Duty-1 Register	0x00000000
0x3100A864	PWM2_AL0_HP	PWM2 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A868	PWM2_AL1_HP	PWM2 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A86C	PWM2_BCTL	PWM2 Channel B Control Register	0x00000000
0x3100A870	PWM2_BH0	PWM2 Channel B-High Duty-0 Register	0x00000000
0x3100A874	PWM2_BH1	PWM2 Channel B-High Duty-1 Register	0x00000000
0x3100A878	PWM2_BH0_HP	PWM2 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x3100A87C	PWM2_BH1_HP	PWM2 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x3100A880	PWM2_BL0	PWM2 Channel B-Low Duty-0 Register	0x00000000
0x3100A884	PWM2_BL1	PWM2 Channel B-Low Duty-1 Register	0x00000000
0x3100A888	PWM2_BL0_HP	PWM2 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A88C	PWM2_BL1_HP	PWM2 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A890	PWM2_CCTL	PWM2 Channel C Control Register	0x00000000
0x3100A894	PWM2_CH0	PWM2 Channel C-High Pulse Duty Register 0	0x00000000
0x3100A898	PWM2_CH1	PWM2 Channel C-High Pulse Duty Register 1	0x00000000
0x3100A89C	PWM2_CH0_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A8A0	PWM2_CH1_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A8A4	PWM2_CL0	PWM2 Channel C-Low Pulse Duty Register 0	0x00000000
0x3100A8A8	PWM2_CL1	PWM2 Channel C-Low Duty-1 Register	0x00000000

Table A-115: ADSP-SC58x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A8AC	PWM2_CL0_HP	PWM2 Channel C-Low Pulse Duty Register 1	0x00000000
0x3100A8B0	PWM2_CL1_HP	PWM2 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A8B4	PWM2_DCTL	PWM2 Channel D Control Register	0x00000000
0x3100A8B8	PWM2_DH0	PWM2 Channel D-High Duty-0 Register	0x00000000
0x3100A8BC	PWM2_DH1	PWM2 Channel D-High Pulse Duty Register 1	0x00000000
0x3100A8C0	PWM2_DH0_HP	PWM2 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x3100A8C4	PWM2_DH1_HP	PWM2 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x3100A8C8	PWM2_DL0	PWM2 Channel D-Low Pulse Duty Register 0	0x00000000
0x3100A8CC	PWM2_DL1	PWM2 Channel D-Low Pulse Duty Register 1	0x00000000
0x3100A8D0	PWM2_DL0_HP	PWM2 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x3100A8D4	PWM2_DL1_HP	PWM2 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x3100A8D8	PWM2_AH_DUTY0	PWM2 Channel A-High Full Duty0 Register	0x00000000
0x3100A8DC	PWM2_AH_DUTY1	PWM2 Channel A-High Full Duty1 Register	0x00000000
0x3100A8E0	PWM2_AL_DUTY0	PWM2 Channel A-Low Full Duty0 Register	0x00000000
0x3100A8E4	PWM2_AL_DUTY1	PWM2 Channel A-Low Full Duty1 Register	0x00000000
0x3100A8E8	PWM2_BH_DUTY0	PWM2 Channel B-High Full Duty0 Register	0x00000000
0x3100A8EC	PWM2_BH_DUTY1	PWM2 Channel B-High Full Duty1 Register	0x00000000
0x3100A8F0	PWM2_BL_DUTY0	PWM2 Channel B-Low Full Duty0 Register	0x00000000
0x3100A8F4	PWM2_BL_DUTY1	PWM2 Channel B-Low Full Duty1 Register	0x00000000
0x3100A8F8	PWM2_CH_DUTY0	PWM2 Channel C-High Full Duty0 Register	0x00000000
0x3100A8FC	PWM2_CH_DUTY1	PWM2 Channel C-High Full Duty1 Register	0x00000000
0x3100A900	PWM2_CL_DUTY0	PWM2 Channel C-Low Full Duty0 Register	0x00000000
0x3100A904	PWM2_CL_DUTY1	PWM2 Channel C-Low Full Duty1 Register	0x00000000
0x3100A908	PWM2_DH_DUTY0	PWM2 Channel D-High Full Duty0 Register	0x00000000
0x3100A90C	PWM2_DH_DUTY1	PWM2 Channel D-High Full Duty1 Register	0x00000000
0x3100A910	PWM2_DL_DUTY0	PWM2 Channel D-Low Full Duty0 Register	0x00000000
0x3100A914	PWM2_DL_DUTY1	PWM2 Channel D-Low Full Duty1 Register	0x00000000

Table A-115: ADSP-SC58x PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100A918	PWM2_CHA_DT	PWM2 Channel A Dead-time Register	0x00000000
0x3100A91C	PWM2_CHB_DT	PWM2 Channel B Dead-time Register	0x00000000
0x3100A920	PWM2_CHC_DT	PWM2 Channel C Dead-time Register	0x00000000
0x3100A924	PWM2_CHD_DT	PWM2 Channel D Dead-time Register	0x00000000

Table A-116: ADSP-SC58x RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108B000	RCU0_CTL	RCU0 Control Register	0x00000300
0x3108B004	RCU0_STAT	RCU0 Status Register	0x00000021
0x3108B008	RCU0_CRCTL	RCU0 Core Reset Outputs Control Register	0x00000006
0x3108B00C	RCU0_CRSTAT	RCU0 Core Reset Outputs Status Register	0x00000006
0x3108B010	RCU0_SIDIS	RCU0 System Interface Disable Register	0x00000000
0x3108B014	RCU0_SISTAT	RCU0 System Interface Status Register	0x00000000
0x3108B018	RCU0_SVECT_LCK	RCU0 SVECT Lock Register	0x00000000
0x3108B01C	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x3108B020	RCU0_SVECT0	RCU0 Software Vector Register 0	0x00000020
0x3108B024	RCU0_SVECT1	RCU0 Software Vector Register 1	0x00500080
0x3108B028	RCU0_SVECT2	RCU0 Software Vector Register 2	0x00500080
0x3108B060	RCU0_MSG	RCU0 Message Register	0x00000000
0x3108B064	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x3108B068	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000

Table A-117: ADSP-SC58x RTC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C8000	RTC0_CLK	RTC0 RTC Clock Register	0x00000000
0x310C8004	RTC0_ALM	RTC0 RTC Alarm Register	0x00000000
0x310C8008	RTC0_IEN	RTC0 Interrupt Enable Register	0x00000000
0x310C800C	RTC0_STAT	RTC0 RTC Status Register	0x00000000
0x310C8010	RTC0_STPWTCH	RTC0 RTC Stop Watch Register	0x00000000
0x310C8018	RTC0_INIT	RTC0 RTC Initialization Register	0x00000000

Table A-117: ADSP-SC58x RTC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C801C	RTC0_INITSTAT	RTC0 RTC Initialization Status Register	0x00000000

Table A-118: ADSP-SC58x SCB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x60042100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60042104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60043100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60043104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60044100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60044104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60045100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60045104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60046100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60046104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60047100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60047104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60048100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60048104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60049100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60049104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6004A100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004A104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6004B100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004B104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6004C100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004C104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6004D100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004D104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6004E100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004E104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001

Table A-118: ADSP-SC58x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x6004F100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6004F104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60050100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60050104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60051100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60051104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60052100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60052104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60053100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60053104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60054100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60054104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60055100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60055104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60056100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60056104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60057100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60057104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60058100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60058104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60059100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60059104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005A100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6005A104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005B100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6005B104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005C100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6005C104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005D100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6005D104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005E100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001

Table A-118: ADSP-SC58x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x6005E104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6005F100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6005F104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60060100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60060104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60061100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60061104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60062100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60062104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60063100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60063104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60064100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60064104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60065100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60065104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60066100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60066104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60067100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60067104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60068100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60068104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60069100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60069104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6006A100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006A104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6006B100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006B104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6006C100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006C104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6006D100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006D104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001

Table A-118: ADSP-SC58x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x6006E100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006E104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6006F100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6006F104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60070100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60070104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60071100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60071104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60072100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60072104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60073100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60073104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60074100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60074104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60075100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60075104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60076100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60076104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60077100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60077104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60078100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60078104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60079100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60079104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007A100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6007A104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007B100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6007B104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007C100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6007C104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007D100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001

Table A-118: ADSP-SC58x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x6007D104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007E100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6007E104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x6007F100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x6007F104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60080100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60080104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60081100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60081104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60082100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60082104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60083100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60083104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60084100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60084104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60085100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60085104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60086100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60086104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60087100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60087104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001
0x60088100	SCB0_MST[n]_RQOS	SCB0 Read Quality of Service for Master n	0x00000001
0x60088104	SCB0_MST[n]_WQOS	SCB0 Write Quality of Service for Master n	0x00000001

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31089000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x31089004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x31089008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000
0x3108900C	SEC0_END	SEC0 Global End Register	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000
0x31089014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x31089018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x3108901C	SEC0_FEND	SEC0 Fault End Register	0x00000000
0x31089020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x31089024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x31089028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x3108902C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x31089030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x31089034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x31089440	SEC0_CCTL[n]	SEC0 SCI Control Register n	0x00000000
0x31089444	SEC0_CSTAT[n]	SEC0 SCI Status Register n	0x00000000
0x31089448	SEC0_CPND[n]	SEC0 Core Pending Register n	0x00000000
0x3108944C	SEC0_CACT[n]	SEC0 SCI Active Register n	0x00000000
0x31089450	SEC0_CPMSK[n]	SEC0 SCI Priority Mask Register n	0x000000FF
0x31089454	SEC0_CGMSK[n]	SEC0 SCI Group Mask Register n	0x00000000
0x31089458	SEC0_CPLVL[n]	SEC0 SCI Priority Level Register n	0x00000007
0x3108945C	SEC0_CSID[n]	SEC0 SCI Source ID Register n	0x00000000
0x31089480	SEC0_CCTL[n]	SEC0 SCI Control Register n	0x00000000
0x31089484	SEC0_CSTAT[n]	SEC0 SCI Status Register n	0x00000000
0x31089488	SEC0_CPND[n]	SEC0 Core Pending Register n	0x00000000
0x3108948C	SEC0_CACT[n]	SEC0 SCI Active Register n	0x00000000
0x31089490	SEC0_CPMSK[n]	SEC0 SCI Priority Mask Register n	0x000000FF
0x31089494	SEC0_CGMSK[n]	SEC0 SCI Group Mask Register n	0x00000000
0x31089498	SEC0_CPLVL[n]	SEC0 SCI Priority Level Register n	0x00000007
0x3108949C	SEC0_CSID[n]	SEC0 SCI Source ID Register n	0x00000000
0x31089800	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089804	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089808	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108980C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089810	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089814	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089818	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108981C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089820	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089824	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089828	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108982C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089830	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089834	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089838	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108983C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089840	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089844	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089848	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108984C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089850	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089854	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089858	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108985C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089860	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089864	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089868	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108986C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089870	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089874	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089878	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108987C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089880	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089884	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089888	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108988C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089890	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089894	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089898	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108989C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310898F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310898FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089900	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089904	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089908	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108990C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089910	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089914	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089918	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108991C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089920	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089924	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089928	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108992C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089930	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089934	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089938	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108993C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089940	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089944	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089948	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108994C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089950	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089954	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089958	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108995C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089960	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089964	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089968	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108996C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089970	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089974	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089978	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108997C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089980	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089984	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089988	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108998C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089990	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089994	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089998	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x3108999C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x310899F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x310899FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089A04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089A80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089A98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089A9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ABC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ACC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ADC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089AF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089AF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089AFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089B78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089B98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089B9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089BF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089BF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089BFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089C70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089C98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089C9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089CEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089CF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089CFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089D68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089D98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089D9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089DE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089DF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089DFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089E60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089E98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089E9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ECC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089ED0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089ED4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089ED8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089EDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089EF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089EFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089F58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089F98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089F9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FBC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FCC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-119: ADSP-SC58x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31089FD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FDC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x31089FF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x31089FFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-120: ADSP-SC58x SINC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31014000	SINC0_CTL	SINC0 Control Register	0x00000000
0x31014004	SINC0_STAT	SINC0 Status Register	0x00000000
0x31014008	SINC0_CLK	SINC0 Clock Control Register	0x10001000
0x31014010	SINC0_RATE0	SINC0 Rate Control for Group 0 Register	0x00000804
0x31014014	SINC0_RATE1	SINC0 Rate Control for Group 1 Register	0x00000804
0x31014018	SINC0_LEVEL0	SINC0 Level Control for Group 0 Register	0x04000000
0x3101401C	SINC0_LEVEL1	SINC0 Level Control for Group 1 Register	0x04000000
0x31014020	SINC0_LIMIT0	SINC0 (Amplitude) Limits for Secondary Filter 0 Register	0x00000000
0x31014024	SINC0_LIMIT1	SINC0 (Amplitude) Limits for Secondary Filter 1 Register	0x00000000
0x31014028	SINC0_LIMIT2	SINC0 (Amplitude) Limits for Secondary Filter 2 Register	0x00000000
0x3101402C	SINC0_LIMIT3	SINC0 (Amplitude) Limits for Secondary Filter 3 Register	0x00000000
0x31014030	SINC0_BIAS0	SINC0 Bias for Group 0 Register	0x00000000
0x31014034	SINC0_BIAS1	SINC0 Bias for Group 1 Register	0x00000000
0x31014038	SINC0_PPTR0	SINC0 Primary (Filters) Pointer for Group 0 Register	0x00000000
0x3101403C	SINC0_PPTR1	SINC0 Primary (Filters) Pointer for Group 1 Register	0x00000000
0x31014040	SINC0_PHEAD0	SINC0 Primary (Filters) Head for Group 0 Register	0x00000000

Table A-120: ADSP-SC58x SINC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31014044	SINC0_PHEAD1	SINC0 Primary (Filters) Head for Group 1 Register	0x00000000
0x31014048	SINC0_PTAIL0	SINC0 Primary (Filters) Tail for Group 0 Register	0x00000000
0x3101404C	SINC0_PTAIL1	SINC0 Primary (Filters) Tail for Group 1 Register	0x00000000
0x31014050	SINC0_HIS_STAT	SINC0 History Status Register	0x00000000
0x31014080	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x31014084	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x31014088	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x3101408C	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x31014090	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x31014094	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x31014098	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x3101409C	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x310140A0	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x310140A4	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x310140A8	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x310140AC	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x310140B0	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x310140B4	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x310140B8	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x310140BC	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000

Table A-121: ADSP-SC58x SMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3100600C	SMC0_B0CTL	SMC0 Bank 0 Control Register	0x01000000
0x31006010	SMC0_B0TIM	SMC0 Bank 0 Timing Register	0x01010101
0x31006014	SMC0_B0ETIM	SMC0 Bank 0 Extended Timing Register	0x00020200
0x3100601C	SMC0_B1CTL	SMC0 Bank 1 Control Register	0x00000000
0x31006020	SMC0_B1TIM	SMC0 Bank 1 Timing Register	0x01010101
0x31006024	SMC0_B1ETIM	SMC0 Bank 1 Extended Timing Register	0x00020200
0x3100602C	SMC0_B2CTL	SMC0 Bank 2 Control Register	0x00000000

Table A-121: ADSP-SC58x SMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31006030	SMC0_B2TIM	SMC0 Bank 2 Timing Register	0x01010101
0x31006034	SMC0_B2ETIM	SMC0 Bank 2 Extended Timing Register	0x00020200
0x3100603C	SMC0_B3CTL	SMC0 Bank 3 Control Register	0x00000000
0x31006040	SMC0_B3TIM	SMC0 Bank 3 Timing Register	0x01010101
0x31006044	SMC0_B3ETIM	SMC0 Bank 3 Extended Timing Register	0x00020200

Table A-122: ADSP-SC58x SMPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31007000	SMPU0_CTL	SMPU0 SMPU Control Register	0x00000000
0x31007004	SMPU0_STAT	SMPU0 SMPU Status Register	0x00000000
0x31007008	SMPU0_IADDR	SMPU0 Interrupt Address Register	0x00000000
0x3100700C	SMPU0_IDTLS	SMPU0 Interrupt Details Register	0x00000000
0x31007010	SMPU0_BADDR	SMPU0 Bus Error Address Register	0x00000000
0x31007014	SMPU0_BDTLS	SMPU0 Bus Error Details Register	0x00000000
0x31007020	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x31007024	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x31007028	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x3100702C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x31007030	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x31007034	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x31007038	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x3100703C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x31007040	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x31007044	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x31007048	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x3100704C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x31007050	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x31007054	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x31007058	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x3100705C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000

Table A-122: ADSP-SC58x SMPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31007060	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x31007064	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x31007068	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x3100706C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x31007070	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x31007074	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x31007078	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x3100707C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x31007080	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x31007084	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x31007088	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x3100708C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x31007090	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x31007094	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x31007098	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x3100709C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x310070A0	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x310070A4	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x310070A8	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x310070AC	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x310070B0	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x310070B4	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x310070B8	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x310070BC	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x310070C0	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x310070C4	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x310070C8	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x310070CC	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x310070D0	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x310070D4	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x310070D8	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000

Table A-122: ADSP-SC58x SMPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310070DC	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x310071A0	SMPU0_EXACADD[n]	SMPU0 Exclusive Access IDn Address	0x00000000
0x310071A4	SMPU0_EXACSTAT[n]	SMPU0 Exclusive Access Status	0x00000000
0x310071A8	SMPU0_EXACADD[n]	SMPU0 Exclusive Access IDn Address	0x00000000
0x310071AC	SMPU0_EXACSTAT[n]	SMPU0 Exclusive Access Status	0x00000000
0x310071B0	SMPU0_EXACADD[n]	SMPU0 Exclusive Access IDn Address	0x00000000
0x310071B4	SMPU0_EXACSTAT[n]	SMPU0 Exclusive Access Status	0x00000000
0x310071B8	SMPU0_EXACADD[n]	SMPU0 Exclusive Access IDn Address	0x00000000
0x310071BC	SMPU0_EXACSTAT[n]	SMPU0 Exclusive Access Status	0x00000000
0x31007220	SMPU0_REVID	SMPU0 SMPU Revision ID Register	0x00000010
0x31007800	SMPU0_SECURECTL	SMPU0 SMPU Control Secure Accesses Register	0x00000000
0x31007820	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x31007824	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x31007828	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x3100782C	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x31007830	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x31007834	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x31007838	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x3100783C	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000

Table A-123: ADSP-SC58x SMPU10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A1000	SMPU10_CTL	SMPU10 SMPU Control Register	0x00000000
0x310A1004	SMPU10_STAT	SMPU10 SMPU Status Register	0x00000000
0x310A1008	SMPU10_IADDR	SMPU10 Interrupt Address Register	0x00000000
0x310A100C	SMPU10_IDTLS	SMPU10 Interrupt Details Register	0x00000000
0x310A1010	SMPU10_BADDR	SMPU10 Bus Error Address Register	0x00000000
0x310A1014	SMPU10_BDTLS	SMPU10 Bus Error Details Register	0x00000000
0x310A1020	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A1024	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000

Table A-123: ADSP-SC58x SMPU10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A1028	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A102C	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A1030	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A1034	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A1038	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A103C	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A1040	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A1044	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A1048	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A104C	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A1050	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A1054	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A1058	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A105C	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A1060	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A1064	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A1068	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A106C	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A1070	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A1074	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A1078	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A107C	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A1080	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A1084	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A1088	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A108C	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A1090	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A1094	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A1098	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A109C	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A10A0	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000

Table A-123: ADSP-SC58x SMPU10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A10A4	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A10A8	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A10AC	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A10B0	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A10B4	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A10B8	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A10BC	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A10C0	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A10C4	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A10C8	SMPU10_RCTL[n]	SMPU10 Region n Control Register	0x00000000
0x310A10CC	SMPU10_RADDR[n]	SMPU10 Region n Address Register	0x00000000
0x310A10D0	SMPU10_RIDA[n]	SMPU10 Region n ID A Register	0x00000000
0x310A10D4	SMPU10_RIDMSKA[n]	SMPU10 Region n ID Mask A Register	0x00000000
0x310A10D8	SMPU10_RIDB[n]	SMPU10 Region n ID B Register	0x00000000
0x310A10DC	SMPU10_RIDMSKB[n]	SMPU10 Region n ID Mask B Register	0x00000000
0x310A11A0	SMPU10_EXACADD[n]	SMPU10 Exclusive Access IDn Address	0x00000000
0x310A11A4	SMPU10_EXACSTAT[n]	SMPU10 Exclusive Access Status	0x00000000
0x310A11A8	SMPU10_EXACADD[n]	SMPU10 Exclusive Access IDn Address	0x00000000
0x310A11AC	SMPU10_EXACSTAT[n]	SMPU10 Exclusive Access Status	0x00000000
0x310A11B0	SMPU10_EXACADD[n]	SMPU10 Exclusive Access IDn Address	0x00000000
0x310A11B4	SMPU10_EXACSTAT[n]	SMPU10 Exclusive Access Status	0x00000000
0x310A11B8	SMPU10_EXACADD[n]	SMPU10 Exclusive Access IDn Address	0x00000000
0x310A11BC	SMPU10_EXACSTAT[n]	SMPU10 Exclusive Access Status	0x00000000
0x310A1220	SMPU10_REVID	SMPU10 SMPU Revision ID Register	0x00000010
0x310A1800	SMPU10_SECURECTL	SMPU10 SMPU Control Secure Accesses Register	0x00000000
0x310A1820	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A1824	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A1828	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A182C	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A1830	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A1834	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000

Table A-123: ADSP-SC58x SMPU10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A1838	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000
0x310A183C	SMPU10_SECURERCTL[n]	SMPU10 Region n Control Secure Accesses Register	0x00000000

Table A-124: ADSP-SC58x SMPU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31083000	SMPU2_CTL	SMPU2 SMPU Control Register	0x00000000
0x31083004	SMPU2_STAT	SMPU2 SMPU Status Register	0x00000000
0x31083008	SMPU2_IADDR	SMPU2 Interrupt Address Register	0x00000000
0x3108300C	SMPU2_IDTLS	SMPU2 Interrupt Details Register	0x00000000
0x31083010	SMPU2_BADDR	SMPU2 Bus Error Address Register	0x00000000
0x31083014	SMPU2_BDTLS	SMPU2 Bus Error Details Register	0x00000000
0x31083020	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x31083024	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083028	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108302C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083030	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083034	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083038	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x3108303C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083040	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x31083044	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083048	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x3108304C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083050	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x31083054	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083058	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108305C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083060	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083064	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083068	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000

Table A-124: ADSP-SC58x SMPU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108306C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083070	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x31083074	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083078	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x3108307C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083080	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x31083084	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x31083088	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x3108308C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x31083090	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x31083094	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x31083098	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x3108309C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830A0	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830A4	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830A8	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830AC	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x310830B0	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x310830B4	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830B8	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830BC	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830C0	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830C4	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x310830C8	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x310830CC	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x310830D0	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x310830D4	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x310830D8	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x310830DC	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x310831A0	SMPU2_EXACADD[n]	SMPU2 Exclusive Access IDn Address	0x00000000
0x310831A4	SMPU2_EXACSTAT[n]	SMPU2 Exclusive Access Status	0x00000000

Table A-124: ADSP-SC58x SMPU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310831A8	SMPU2_EXACADD[n]	SMPU2 Exclusive Access IDn Address	0x00000000
0x310831AC	SMPU2_EXACSTAT[n]	SMPU2 Exclusive Access Status	0x00000000
0x310831B0	SMPU2_EXACADD[n]	SMPU2 Exclusive Access IDn Address	0x00000000
0x310831B4	SMPU2_EXACSTAT[n]	SMPU2 Exclusive Access Status	0x00000000
0x310831B8	SMPU2_EXACADD[n]	SMPU2 Exclusive Access IDn Address	0x00000000
0x310831BC	SMPU2_EXACSTAT[n]	SMPU2 Exclusive Access Status	0x00000000
0x31083220	SMPU2_REVID	SMPU2 SMPU Revision ID Register	0x00000010
0x31083800	SMPU2_SECURECTL	SMPU2 SMPU Control Secure Accesses Register	0x00000000
0x31083820	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083824	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083828	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x3108382C	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083830	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083834	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x31083838	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000
0x3108383C	SMPU2_SECURERCTL[n]	SMPU2 Region n Control Secure Accesses Register	0x00000000

Table A-125: ADSP-SC58x SMPU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31084000	SMPU3_CTL	SMPU3 SMPU Control Register	0x00000000
0x31084004	SMPU3_STAT	SMPU3 SMPU Status Register	0x00000000
0x31084008	SMPU3_IADDR	SMPU3 Interrupt Address Register	0x00000000
0x3108400C	SMPU3_IDTLS	SMPU3 Interrupt Details Register	0x00000000
0x31084010	SMPU3_BADDR	SMPU3 Bus Error Address Register	0x00000000
0x31084014	SMPU3_BDTLS	SMPU3 Bus Error Details Register	0x00000000
0x31084020	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084024	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084028	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108402C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084030	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000

Table A-125: ADSP-SC58x SMPU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31084034	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084038	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x3108403C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084040	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x31084044	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084048	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x3108404C	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084050	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084054	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084058	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108405C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084060	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x31084064	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084068	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x3108406C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084070	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x31084074	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084078	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x3108407C	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084080	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x31084084	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x31084088	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x3108408C	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x31084090	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x31084094	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084098	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x3108409C	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840A0	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840A4	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840A8	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840AC	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000

Table A-125: ADSP-SC58x SMPU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310840B0	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x310840B4	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840B8	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840BC	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840C0	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840C4	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x310840C8	SMPU3_RCTL[n]	SMPU3 Region n Control Register	0x00000000
0x310840CC	SMPU3_RADDR[n]	SMPU3 Region n Address Register	0x00000000
0x310840D0	SMPU3_RIDA[n]	SMPU3 Region n ID A Register	0x00000000
0x310840D4	SMPU3_RIDMSKA[n]	SMPU3 Region n ID Mask A Register	0x00000000
0x310840D8	SMPU3_RIDB[n]	SMPU3 Region n ID B Register	0x00000000
0x310840DC	SMPU3_RIDMSKB[n]	SMPU3 Region n ID Mask B Register	0x00000000
0x31084220	SMPU3_REVID	SMPU3 SMPU Revision ID Register	0x00000010
0x31084800	SMPU3_SECURECTL	SMPU3 SMPU Control Secure Accesses Register	0x00000000
0x31084820	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084824	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084828	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x3108482C	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084830	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084834	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x31084838	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000
0x3108483C	SMPU3_SECURERCTL[n]	SMPU3 Region n Control Secure Accesses Register	0x00000000

Table A-126: ADSP-SC58x SMPU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31085000	SMPU4_CTL	SMPU4 SMPU Control Register	0x00000000
0x31085004	SMPU4_STAT	SMPU4 SMPU Status Register	0x00000000
0x31085008	SMPU4_IADDR	SMPU4 Interrupt Address Register	0x00000000
0x3108500C	SMPU4_IDTLS	SMPU4 Interrupt Details Register	0x00000000
0x31085010	SMPU4_BADDR	SMPU4 Bus Error Address Register	0x00000000

Table A-126: ADSP-SC58x SMPU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31085014	SMPU4_BDTLS	SMPU4 Bus Error Details Register	0x00000000
0x31085020	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x31085024	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x31085028	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x3108502C	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x31085030	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x31085034	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x31085038	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x3108503C	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x31085040	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x31085044	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x31085048	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x3108504C	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x31085050	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x31085054	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x31085058	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x3108505C	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x31085060	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x31085064	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x31085068	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x3108506C	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x31085070	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x31085074	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x31085078	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x3108507C	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x31085080	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x31085084	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x31085088	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x3108508C	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x31085090	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x31085094	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000

Table A-126: ADSP-SC58x SMPU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31085098	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x3108509C	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x310850A0	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x310850A4	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x310850A8	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x310850AC	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x310850B0	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x310850B4	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x310850B8	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x310850BC	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x310850C0	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x310850C4	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x310850C8	SMPU4_RCTL[n]	SMPU4 Region n Control Register	0x00000000
0x310850CC	SMPU4_RADDR[n]	SMPU4 Region n Address Register	0x00000000
0x310850D0	SMPU4_RIDA[n]	SMPU4 Region n ID A Register	0x00000000
0x310850D4	SMPU4_RIDMSKA[n]	SMPU4 Region n ID Mask A Register	0x00000000
0x310850D8	SMPU4_RIDB[n]	SMPU4 Region n ID B Register	0x00000000
0x310850DC	SMPU4_RIDMSKB[n]	SMPU4 Region n ID Mask B Register	0x00000000
0x31085220	SMPU4_REVID	SMPU4 SMPU Revision ID Register	0x00000010
0x31085800	SMPU4_SECURECTL	SMPU4 SMPU Control Secure Accesses Register	0x00000000
0x31085820	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x31085824	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x31085828	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x3108582C	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x31085830	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x31085834	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x31085838	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000
0x3108583C	SMPU4_SECURERCTL[n]	SMPU4 Region n Control Secure Accesses Register	0x00000000

Table A-127: ADSP-SC58x SMPU5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31086000	SMPU5_CTL	SMPU5 SMPU Control Register	0x00000000
0x31086004	SMPU5_STAT	SMPU5 SMPU Status Register	0x00000000
0x31086008	SMPU5_IADDR	SMPU5 Interrupt Address Register	0x00000000
0x3108600C	SMPU5_IDTLS	SMPU5 Interrupt Details Register	0x00000000
0x31086010	SMPU5_BADDR	SMPU5 Bus Error Address Register	0x00000000
0x31086014	SMPU5_BDTLS	SMPU5 Bus Error Details Register	0x00000000
0x31086020	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x31086024	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x31086028	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x3108602C	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x31086030	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x31086034	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086038	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x3108603C	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x31086040	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x31086044	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x31086048	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x3108604C	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086050	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x31086054	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x31086058	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x3108605C	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x31086060	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x31086064	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086068	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x3108606C	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x31086070	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x31086074	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x31086078	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x3108607C	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086080	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000

Table A-127: ADSP-SC58x SMPU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31086084	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x31086088	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x3108608C	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x31086090	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x31086094	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086098	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x3108609C	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x310860A0	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x310860A4	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x310860A8	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x310860AC	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x310860B0	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x310860B4	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x310860B8	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x310860BC	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x310860C0	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x310860C4	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x310860C8	SMPU5_RCTL[n]	SMPU5 Region n Control Register	0x00000000
0x310860CC	SMPU5_RADDR[n]	SMPU5 Region n Address Register	0x00000000
0x310860D0	SMPU5_RIDA[n]	SMPU5 Region n ID A Register	0x00000000
0x310860D4	SMPU5_RIDMSKA[n]	SMPU5 Region n ID Mask A Register	0x00000000
0x310860D8	SMPU5_RIDB[n]	SMPU5 Region n ID B Register	0x00000000
0x310860DC	SMPU5_RIDMSKB[n]	SMPU5 Region n ID Mask B Register	0x00000000
0x31086220	SMPU5_REVID	SMPU5 SMPU Revision ID Register	0x00000010
0x31086800	SMPU5_SECURECTL	SMPU5 SMPU Control Secure Accesses Register	0x00000000
0x31086820	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x31086824	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x31086828	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x3108682C	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x31086830	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x31086834	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000

Table A-127: ADSP-SC58x SMPU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31086838	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000
0x3108683C	SMPU5_SECURERCTL[n]	SMPU5 Region n Control Secure Accesses Register	0x00000000

Table A-128: ADSP-SC58x SMPU6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31087000	SMPU6_CTL	SMPU6 SMPU Control Register	0x00000000
0x31087004	SMPU6_STAT	SMPU6 SMPU Status Register	0x00000000
0x31087008	SMPU6_IADDR	SMPU6 Interrupt Address Register	0x00000000
0x3108700C	SMPU6_IDTLS	SMPU6 Interrupt Details Register	0x00000000
0x31087010	SMPU6_BADDR	SMPU6 Bus Error Address Register	0x00000000
0x31087014	SMPU6_BDTLS	SMPU6 Bus Error Details Register	0x00000000
0x31087020	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x31087024	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x31087028	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x3108702C	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x31087030	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x31087034	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087038	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x3108703C	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x31087040	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x31087044	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x31087048	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x3108704C	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087050	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x31087054	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x31087058	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x3108705C	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x31087060	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x31087064	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087068	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000

Table A-128: ADSP-SC58x SMPU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108706C	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x31087070	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x31087074	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x31087078	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x3108707C	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087080	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x31087084	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x31087088	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x3108708C	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x31087090	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x31087094	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087098	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x3108709C	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x310870A0	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x310870A4	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x310870A8	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x310870AC	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x310870B0	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x310870B4	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x310870B8	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x310870BC	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x310870C0	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x310870C4	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x310870C8	SMPU6_RCTL[n]	SMPU6 Region n Control Register	0x00000000
0x310870CC	SMPU6_RADDR[n]	SMPU6 Region n Address Register	0x00000000
0x310870D0	SMPU6_RIDA[n]	SMPU6 Region n ID A Register	0x00000000
0x310870D4	SMPU6_RIDMSKA[n]	SMPU6 Region n ID Mask A Register	0x00000000
0x310870D8	SMPU6_RIDB[n]	SMPU6 Region n ID B Register	0x00000000
0x310870DC	SMPU6_RIDMSKB[n]	SMPU6 Region n ID Mask B Register	0x00000000
0x31087220	SMPU6_REVID	SMPU6 SMPU Revision ID Register	0x00000010
0x31087800	SMPU6_SECURECTL	SMPU6 SMPU Control Secure Accesses Register	0x00000000

Table A-128: ADSP-SC58x SMPU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31087820	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x31087824	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x31087828	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x3108782C	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x31087830	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x31087834	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x31087838	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000
0x3108783C	SMPU6_SECURERCTL[n]	SMPU6 Region n Control Secure Accesses Register	0x00000000

Table A-129: ADSP-SC58x SMPU7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31088000	SMPU7_CTL	SMPU7 SMPU Control Register	0x00000000
0x31088004	SMPU7_STAT	SMPU7 SMPU Status Register	0x00000000
0x31088008	SMPU7_IADDR	SMPU7 Interrupt Address Register	0x00000000
0x3108800C	SMPU7_IDTLS	SMPU7 Interrupt Details Register	0x00000000
0x31088010	SMPU7_BADDR	SMPU7 Bus Error Address Register	0x00000000
0x31088014	SMPU7_BDTLS	SMPU7 Bus Error Details Register	0x00000000
0x31088020	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x31088024	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x31088028	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x3108802C	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x31088030	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x31088034	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088038	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x3108803C	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x31088040	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x31088044	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x31088048	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x3108804C	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088050	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000

Table A-129: ADSP-SC58x SMPU7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31088054	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x31088058	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x3108805C	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x31088060	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x31088064	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088068	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x3108806C	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x31088070	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x31088074	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x31088078	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x3108807C	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088080	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x31088084	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x31088088	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x3108808C	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x31088090	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x31088094	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088098	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x3108809C	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x310880A0	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x310880A4	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x310880A8	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x310880AC	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x310880B0	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x310880B4	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000
0x310880B8	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x310880BC	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x310880C0	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x310880C4	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x310880C8	SMPU7_RCTL[n]	SMPU7 Region n Control Register	0x00000000
0x310880CC	SMPU7_RADDR[n]	SMPU7 Region n Address Register	0x00000000

Table A-129: ADSP-SC58x SMPU7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310880D0	SMPU7_RIDA[n]	SMPU7 Region n ID A Register	0x00000000
0x310880D4	SMPU7_RIDMSKA[n]	SMPU7 Region n ID Mask A Register	0x00000000
0x310880D8	SMPU7_RIDB[n]	SMPU7 Region n ID B Register	0x00000000
0x310880DC	SMPU7_RIDMSKB[n]	SMPU7 Region n ID Mask B Register	0x00000000
0x31088220	SMPU7_REVID	SMPU7 SMPU Revision ID Register	0x00000010
0x31088800	SMPU7_SECURECTL	SMPU7 SMPU Control Secure Accesses Register	0x00000000
0x31088820	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x31088824	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x31088828	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x3108882C	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x31088830	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x31088834	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x31088838	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000
0x3108883C	SMPU7_SECURERCTL[n]	SMPU7 Region n Control Secure Accesses Register	0x00000000

Table A-130: ADSP-SC58x SMPU8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31099000	SMPU8_CTL	SMPU8 SMPU Control Register	0x00000000
0x31099004	SMPU8_STAT	SMPU8 SMPU Status Register	0x00000000
0x31099008	SMPU8_IADDR	SMPU8 Interrupt Address Register	0x00000000
0x3109900C	SMPU8_IDTLS	SMPU8 Interrupt Details Register	0x00000000
0x31099010	SMPU8_BADDR	SMPU8 Bus Error Address Register	0x00000000
0x31099014	SMPU8_BDTLS	SMPU8 Bus Error Details Register	0x00000000
0x31099020	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x31099024	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x31099028	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x3109902C	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x31099030	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x31099034	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099038	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000

Table A-130: ADSP-SC58x SMPU8 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109903C	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x31099040	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x31099044	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x31099048	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x3109904C	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099050	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x31099054	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x31099058	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x3109905C	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x31099060	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x31099064	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099068	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x3109906C	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x31099070	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x31099074	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x31099078	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x3109907C	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099080	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x31099084	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x31099088	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x3109908C	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x31099090	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x31099094	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099098	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x3109909C	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x310990A0	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x310990A4	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x310990A8	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x310990AC	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x310990B0	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x310990B4	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000

Table A-130: ADSP-SC58x SMPU8 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310990B8	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x310990BC	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x310990C0	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x310990C4	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x310990C8	SMPU8_RCTL[n]	SMPU8 Region n Control Register	0x00000000
0x310990CC	SMPU8_RADDR[n]	SMPU8 Region n Address Register	0x00000000
0x310990D0	SMPU8_RIDA[n]	SMPU8 Region n ID A Register	0x00000000
0x310990D4	SMPU8_RIDMSKA[n]	SMPU8 Region n ID Mask A Register	0x00000000
0x310990D8	SMPU8_RIDB[n]	SMPU8 Region n ID B Register	0x00000000
0x310990DC	SMPU8_RIDMSKB[n]	SMPU8 Region n ID Mask B Register	0x00000000
0x31099220	SMPU8_REVID	SMPU8 SMPU Revision ID Register	0x00000010
0x31099800	SMPU8_SECURECTL	SMPU8 SMPU Control Secure Accesses Register	0x00000000
0x31099820	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x31099824	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x31099828	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x3109982C	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x31099830	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x31099834	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x31099838	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000
0x3109983C	SMPU8_SECURERCTL[n]	SMPU8 Region n Control Secure Accesses Register	0x00000000

Table A-131: ADSP-SC58x SMPU9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310A0000	SMPU9_CTL	SMPU9 SMPU Control Register	0x00000000
0x310A0004	SMPU9_STAT	SMPU9 SMPU Status Register	0x00000000
0x310A0008	SMPU9_IADDR	SMPU9 Interrupt Address Register	0x00000000
0x310A000C	SMPU9_IDTLS	SMPU9 Interrupt Details Register	0x00000000
0x310A0010	SMPU9_BADDR	SMPU9 Bus Error Address Register	0x00000000
0x310A0014	SMPU9_BDTLS	SMPU9 Bus Error Details Register	0x00000000
0x310A0020	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000

Table A-131: ADSP-SC58x SMPU9 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A0024	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0028	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A002C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0030	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0034	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0038	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A003C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0040	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A0044	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0048	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A004C	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0050	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A0054	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0058	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A005C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0060	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0064	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0068	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A006C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0070	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A0074	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0078	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A007C	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0080	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A0084	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A0088	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A008C	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A0090	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A0094	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A0098	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A009C	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000

Table A-131: ADSP-SC58x SMPU9 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A00A0	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00A4	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00A8	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00AC	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A00B0	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A00B4	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A00B8	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00BC	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00C0	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00C4	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A00C8	SMPU9_RCTL[n]	SMPU9 Region n Control Register	0x00000000
0x310A00CC	SMPU9_RADDR[n]	SMPU9 Region n Address Register	0x00000000
0x310A00D0	SMPU9_RIDA[n]	SMPU9 Region n ID A Register	0x00000000
0x310A00D4	SMPU9_RIDMSKA[n]	SMPU9 Region n ID Mask A Register	0x00000000
0x310A00D8	SMPU9_RIDB[n]	SMPU9 Region n ID B Register	0x00000000
0x310A00DC	SMPU9_RIDMSKB[n]	SMPU9 Region n ID Mask B Register	0x00000000
0x310A01A0	SMPU9_EXACADD[n]	SMPU9 Exclusive Access IDn Address	0x00000000
0x310A01A4	SMPU9_EXACSTAT[n]	SMPU9 Exclusive Access Status	0x00000000
0x310A01A8	SMPU9_EXACADD[n]	SMPU9 Exclusive Access IDn Address	0x00000000
0x310A01AC	SMPU9_EXACSTAT[n]	SMPU9 Exclusive Access Status	0x00000000
0x310A01B0	SMPU9_EXACADD[n]	SMPU9 Exclusive Access IDn Address	0x00000000
0x310A01B4	SMPU9_EXACSTAT[n]	SMPU9 Exclusive Access Status	0x00000000
0x310A01B8	SMPU9_EXACADD[n]	SMPU9 Exclusive Access IDn Address	0x00000000
0x310A01BC	SMPU9_EXACSTAT[n]	SMPU9 Exclusive Access Status	0x00000000
0x310A0220	SMPU9_REVID	SMPU9 SMPU Revision ID Register	0x00000010
0x310A0800	SMPU9_SECURECTL	SMPU9 SMPU Control Secure Accesses Register	0x00000000
0x310A0820	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0824	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0828	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A082C	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0830	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000

Table A-131: ADSP-SC58x SMPU9 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310A0834	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A0838	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000
0x310A083C	SMPU9_SECURERCTL[n]	SMPU9 Region n Control Secure Accesses Register	0x00000000

Table A-132: ADSP-SC58x SPDIF0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9280	SPDIF0_TX_CTL	SPDIF0 Transmit Control Register	0x00000000
0x310C9284	SPDIF0_TX_STAT_A0	SPDIF0 Transmit Status A0 Register	0x00000000
0x310C9288	SPDIF0_TX_STAT_B0	SPDIF0 Transmit Status B0 Register	0x00000000
0x310C92A0	SPDIF0_RX_CTL	SPDIF0 Receive Control	0x00000018
0x310C92A4	SPDIF0_RX_STAT	SPDIF0 Receive Status Register	0x00000000
0x310C92A8	SPDIF0_RX_STAT0_A	SPDIF0 Receive Status A0 Register	0x00000000
0x310C92AC	SPDIF0_RX_STAT0_B	SPDIF0 Receive Status B0 Register	0x00000000
0x310C92B0	SPDIF0_RX_STAT1_A	SPDIF0 Receive Status A1 Register	0x00000000
0x310C92B4	SPDIF0_RX_STAT1_B	SPDIF0 Receive Status B1 Register	0x00000000
0x310C9350	SPDIF0_TX_STAT_A1	SPDIF0 Transmit Status A1 Register	0x00000000
0x310C9354	SPDIF0_TX_STAT_A2	SPDIF0 Transmit Status A2 Register	0x00000000
0x310C9358	SPDIF0_TX_STAT_A3	SPDIF0 Transmit Status A3 Register	0x00000000
0x310C935C	SPDIF0_TX_STAT_A4	SPDIF0 Transmit Status A4 Register	0x00000000
0x310C9360	SPDIF0_TX_STAT_A5	SPDIF0 Transmit Status A5 Register	0x00000000
0x310C9368	SPDIF0_TX_STAT_B1	SPDIF0 Transmit Status B1 Register	0x00000000
0x310C936C	SPDIF0_TX_STAT_B2	SPDIF0 Transmit Status B2 Register	0x00000000
0x310C9370	SPDIF0_TX_STAT_B3	SPDIF0 Transmit Status B3 Register	0x00000000
0x310C9374	SPDIF0_TX_STAT_B4	SPDIF0 Transmit Status B4 Register	0x00000000
0x310C9378	SPDIF0_TX_STAT_B5	SPDIF0 Transmit Status B5 Register	0x00000000
0x310C9380	SPDIF0_TX_UBUFF_A0	SPDIF0 Transmit User Buffer A0 Register	0x00000000
0x310C9384	SPDIF0_TX_UBUFF_A1	SPDIF0 Transmit User Buffer A1 Register	0x00000000
0x310C9388	SPDIF0_TX_UBUFF_A2	SPDIF0 Transmit User Buffer A2 Register	0x00000000
0x310C938C	SPDIF0_TX_UBUFF_A3	SPDIF0 Transmit User Buffer A3 Register	0x00000000
0x310C9390	SPDIF0_TX_UBUFF_A4	SPDIF0 Transmit User Buffer A4 Register	0x00000000

Table A-132: ADSP-SC58x SPDIF0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C9394	SPDIF0_TX_UBUFF_A5	SPDIF0 Transmit User Buffer A5 Register	0x00000000
0x310C93A0	SPDIF0_TX_UBUFF_B0	SPDIF0 Transmit User Buffer B0 Register	0x00000000
0x310C93A4	SPDIF0_TX_UBUFF_B1	SPDIF0 Transmit User Buffer B1 Register	0x00000000
0x310C93A8	SPDIF0_TX_UBUFF_B2	SPDIF0 Transmit User Buffer B2 Register	0x00000000
0x310C93AC	SPDIF0_TX_UBUFF_B3	SPDIF0 Transmit User Buffer B3 Register	0x00000000
0x310C93B0	SPDIF0_TX_UBUFF_B4	SPDIF0 Transmit User Buffer B4 Register	0x00000000
0x310C93B4	SPDIF0_TX_UBUFF_B5	SPDIF0 Transmit User Buffer B5 Register	0x00000000
0x310C93BC	SPDIF0_TX_USRUPDT	SPDIF0 User Bit Update Register	0x00000000

Table A-133: ADSP-SC58x SPDIF1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB280	SPDIF1_TX_CTL	SPDIF1 Transmit Control Register	0x00000000
0x310CB284	SPDIF1_TX_STAT_A0	SPDIF1 Transmit Status A0 Register	0x00000000
0x310CB288	SPDIF1_TX_STAT_B0	SPDIF1 Transmit Status B0 Register	0x00000000
0x310CB2A0	SPDIF1_RX_CTL	SPDIF1 Receive Control	0x00000018
0x310CB2A4	SPDIF1_RX_STAT	SPDIF1 Receive Status Register	0x00000000
0x310CB2A8	SPDIF1_RX_STAT0_A	SPDIF1 Receive Status A0 Register	0x00000000
0x310CB2AC	SPDIF1_RX_STAT0_B	SPDIF1 Receive Status B0 Register	0x00000000
0x310CB2B0	SPDIF1_RX_STAT1_A	SPDIF1 Receive Status A1 Register	0x00000000
0x310CB2B4	SPDIF1_RX_STAT1_B	SPDIF1 Receive Status B1 Register	0x00000000
0x310CB350	SPDIF1_TX_STAT_A1	SPDIF1 Transmit Status A1 Register	0x00000000
0x310CB354	SPDIF1_TX_STAT_A2	SPDIF1 Transmit Status A2 Register	0x00000000
0x310CB358	SPDIF1_TX_STAT_A3	SPDIF1 Transmit Status A3 Register	0x00000000
0x310CB35C	SPDIF1_TX_STAT_A4	SPDIF1 Transmit Status A4 Register	0x00000000
0x310CB360	SPDIF1_TX_STAT_A5	SPDIF1 Transmit Status A5 Register	0x00000000
0x310CB368	SPDIF1_TX_STAT_B1	SPDIF1 Transmit Status B1 Register	0x00000000
0x310CB36C	SPDIF1_TX_STAT_B2	SPDIF1 Transmit Status B2 Register	0x00000000
0x310CB370	SPDIF1_TX_STAT_B3	SPDIF1 Transmit Status B3 Register	0x00000000
0x310CB374	SPDIF1_TX_STAT_B4	SPDIF1 Transmit Status B4 Register	0x00000000
0x310CB378	SPDIF1_TX_STAT_B5	SPDIF1 Transmit Status B5 Register	0x00000000

Table A-133: ADSP-SC58x SPDIF1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310CB380	SPDIF1_TX_UBUFF_A0	SPDIF1 Transmit User Buffer A0 Register	0x00000000
0x310CB384	SPDIF1_TX_UBUFF_A1	SPDIF1 Transmit User Buffer A1 Register	0x00000000
0x310CB388	SPDIF1_TX_UBUFF_A2	SPDIF1 Transmit User Buffer A2 Register	0x00000000
0x310CB38C	SPDIF1_TX_UBUFF_A3	SPDIF1 Transmit User Buffer A3 Register	0x00000000
0x310CB390	SPDIF1_TX_UBUFF_A4	SPDIF1 Transmit User Buffer A4 Register	0x00000000
0x310CB394	SPDIF1_TX_UBUFF_A5	SPDIF1 Transmit User Buffer A5 Register	0x00000000
0x310CB3A0	SPDIF1_TX_UBUFF_B0	SPDIF1 Transmit User Buffer B0 Register	0x00000000
0x310CB3A4	SPDIF1_TX_UBUFF_B1	SPDIF1 Transmit User Buffer B1 Register	0x00000000
0x310CB3A8	SPDIF1_TX_UBUFF_B2	SPDIF1 Transmit User Buffer B2 Register	0x00000000
0x310CB3AC	SPDIF1_TX_UBUFF_B3	SPDIF1 Transmit User Buffer B3 Register	0x00000000
0x310CB3B0	SPDIF1_TX_UBUFF_B4	SPDIF1 Transmit User Buffer B4 Register	0x00000000
0x310CB3B4	SPDIF1_TX_UBUFF_B5	SPDIF1 Transmit User Buffer B5 Register	0x00000000
0x310CB3BC	SPDIF1_TX_USRUPDT	SPDIF1 User Bit Update Register	0x00000000

Table A-134: ADSP-SC58x SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31042004	SPI0_CTL	SPI0 Control Register	0x00000050
0x31042008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x3104200C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000
0x31042010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x31042014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x31042018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x3104201C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x31042020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x31042024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x31042028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x31042030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x31042034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x31042038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x31042040	SPI0_STAT	SPI0 Status Register	0x00440001

Table A-134: ADSP-SC58x SPI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31042044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x31042048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x31042050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000
0x31042058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000

Table A-135: ADSP-SC58x SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31043004	SPI1_CTL	SPI1 Control Register	0x00000050
0x31043008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x3104300C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x31043010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x31043014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x31043018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x3104301C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000
0x31043020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x31043024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x31043028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x31043030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x31043034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000
0x31043038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x31043040	SPI1_STAT	SPI1 Status Register	0x00440001
0x31043044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x31043048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x31043050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x31043058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000

Table A-136: ADSP-SC58x SPI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31044004	SPI2_CTL	SPI2 Control Register	0x00000050
0x31044008	SPI2_RXCTL	SPI2 Receive Control Register	0x00000000

Table A-136: ADSP-SC58x SPI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3104400C	SPI2_TXCTL	SPI2 Transmit Control Register	0x00000000
0x31044010	SPI2_CLK	SPI2 Clock Rate Register	0x00000000
0x31044014	SPI2_DLY	SPI2 Delay Register	0x00000301
0x31044018	SPI2_SLVSEL	SPI2 Slave Select Register	0x0000FE00
0x3104401C	SPI2_RWC	SPI2 Received Word Count Register	0x00000000
0x31044020	SPI2_RWCR	SPI2 Received Word Count Reload Register	0x00000000
0x31044024	SPI2_TWC	SPI2 Transmitted Word Count Register	0x00000000
0x31044028	SPI2_TWCR	SPI2 Transmitted Word Count Reload Register	0x00000000
0x31044030	SPI2_IMSK	SPI2 Interrupt Mask Register	0x00000000
0x31044034	SPI2_IMSK_CLR	SPI2 Interrupt Mask Clear Register	0x00000000
0x31044038	SPI2_IMSK_SET	SPI2 Interrupt Mask Set Register	0x00000000
0x31044040	SPI2_STAT	SPI2 Status Register	0x00440001
0x31044044	SPI2_ILAT	SPI2 Masked Interrupt Condition Register	0x00000000
0x31044048	SPI2_ILAT_CLR	SPI2 Masked Interrupt Clear Register	0x00000000
0x31044050	SPI2_RFIFO	SPI2 Receive FIFO Data Register	0x00000000
0x31044058	SPI2_TFIFO	SPI2 Transmit FIFO Data Register	0x00000000
0x31044060	SPI2_MMRDH	SPI2 Memory Mapped Read Header (Only on SPI2)	0x00000000
0x31044064	SPI2_MMTOP	SPI2 SPI Memory Top Address (Only on SPI2)	0x00000000

Table A-137: ADSP-SC58x SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x31002004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x31002008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100200C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000

Table A-137: ADSP-SC58x SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000
0x31002024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x31002040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100204C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x31002084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x31002088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100208C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310020A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x310020A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310020A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x310020C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310020C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310020C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310020CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-138: ADSP-SC58x SPORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002100	SPORT1_CTL_A	SPORT1 Half SPORT 'A' Control Register	0x00000000
0x31002104	SPORT1_DIV_A	SPORT1 Half SPORT 'A' Divisor Register	0x00000000
0x31002108	SPORT1_MCTL_A	SPORT1 Half SPORT 'A' Multichannel Control Register	0x00000000

Table A-138: ADSP-SC58x SPORT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100210C	SPORT1_CS0_A	SPORT1 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002110	SPORT1_CS1_A	SPORT1 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002114	SPORT1_CS2_A	SPORT1 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002118	SPORT1_CS3_A	SPORT1 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002120	SPORT1_ERR_A	SPORT1 Half SPORT 'A' Error Register	0x00000000
0x31002124	SPORT1_MSTAT_A	SPORT1 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002128	SPORT1_CTL2_A	SPORT1 Half SPORT 'A' Control 2 Register	0x00000000
0x31002140	SPORT1_TXPRI_A	SPORT1 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002144	SPORT1_RXPRI_A	SPORT1 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002148	SPORT1_TXSEC_A	SPORT1 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100214C	SPORT1_RXSEC_A	SPORT1 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002180	SPORT1_CTL_B	SPORT1 Half SPORT 'B' Control Register	0x00000000
0x31002184	SPORT1_DIV_B	SPORT1 Half SPORT 'B' Divisor Register	0x00000000
0x31002188	SPORT1_MCTL_B	SPORT1 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100218C	SPORT1_CS0_B	SPORT1 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002190	SPORT1_CS1_B	SPORT1 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002194	SPORT1_CS2_B	SPORT1 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002198	SPORT1_CS3_B	SPORT1 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310021A0	SPORT1_ERR_B	SPORT1 Half SPORT 'B' Error Register	0x00000000
0x310021A4	SPORT1_MSTAT_B	SPORT1 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310021A8	SPORT1_CTL2_B	SPORT1 Half SPORT 'B' Control 2 Register	0x00000000
0x310021C0	SPORT1_TXPRI_B	SPORT1 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310021C4	SPORT1_RXPRI_B	SPORT1 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310021C8	SPORT1_TXSEC_B	SPORT1 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310021CC	SPORT1_RXSEC_B	SPORT1 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-139: ADSP-SC58x SPORT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002200	SPORT2_CTL_A	SPORT2 Half SPORT 'A' Control Register	0x00000000
0x31002204	SPORT2_DIV_A	SPORT2 Half SPORT 'A' Divisor Register	0x00000000
0x31002208	SPORT2_MCTL_A	SPORT2 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100220C	SPORT2_CS0_A	SPORT2 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002210	SPORT2_CS1_A	SPORT2 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002214	SPORT2_CS2_A	SPORT2 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002218	SPORT2_CS3_A	SPORT2 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002220	SPORT2_ERR_A	SPORT2 Half SPORT 'A' Error Register	0x00000000
0x31002224	SPORT2_MSTAT_A	SPORT2 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002228	SPORT2_CTL2_A	SPORT2 Half SPORT 'A' Control 2 Register	0x00000000
0x31002240	SPORT2_TXPRI_A	SPORT2 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002244	SPORT2_RXPRI_A	SPORT2 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002248	SPORT2_TXSEC_A	SPORT2 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100224C	SPORT2_RXSEC_A	SPORT2 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002280	SPORT2_CTL_B	SPORT2 Half SPORT 'B' Control Register	0x00000000
0x31002284	SPORT2_DIV_B	SPORT2 Half SPORT 'B' Divisor Register	0x00000000
0x31002288	SPORT2_MCTL_B	SPORT2 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100228C	SPORT2_CS0_B	SPORT2 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002290	SPORT2_CS1_B	SPORT2 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002294	SPORT2_CS2_B	SPORT2 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002298	SPORT2_CS3_B	SPORT2 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310022A0	SPORT2_ERR_B	SPORT2 Half SPORT 'B' Error Register	0x00000000
0x310022A4	SPORT2_MSTAT_B	SPORT2 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310022A8	SPORT2_CTL2_B	SPORT2 Half SPORT 'B' Control 2 Register	0x00000000
0x310022C0	SPORT2_TXPRI_B	SPORT2 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000

Table A-139: ADSP-SC58x SPORT2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310022C4	SPORT2_RXPRI_B	SPORT2 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310022C8	SPORT2_TXSEC_B	SPORT2 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310022CC	SPORT2_RXSEC_B	SPORT2 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-140: ADSP-SC58x SPORT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002300	SPORT3_CTL_A	SPORT3 Half SPORT 'A' Control Register	0x00000000
0x31002304	SPORT3_DIV_A	SPORT3 Half SPORT 'A' Divisor Register	0x00000000
0x31002308	SPORT3_MCTL_A	SPORT3 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100230C	SPORT3_CS0_A	SPORT3 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002310	SPORT3_CS1_A	SPORT3 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002314	SPORT3_CS2_A	SPORT3 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002318	SPORT3_CS3_A	SPORT3 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002320	SPORT3_ERR_A	SPORT3 Half SPORT 'A' Error Register	0x00000000
0x31002324	SPORT3_MSTAT_A	SPORT3 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002328	SPORT3_CTL2_A	SPORT3 Half SPORT 'A' Control 2 Register	0x00000000
0x31002340	SPORT3_TXPRI_A	SPORT3 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002344	SPORT3_RXPRI_A	SPORT3 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002348	SPORT3_TXSEC_A	SPORT3 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100234C	SPORT3_RXSEC_A	SPORT3 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002380	SPORT3_CTL_B	SPORT3 Half SPORT 'B' Control Register	0x00000000
0x31002384	SPORT3_DIV_B	SPORT3 Half SPORT 'B' Divisor Register	0x00000000
0x31002388	SPORT3_MCTL_B	SPORT3 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100238C	SPORT3_CS0_B	SPORT3 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002390	SPORT3_CS1_B	SPORT3 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002394	SPORT3_CS2_B	SPORT3 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000

Table A-140: ADSP-SC58x SPORT3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002398	SPORT3_CS3_B	SPORT3 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310023A0	SPORT3_ERR_B	SPORT3 Half SPORT 'B' Error Register	0x00000000
0x310023A4	SPORT3_MSTAT_B	SPORT3 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310023A8	SPORT3_CTL2_B	SPORT3 Half SPORT 'B' Control 2 Register	0x00000000
0x310023C0	SPORT3_TXPRI_B	SPORT3 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310023C4	SPORT3_RXPRI_B	SPORT3 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310023C8	SPORT3_TXSEC_B	SPORT3 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310023CC	SPORT3_RXSEC_B	SPORT3 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-141: ADSP-SC58x SPORT4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002400	SPORT4_CTL_A	SPORT4 Half SPORT 'A' Control Register	0x00000000
0x31002404	SPORT4_DIV_A	SPORT4 Half SPORT 'A' Divisor Register	0x00000000
0x31002408	SPORT4_MCTL_A	SPORT4 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100240C	SPORT4_CS0_A	SPORT4 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002410	SPORT4_CS1_A	SPORT4 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002414	SPORT4_CS2_A	SPORT4 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002418	SPORT4_CS3_A	SPORT4 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002420	SPORT4_ERR_A	SPORT4 Half SPORT 'A' Error Register	0x00000000
0x31002424	SPORT4_MSTAT_A	SPORT4 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002428	SPORT4_CTL2_A	SPORT4 Half SPORT 'A' Control 2 Register	0x00000000
0x31002440	SPORT4_TXPRI_A	SPORT4 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002444	SPORT4_RXPRI_A	SPORT4 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002448	SPORT4_TXSEC_A	SPORT4 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100244C	SPORT4_RXSEC_A	SPORT4 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002480	SPORT4_CTL_B	SPORT4 Half SPORT 'B' Control Register	0x00000000
0x31002484	SPORT4_DIV_B	SPORT4 Half SPORT 'B' Divisor Register	0x00000000

Table A-141: ADSP-SC58x SPORT4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002488	SPORT4_MCTL_B	SPORT4 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100248C	SPORT4_CS0_B	SPORT4 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002490	SPORT4_CS1_B	SPORT4 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002494	SPORT4_CS2_B	SPORT4 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002498	SPORT4_CS3_B	SPORT4 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310024A0	SPORT4_ERR_B	SPORT4 Half SPORT 'B' Error Register	0x00000000
0x310024A4	SPORT4_MSTAT_B	SPORT4 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310024A8	SPORT4_CTL2_B	SPORT4 Half SPORT 'B' Control 2 Register	0x00000000
0x310024C0	SPORT4_TXPRI_B	SPORT4 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310024C4	SPORT4_RXPRI_B	SPORT4 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310024C8	SPORT4_TXSEC_B	SPORT4 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310024CC	SPORT4_RXSEC_B	SPORT4 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-142: ADSP-SC58x SPORT5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002500	SPORT5_CTL_A	SPORT5 Half SPORT 'A' Control Register	0x00000000
0x31002504	SPORT5_DIV_A	SPORT5 Half SPORT 'A' Divisor Register	0x00000000
0x31002508	SPORT5_MCTL_A	SPORT5 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100250C	SPORT5_CS0_A	SPORT5 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002510	SPORT5_CS1_A	SPORT5 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002514	SPORT5_CS2_A	SPORT5 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002518	SPORT5_CS3_A	SPORT5 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002520	SPORT5_ERR_A	SPORT5 Half SPORT 'A' Error Register	0x00000000
0x31002524	SPORT5_MSTAT_A	SPORT5 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002528	SPORT5_CTL2_A	SPORT5 Half SPORT 'A' Control 2 Register	0x00000000

Table A-142: ADSP-SC58x SPORT5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002540	SPORT5_TXPRI_A	SPORT5 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002544	SPORT5_RXPRI_A	SPORT5 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002548	SPORT5_TXSEC_A	SPORT5 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100254C	SPORT5_RXSEC_A	SPORT5 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002580	SPORT5_CTL_B	SPORT5 Half SPORT 'B' Control Register	0x00000000
0x31002584	SPORT5_DIV_B	SPORT5 Half SPORT 'B' Divisor Register	0x00000000
0x31002588	SPORT5_MCTL_B	SPORT5 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100258C	SPORT5_CS0_B	SPORT5 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002590	SPORT5_CS1_B	SPORT5 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002594	SPORT5_CS2_B	SPORT5 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002598	SPORT5_CS3_B	SPORT5 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310025A0	SPORT5_ERR_B	SPORT5 Half SPORT 'B' Error Register	0x00000000
0x310025A4	SPORT5_MSTAT_B	SPORT5 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310025A8	SPORT5_CTL2_B	SPORT5 Half SPORT 'B' Control 2 Register	0x00000000
0x310025C0	SPORT5_TXPRI_B	SPORT5 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310025C4	SPORT5_RXPRI_B	SPORT5 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310025C8	SPORT5_TXSEC_B	SPORT5 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310025CC	SPORT5_RXSEC_B	SPORT5 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-143: ADSP-SC58x SPORT6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002600	SPORT6_CTL_A	SPORT6 Half SPORT 'A' Control Register	0x00000000
0x31002604	SPORT6_DIV_A	SPORT6 Half SPORT 'A' Divisor Register	0x00000000
0x31002608	SPORT6_MCTL_A	SPORT6 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100260C	SPORT6_CS0_A	SPORT6 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002610	SPORT6_CS1_A	SPORT6 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000

Table A-143: ADSP-SC58x SPORT6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31002614	SPORT6_CS2_A	SPORT6 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002618	SPORT6_CS3_A	SPORT6 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002620	SPORT6_ERR_A	SPORT6 Half SPORT 'A' Error Register	0x00000000
0x31002624	SPORT6_MSTAT_A	SPORT6 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002628	SPORT6_CTL2_A	SPORT6 Half SPORT 'A' Control 2 Register	0x00000000
0x31002640	SPORT6_TXPRI_A	SPORT6 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002644	SPORT6_RXPRI_A	SPORT6 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002648	SPORT6_TXSEC_A	SPORT6 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100264C	SPORT6_RXSEC_A	SPORT6 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002680	SPORT6_CTL_B	SPORT6 Half SPORT 'B' Control Register	0x00000000
0x31002684	SPORT6_DIV_B	SPORT6 Half SPORT 'B' Divisor Register	0x00000000
0x31002688	SPORT6_MCTL_B	SPORT6 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100268C	SPORT6_CS0_B	SPORT6 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002690	SPORT6_CS1_B	SPORT6 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002694	SPORT6_CS2_B	SPORT6 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002698	SPORT6_CS3_B	SPORT6 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310026A0	SPORT6_ERR_B	SPORT6 Half SPORT 'B' Error Register	0x00000000
0x310026A4	SPORT6_MSTAT_B	SPORT6 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310026A8	SPORT6_CTL2_B	SPORT6 Half SPORT 'B' Control 2 Register	0x00000000
0x310026C0	SPORT6_TXPRI_B	SPORT6 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x310026C4	SPORT6_RXPRI_B	SPORT6 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310026C8	SPORT6_TXSEC_B	SPORT6 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310026CC	SPORT6_RXSEC_B	SPORT6 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-144: ADSP-SC58x SPORT7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31002700	SPORT7_CTL_A	SPORT7 Half SPORT 'A' Control Register	0x00000000
0x31002704	SPORT7_DIV_A	SPORT7 Half SPORT 'A' Divisor Register	0x00000000
0x31002708	SPORT7_MCTL_A	SPORT7 Half SPORT 'A' Multichannel Control Register	0x00000000
0x3100270C	SPORT7_CS0_A	SPORT7 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x31002710	SPORT7_CS1_A	SPORT7 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x31002714	SPORT7_CS2_A	SPORT7 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x31002718	SPORT7_CS3_A	SPORT7 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x31002720	SPORT7_ERR_A	SPORT7 Half SPORT 'A' Error Register	0x00000000
0x31002724	SPORT7_MSTAT_A	SPORT7 Half SPORT 'A' Multichannel Status Register	0x00000000
0x31002728	SPORT7_CTL2_A	SPORT7 Half SPORT 'A' Control 2 Register	0x00000000
0x31002740	SPORT7_TXPRI_A	SPORT7 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x31002744	SPORT7_RXPRI_A	SPORT7 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x31002748	SPORT7_TXSEC_A	SPORT7 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x3100274C	SPORT7_RXSEC_A	SPORT7 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x31002780	SPORT7_CTL_B	SPORT7 Half SPORT 'B' Control Register	0x00000000
0x31002784	SPORT7_DIV_B	SPORT7 Half SPORT 'B' Divisor Register	0x00000000
0x31002788	SPORT7_MCTL_B	SPORT7 Half SPORT 'B' Multichannel Control Register	0x00000000
0x3100278C	SPORT7_CS0_B	SPORT7 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x31002790	SPORT7_CS1_B	SPORT7 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x31002794	SPORT7_CS2_B	SPORT7 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x31002798	SPORT7_CS3_B	SPORT7 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x310027A0	SPORT7_ERR_B	SPORT7 Half SPORT 'B' Error Register	0x00000000
0x310027A4	SPORT7_MSTAT_B	SPORT7 Half SPORT 'B' Multichannel Status Register	0x00000000
0x310027A8	SPORT7_CTL2_B	SPORT7 Half SPORT 'B' Control 2 Register	0x00000000
0x310027C0	SPORT7_TXPRI_B	SPORT7 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000

Table A-144: ADSP-SC58x SPORT7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310027C4	SPORT7_RXPRI_B	SPORT7 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x310027C8	SPORT7_TXSEC_B	SPORT7 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x310027CC	SPORT7_RXSEC_B	SPORT7 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x3108C004	SPU0_STAT	SPU0 Status Register	0x00000000
0x3108C400	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C404	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C408	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C40C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C410	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C414	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C418	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C41C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C420	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C424	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C428	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C42C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C430	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C434	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C438	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C43C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C440	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C444	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C448	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C44C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C450	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C454	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C458	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C45C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C460	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C464	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C468	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C46C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C470	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C474	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C478	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C47C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C480	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C484	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C488	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C48C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C490	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C494	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C498	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C49C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C4D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4F4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4F8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C4FC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C500	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C504	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C508	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C50C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C510	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C514	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C518	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C51C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C520	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C524	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C528	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C52C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C530	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C534	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C538	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C53C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C540	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C544	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C548	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C54C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C550	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C554	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C558	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C55C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C560	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C564	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C568	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C56C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C570	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C574	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C578	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C57C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C580	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C584	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C588	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C58C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C590	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C594	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C598	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C59C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C5CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5F4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5F8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C5FC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C600	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C604	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C608	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C60C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C610	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C614	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C618	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C61C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C620	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C624	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C628	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C62C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C630	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C634	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C638	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C63C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C640	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C644	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C648	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C64C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C650	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C654	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C658	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C65C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C660	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C664	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C668	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C66C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C670	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C674	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C678	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C67C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C680	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C684	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C688	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C68C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C690	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C694	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C698	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C69C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108C6C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C6F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x3108C840	SPU0_SECURECTL	SPU0 Secure Control Register	0x00000000
0x3108C84C	SPU0_SECURECHK	SPU0 Secure Check Register	0xFFFFFFFF
0x3108C980	SPU0_SECUREC[n]	SPU0 Secure Core Registers	0x00000001
0x3108C984	SPU0_SECUREC[n]	SPU0 Secure Core Registers	0x00000001
0x3108C988	SPU0_SECUREC[n]	SPU0 Secure Core Registers	0x00000001
0x3108CA00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CA38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CA9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CAB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CABC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CACC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CADC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAF4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAF8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CAFC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CB30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CB9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CBAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBBC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBCC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBDC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBF4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBF8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CBFC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CC28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CC9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-145: ADSP-SC58x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108CCA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCBC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCCC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCDC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x3108CCF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-146: ADSP-SC58x SWU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31015000	SWU0_GCTL	SWU0 Global Control Register	0x00000000
0x31015004	SWU0_GSTAT	SWU0 Global Status Register	0x00000000
0x31015010	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x31015014	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x31015018	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x3101501C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x31015020	SWU0_CNT[n]	SWU0 Count Register n	0x00000000

Table A-146: ADSP-SC58x SWU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31015024	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x31015028	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x3101502C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x31015030	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x31015034	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x31015038	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x3101503C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x31015040	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x31015044	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x31015048	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x3101504C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x31015050	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x31015054	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x31015058	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x3101505C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x31015060	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x31015064	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x31015068	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x3101506C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x31015070	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x31015074	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x31015078	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x3101507C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x31015080	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x31015084	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x31015088	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x3101508C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000

Table A-147: ADSP-SC58x SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31091000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x31091004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x31091010	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31091014	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31091018	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109101C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31091020	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31091024	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31091028	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109102C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31091030	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31091034	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31091038	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109103C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31091040	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31091044	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31091048	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109104C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31091050	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31091054	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31091058	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109105C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31091060	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x31091064	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31091068	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109106C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x31091070	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x31091074	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x31091078	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x3109107C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x31091080	SWU1_CNT[n]	SWU1 Count Register n	0x00000000

Table A-147: ADSP-SC58x SWU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31091084	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x31091088	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x3109108C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000

Table A-148: ADSP-SC58x SWU10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31143000	SWU10_GCTL	SWU10 Global Control Register	0x00000000
0x31143004	SWU10_GSTAT	SWU10 Global Status Register	0x00000000
0x31143010	SWU10_CTL[n]	SWU10 Control Register n	0x00000000
0x31143014	SWU10_LA[n]	SWU10 Lower Address Register n	0x00000000
0x31143018	SWU10_UA[n]	SWU10 Upper Address Register n	0x00000000
0x3114301C	SWU10_ID[n]	SWU10 ID Register n	0x00000000
0x31143020	SWU10_CNT[n]	SWU10 Count Register n	0x00000000
0x31143024	SWU10_TARG[n]	SWU10 Target Register n	0x00000000
0x31143028	SWU10_HIST[n]	SWU10 Bandwidth History Register n	0x00000000
0x3114302C	SWU10_CUR[n]	SWU10 Current Register n	0x00000000
0x31143030	SWU10_CTL[n]	SWU10 Control Register n	0x00000000
0x31143034	SWU10_LA[n]	SWU10 Lower Address Register n	0x00000000
0x31143038	SWU10_UA[n]	SWU10 Upper Address Register n	0x00000000
0x3114303C	SWU10_ID[n]	SWU10 ID Register n	0x00000000
0x31143040	SWU10_CNT[n]	SWU10 Count Register n	0x00000000
0x31143044	SWU10_TARG[n]	SWU10 Target Register n	0x00000000
0x31143048	SWU10_HIST[n]	SWU10 Bandwidth History Register n	0x00000000
0x3114304C	SWU10_CUR[n]	SWU10 Current Register n	0x00000000
0x31143050	SWU10_CTL[n]	SWU10 Control Register n	0x00000000
0x31143054	SWU10_LA[n]	SWU10 Lower Address Register n	0x00000000
0x31143058	SWU10_UA[n]	SWU10 Upper Address Register n	0x00000000
0x3114305C	SWU10_ID[n]	SWU10 ID Register n	0x00000000
0x31143060	SWU10_CNT[n]	SWU10 Count Register n	0x00000000
0x31143064	SWU10_TARG[n]	SWU10 Target Register n	0x00000000

Table A-148: ADSP-SC58x SWU10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31143068	SWU10_HIST[n]	SWU10 Bandwidth History Register n	0x00000000
0x3114306C	SWU10_CUR[n]	SWU10 Current Register n	0x00000000
0x31143070	SWU10_CTL[n]	SWU10 Control Register n	0x00000000
0x31143074	SWU10_LA[n]	SWU10 Lower Address Register n	0x00000000
0x31143078	SWU10_UA[n]	SWU10 Upper Address Register n	0x00000000
0x3114307C	SWU10_ID[n]	SWU10 ID Register n	0x00000000
0x31143080	SWU10_CNT[n]	SWU10 Count Register n	0x00000000
0x31143084	SWU10_TARG[n]	SWU10 Target Register n	0x00000000
0x31143088	SWU10_HIST[n]	SWU10 Bandwidth History Register n	0x00000000
0x3114308C	SWU10_CUR[n]	SWU10 Current Register n	0x00000000

Table A-149: ADSP-SC58x SWU11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31097000	SWU11_GCTL	SWU11 Global Control Register	0x00000000
0x31097004	SWU11_GSTAT	SWU11 Global Status Register	0x00000000
0x31097010	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097014	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097018	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109701C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097020	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097024	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097028	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109702C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097030	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097034	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097038	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109703C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097040	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097044	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097048	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000

Table A-149: ADSP-SC58x SWU11 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109704C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097050	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097054	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097058	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109705C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097060	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097064	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097068	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109706C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000
0x31097070	SWU11_CTL[n]	SWU11 Control Register n	0x00000000
0x31097074	SWU11_LA[n]	SWU11 Lower Address Register n	0x00000000
0x31097078	SWU11_UA[n]	SWU11 Upper Address Register n	0x00000000
0x3109707C	SWU11_ID[n]	SWU11 ID Register n	0x00000000
0x31097080	SWU11_CNT[n]	SWU11 Count Register n	0x00000000
0x31097084	SWU11_TARG[n]	SWU11 Target Register n	0x00000000
0x31097088	SWU11_HIST[n]	SWU11 Bandwidth History Register n	0x00000000
0x3109708C	SWU11_CUR[n]	SWU11 Current Register n	0x00000000

Table A-150: ADSP-SC58x SWU12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31041000	SWU12_GCTL	SWU12 Global Control Register	0x00000000
0x31041004	SWU12_GSTAT	SWU12 Global Status Register	0x00000000
0x31041010	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x31041014	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x31041018	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x3104101C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x31041020	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x31041024	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x31041028	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x3104102C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000

Table A-150: ADSP-SC58x SWU12 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31041030	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x31041034	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x31041038	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x3104103C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x31041040	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x31041044	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x31041048	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x3104104C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000
0x31041050	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x31041054	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x31041058	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x3104105C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x31041060	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x31041064	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x31041068	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x3104106C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000
0x31041070	SWU12_CTL[n]	SWU12 Control Register n	0x00000000
0x31041074	SWU12_LA[n]	SWU12 Lower Address Register n	0x00000000
0x31041078	SWU12_UA[n]	SWU12 Upper Address Register n	0x00000000
0x3104107C	SWU12_ID[n]	SWU12 ID Register n	0x00000000
0x31041080	SWU12_CNT[n]	SWU12 Count Register n	0x00000000
0x31041084	SWU12_TARG[n]	SWU12 Target Register n	0x00000000
0x31041088	SWU12_HIST[n]	SWU12 Bandwidth History Register n	0x00000000
0x3104108C	SWU12_CUR[n]	SWU12 Current Register n	0x00000000

Table A-151: ADSP-SC58x SWU13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109E000	SWU13_GCTL	SWU13 Global Control Register	0x00000000
0x3109E004	SWU13_GSTAT	SWU13 Global Status Register	0x00000000
0x3109E010	SWU13_CTL[n]	SWU13 Control Register n	0x00000000

Table A-151: ADSP-SC58x SWU13 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109E014	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E018	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E01C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E020	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E024	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E028	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E02C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E030	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E034	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E038	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E03C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E040	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E044	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E048	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E04C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E050	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E054	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E058	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E05C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E060	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E064	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E068	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E06C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000
0x3109E070	SWU13_CTL[n]	SWU13 Control Register n	0x00000000
0x3109E074	SWU13_LA[n]	SWU13 Lower Address Register n	0x00000000
0x3109E078	SWU13_UA[n]	SWU13 Upper Address Register n	0x00000000
0x3109E07C	SWU13_ID[n]	SWU13 ID Register n	0x00000000
0x3109E080	SWU13_CNT[n]	SWU13 Count Register n	0x00000000
0x3109E084	SWU13_TARG[n]	SWU13 Target Register n	0x00000000
0x3109E088	SWU13_HIST[n]	SWU13 Bandwidth History Register n	0x00000000
0x3109E08C	SWU13_CUR[n]	SWU13 Current Register n	0x00000000

Table A-152: ADSP-SC58x SWU14 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3109F000	SWU14_GCTL	SWU14 Global Control Register	0x00000000
0x3109F004	SWU14_GSTAT	SWU14 Global Status Register	0x00000000
0x3109F010	SWU14_CTL[n]	SWU14 Control Register n	0x00000000
0x3109F014	SWU14_LA[n]	SWU14 Lower Address Register n	0x00000000
0x3109F018	SWU14_UA[n]	SWU14 Upper Address Register n	0x00000000
0x3109F01C	SWU14_ID[n]	SWU14 ID Register n	0x00000000
0x3109F020	SWU14_CNT[n]	SWU14 Count Register n	0x00000000
0x3109F024	SWU14_TARG[n]	SWU14 Target Register n	0x00000000
0x3109F028	SWU14_HIST[n]	SWU14 Bandwidth History Register n	0x00000000
0x3109F02C	SWU14_CUR[n]	SWU14 Current Register n	0x00000000
0x3109F030	SWU14_CTL[n]	SWU14 Control Register n	0x00000000
0x3109F034	SWU14_LA[n]	SWU14 Lower Address Register n	0x00000000
0x3109F038	SWU14_UA[n]	SWU14 Upper Address Register n	0x00000000
0x3109F03C	SWU14_ID[n]	SWU14 ID Register n	0x00000000
0x3109F040	SWU14_CNT[n]	SWU14 Count Register n	0x00000000
0x3109F044	SWU14_TARG[n]	SWU14 Target Register n	0x00000000
0x3109F048	SWU14_HIST[n]	SWU14 Bandwidth History Register n	0x00000000
0x3109F04C	SWU14_CUR[n]	SWU14 Current Register n	0x00000000
0x3109F050	SWU14_CTL[n]	SWU14 Control Register n	0x00000000
0x3109F054	SWU14_LA[n]	SWU14 Lower Address Register n	0x00000000
0x3109F058	SWU14_UA[n]	SWU14 Upper Address Register n	0x00000000
0x3109F05C	SWU14_ID[n]	SWU14 ID Register n	0x00000000
0x3109F060	SWU14_CNT[n]	SWU14 Count Register n	0x00000000
0x3109F064	SWU14_TARG[n]	SWU14 Target Register n	0x00000000
0x3109F068	SWU14_HIST[n]	SWU14 Bandwidth History Register n	0x00000000
0x3109F06C	SWU14_CUR[n]	SWU14 Current Register n	0x00000000
0x3109F070	SWU14_CTL[n]	SWU14 Control Register n	0x00000000
0x3109F074	SWU14_LA[n]	SWU14 Lower Address Register n	0x00000000
0x3109F078	SWU14_UA[n]	SWU14 Upper Address Register n	0x00000000
0x3109F07C	SWU14_ID[n]	SWU14 ID Register n	0x00000000
0x3109F080	SWU14_CNT[n]	SWU14 Count Register n	0x00000000

Table A-152: ADSP-SC58x SWU14 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109F084	SWU14_TARG[n]	SWU14 Target Register n	0x00000000
0x3109F088	SWU14_HIST[n]	SWU14 Bandwidth History Register n	0x00000000
0x3109F08C	SWU14_CUR[n]	SWU14 Current Register n	0x00000000

Table A-153: ADSP-SC58x SWU15 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31098000	SWU15_GCTL	SWU15 Global Control Register	0x00000000
0x31098004	SWU15_GSTAT	SWU15 Global Status Register	0x00000000
0x31098010	SWU15_CTL[n]	SWU15 Control Register n	0x00000000
0x31098014	SWU15_LA[n]	SWU15 Lower Address Register n	0x00000000
0x31098018	SWU15_UA[n]	SWU15 Upper Address Register n	0x00000000
0x3109801C	SWU15_ID[n]	SWU15 ID Register n	0x00000000
0x31098020	SWU15_CNT[n]	SWU15 Count Register n	0x00000000
0x31098024	SWU15_TARG[n]	SWU15 Target Register n	0x00000000
0x31098028	SWU15_HIST[n]	SWU15 Bandwidth History Register n	0x00000000
0x3109802C	SWU15_CUR[n]	SWU15 Current Register n	0x00000000
0x31098030	SWU15_CTL[n]	SWU15 Control Register n	0x00000000
0x31098034	SWU15_LA[n]	SWU15 Lower Address Register n	0x00000000
0x31098038	SWU15_UA[n]	SWU15 Upper Address Register n	0x00000000
0x3109803C	SWU15_ID[n]	SWU15 ID Register n	0x00000000
0x31098040	SWU15_CNT[n]	SWU15 Count Register n	0x00000000
0x31098044	SWU15_TARG[n]	SWU15 Target Register n	0x00000000
0x31098048	SWU15_HIST[n]	SWU15 Bandwidth History Register n	0x00000000
0x3109804C	SWU15_CUR[n]	SWU15 Current Register n	0x00000000
0x31098050	SWU15_CTL[n]	SWU15 Control Register n	0x00000000
0x31098054	SWU15_LA[n]	SWU15 Lower Address Register n	0x00000000
0x31098058	SWU15_UA[n]	SWU15 Upper Address Register n	0x00000000
0x3109805C	SWU15_ID[n]	SWU15 ID Register n	0x00000000
0x31098060	SWU15_CNT[n]	SWU15 Count Register n	0x00000000
0x31098064	SWU15_TARG[n]	SWU15 Target Register n	0x00000000

Table A-153: ADSP-SC58x SWU15 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31098068	SWU15_HIST[n]	SWU15 Bandwidth History Register n	0x00000000
0x3109806C	SWU15_CUR[n]	SWU15 Current Register n	0x00000000
0x31098070	SWU15_CTL[n]	SWU15 Control Register n	0x00000000
0x31098074	SWU15_LA[n]	SWU15 Lower Address Register n	0x00000000
0x31098078	SWU15_UA[n]	SWU15 Upper Address Register n	0x00000000
0x3109807C	SWU15_ID[n]	SWU15 ID Register n	0x00000000
0x31098080	SWU15_CNT[n]	SWU15 Count Register n	0x00000000
0x31098084	SWU15_TARG[n]	SWU15 Target Register n	0x00000000
0x31098088	SWU15_HIST[n]	SWU15 Bandwidth History Register n	0x00000000
0x3109808C	SWU15_CUR[n]	SWU15 Current Register n	0x00000000

Table A-154: ADSP-SC58x SWU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31092000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x31092004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x31092010	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31092014	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31092018	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109201C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31092020	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31092024	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31092028	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109202C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31092030	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31092034	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31092038	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109203C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31092040	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31092044	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31092048	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000

Table A-154: ADSP-SC58x SWU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3109204C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31092050	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31092054	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31092058	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109205C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31092060	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31092064	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31092068	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109206C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x31092070	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x31092074	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x31092078	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x3109207C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x31092080	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x31092084	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x31092088	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x3109208C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000

Table A-155: ADSP-SC58x SWU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31093000	SWU3_GCTL	SWU3 Global Control Register	0x00000000
0x31093004	SWU3_GSTAT	SWU3 Global Status Register	0x00000000
0x31093010	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x31093014	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x31093018	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x3109301C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x31093020	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x31093024	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x31093028	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x3109302C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table A-155: ADSP-SC58x SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31093030	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x31093034	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x31093038	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x3109303C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x31093040	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x31093044	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x31093048	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x3109304C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x31093050	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x31093054	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x31093058	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x3109305C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x31093060	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x31093064	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x31093068	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x3109306C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x31093070	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x31093074	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x31093078	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x3109307C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x31093080	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x31093084	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x31093088	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x3109308C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table A-156: ADSP-SC58x SWU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31094000	SWU4_GCTL	SWU4 Global Control Register	0x00000000
0x31094004	SWU4_GSTAT	SWU4 Global Status Register	0x00000000
0x31094010	SWU4_CTL[n]	SWU4 Control Register n	0x00000000

Table A-156: ADSP-SC58x SWU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31094014	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x31094018	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x3109401C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x31094020	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x31094024	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x31094028	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x3109402C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x31094030	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x31094034	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x31094038	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x3109403C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x31094040	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x31094044	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x31094048	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x3109404C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x31094050	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x31094054	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x31094058	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x3109405C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x31094060	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x31094064	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x31094068	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x3109406C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x31094070	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x31094074	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x31094078	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x3109407C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x31094080	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x31094084	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x31094088	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x3109408C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000

Table A-157: ADSP-SC58x SWU5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31095000	SWU5_GCTL	SWU5 Global Control Register	0x00000000
0x31095004	SWU5_GSTAT	SWU5 Global Status Register	0x00000000
0x31095010	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x31095014	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x31095018	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x3109501C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x31095020	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x31095024	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x31095028	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x3109502C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x31095030	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x31095034	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x31095038	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x3109503C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x31095040	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x31095044	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x31095048	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x3109504C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x31095050	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x31095054	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x31095058	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x3109505C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x31095060	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x31095064	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x31095068	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x3109506C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x31095070	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x31095074	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x31095078	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x3109507C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x31095080	SWU5_CNT[n]	SWU5 Count Register n	0x00000000

Table A-157: ADSP-SC58x SWU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31095084	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x31095088	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x3109508C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000

Table A-158: ADSP-SC58x SWU6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31096000	SWU6_GCTL	SWU6 Global Control Register	0x00000000
0x31096004	SWU6_GSTAT	SWU6 Global Status Register	0x00000000
0x31096010	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x31096014	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x31096018	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x3109601C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x31096020	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x31096024	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x31096028	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x3109602C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x31096030	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x31096034	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x31096038	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x3109603C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x31096040	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x31096044	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x31096048	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x3109604C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x31096050	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x31096054	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x31096058	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x3109605C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x31096060	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x31096064	SWU6_TARG[n]	SWU6 Target Register n	0x00000000

Table A-158: ADSP-SC58x SWU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31096068	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x3109606C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x31096070	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x31096074	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x31096078	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x3109607C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x31096080	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x31096084	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x31096088	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x3109608C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000

Table A-159: ADSP-SC58x SWU7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31140000	SWU7_GCTL	SWU7 Global Control Register	0x00000000
0x31140004	SWU7_GSTAT	SWU7 Global Status Register	0x00000000
0x31140010	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140014	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140018	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114001C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140020	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140024	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140028	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114002C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140030	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140034	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140038	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114003C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140040	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140044	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140048	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000

Table A-159: ADSP-SC58x SWU7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3114004C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140050	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140054	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140058	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114005C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140060	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140064	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140068	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114006C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x31140070	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x31140074	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x31140078	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x3114007C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x31140080	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x31140084	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x31140088	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x3114008C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000

Table A-160: ADSP-SC58x SWU8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31141000	SWU8_GCTL	SWU8 Global Control Register	0x00000000
0x31141004	SWU8_GSTAT	SWU8 Global Status Register	0x00000000
0x31141010	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141014	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141018	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114101C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141020	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141024	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141028	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114102C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000

Table A-160: ADSP-SC58x SWU8 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31141030	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141034	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141038	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114103C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141040	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141044	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141048	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114104C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000
0x31141050	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141054	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141058	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114105C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141060	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141064	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141068	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114106C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000
0x31141070	SWU8_CTL[n]	SWU8 Control Register n	0x00000000
0x31141074	SWU8_LA[n]	SWU8 Lower Address Register n	0x00000000
0x31141078	SWU8_UA[n]	SWU8 Upper Address Register n	0x00000000
0x3114107C	SWU8_ID[n]	SWU8 ID Register n	0x00000000
0x31141080	SWU8_CNT[n]	SWU8 Count Register n	0x00000000
0x31141084	SWU8_TARG[n]	SWU8 Target Register n	0x00000000
0x31141088	SWU8_HIST[n]	SWU8 Bandwidth History Register n	0x00000000
0x3114108C	SWU8_CUR[n]	SWU8 Current Register n	0x00000000

Table A-161: ADSP-SC58x SWU9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31142000	SWU9_GCTL	SWU9 Global Control Register	0x00000000
0x31142004	SWU9_GSTAT	SWU9 Global Status Register	0x00000000
0x31142010	SWU9_CTL[n]	SWU9 Control Register n	0x00000000

Table A-161: ADSP-SC58x SWU9 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31142014	SWU9_LA[n]	SWU9 Lower Address Register n	0x00000000
0x31142018	SWU9_UA[n]	SWU9 Upper Address Register n	0x00000000
0x3114201C	SWU9_ID[n]	SWU9 ID Register n	0x00000000
0x31142020	SWU9_CNT[n]	SWU9 Count Register n	0x00000000
0x31142024	SWU9_TARG[n]	SWU9 Target Register n	0x00000000
0x31142028	SWU9_HIST[n]	SWU9 Bandwidth History Register n	0x00000000
0x3114202C	SWU9_CUR[n]	SWU9 Current Register n	0x00000000
0x31142030	SWU9_CTL[n]	SWU9 Control Register n	0x00000000
0x31142034	SWU9_LA[n]	SWU9 Lower Address Register n	0x00000000
0x31142038	SWU9_UA[n]	SWU9 Upper Address Register n	0x00000000
0x3114203C	SWU9_ID[n]	SWU9 ID Register n	0x00000000
0x31142040	SWU9_CNT[n]	SWU9 Count Register n	0x00000000
0x31142044	SWU9_TARG[n]	SWU9 Target Register n	0x00000000
0x31142048	SWU9_HIST[n]	SWU9 Bandwidth History Register n	0x00000000
0x3114204C	SWU9_CUR[n]	SWU9 Current Register n	0x00000000
0x31142050	SWU9_CTL[n]	SWU9 Control Register n	0x00000000
0x31142054	SWU9_LA[n]	SWU9 Lower Address Register n	0x00000000
0x31142058	SWU9_UA[n]	SWU9 Upper Address Register n	0x00000000
0x3114205C	SWU9_ID[n]	SWU9 ID Register n	0x00000000
0x31142060	SWU9_CNT[n]	SWU9 Count Register n	0x00000000
0x31142064	SWU9_TARG[n]	SWU9 Target Register n	0x00000000
0x31142068	SWU9_HIST[n]	SWU9 Bandwidth History Register n	0x00000000
0x3114206C	SWU9_CUR[n]	SWU9 Current Register n	0x00000000
0x31142070	SWU9_CTL[n]	SWU9 Control Register n	0x00000000
0x31142074	SWU9_LA[n]	SWU9 Lower Address Register n	0x00000000
0x31142078	SWU9_UA[n]	SWU9 Upper Address Register n	0x00000000
0x3114207C	SWU9_ID[n]	SWU9 ID Register n	0x00000000
0x31142080	SWU9_CNT[n]	SWU9 Count Register n	0x00000000
0x31142084	SWU9_TARG[n]	SWU9 Target Register n	0x00000000
0x31142088	SWU9_HIST[n]	SWU9 Bandwidth History Register n	0x00000000
0x3114208C	SWU9_CUR[n]	SWU9 Current Register n	0x00000000

Table A-162: ADSP-SC58x TAPC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31130000	TAPC_IDCODE	TAPC IDCODE Register	0x028080CB
0x31130004	TAPC_USERCODE	TAPC USERCODE Register	0x00000000
0x31130008	TAPC_SDBGKEY_CTL	TAPC Secure Debug Key Control Register	0x00000000
0x3113000C	TAPC_SDBGKEY_STAT	TAPC Secure Debug Key Status Register	0x00000000
0x31130010	TAPC_SDBGKEY0	TAPC Secure Debug Key 0 Register	0x00000000
0x31130014	TAPC_SDBGKEY1	TAPC Secure Debug Key 1 Register	0x00000000
0x31130018	TAPC_SDBGKEY2	TAPC Secure Debug Key 2 Register	0x00000000
0x3113001C	TAPC_SDBGKEY3	TAPC Secure Debug Key 3 Register	0x00000000
0x31131000	TAPC_DBGCTL	TAPC Debug Control Register	0x00000000

Table A-163: ADSP-SC58x TIMER0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001004	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x31001008	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x3100100C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x31001010	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x31001014	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000
0x31001018	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x3100101C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000000FF
0x31001020	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000000FF
0x31001024	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000000FF
0x31001028	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x3100102C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x31001030	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x31001034	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x31001038	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x3100103C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000
0x31001040	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x31001060	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31001064	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001

Table A-163: ADSP-SC58x TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31001068	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3100106C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31001070	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31001080	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31001084	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31001088	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3100108C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31001090	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310010A0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310010A4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310010A8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310010AC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310010B0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310010C0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310010C4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310010C8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310010CC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310010D0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x310010E0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x310010E4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x310010E8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x310010EC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x310010F0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31001100	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31001104	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31001108	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3100110C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31001110	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31001120	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31001124	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31001128	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000

Table A-163: ADSP-SC58x TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3100112C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31001130	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x31001140	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x31001144	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x31001148	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x3100114C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x31001150	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-164: ADSP-SC58x TMU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31016000	TMU0_CTL	TMU0 TMU Control Register	0x00000000
0x31016004	TMU0_TEMP	TMU0 Temperature Value Register	0x00000000
0x31016008	TMU0_AVG	TMU0 Averaging Register	0x00000000
0x3101600C	TMU0_FLT_LIM_HI	TMU0 Fault High Limit Register	0x0000007F
0x31016010	TMU0_ALRT_LIM_HI	TMU0 Alert High Limit Register	0x0000007F
0x31016014	TMU0_FLT_LIM_LO	TMU0 Fault Low Limit Register	0x00000080
0x31016018	TMU0_ALRT_LIM_LO	TMU0 Alert Low Limit Register	0x00000080
0x3101601C	TMU0_STAT	TMU0 Status Register	0x00000000
0x31016024	TMU0_GAIN	TMU0 Gain Value Register	0x00000000
0x31016028	TMU0_IMSK	TMU0 Interrupt Mask Register	0x0000000F
0x3101602C	TMU0_OFFSET	TMU0 Offset Register	0x00000000

Table A-165: ADSP-SC58x TRNG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310D0000	TRNG0_INPUT[n]	TRNG0 TRNG Input Registers	0x00000000
0x310D0000	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0004	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0008	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D000C	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x310D0010	TRNG0_STAT	TRNG0 TRNG Status Register	0x00000000

Table A-165: ADSP-SC58x TRNG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310D0010	TRNG0_INTACK	TRNG0 TRNG Interrupt Acknowledge Register	0x00000000
0x310D0014	TRNG0_CTL	TRNG0 TRNG Control Register	0x00000000
0x310D0018	TRNG0_CFG	TRNG0 TRNG Configuration Register	0x00000000
0x310D001C	TRNG0_ALMCNT	TRNG0 TRNG Alarm Counter Register	0x000000FF
0x310D0020	TRNG0_FROEN	TRNG0 TRNG FRO Enable Register	0x000000FF
0x310D0024	TRNG0_FRODETUNE	TRNG0 TRNG FRO De-tune Register	0x00000000
0x310D0028	TRNG0_ALMMSK	TRNG0 TRNG Alarm Mask Register	0x00000000
0x310D002C	TRNG0_ALMSTP	TRNG0 TRNG Alarm Stop Register	0x00000000
0x310D0030	TRNG0_LFSR_L	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D0034	TRNG0_LFSR_M	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D0038	TRNG0_LFSR_H	TRNG0 TRNG LFSR Access Register	0x00000000
0x310D003C	TRNG0_CNT	TRNG0 Counter Register	0x00000000
0x310D0040	TRNG0_RUNCNT	TRNG0 TRNG Run Count Registers	0x00000000
0x310D0044	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0048	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D004C	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0050	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0054	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D0058	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x310D005C	TRNG0_MONOBITCNT	TRNG0 TRNG Monobit Test Result Register	0x00002710
0x310D0060	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0064	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0068	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D006C	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x310D0070	TRNG0_TEST	TRNG0 TRNG Test Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A000	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A004	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A008	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A00C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A010	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A014	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A018	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A01C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A020	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A024	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A028	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A02C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A030	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A034	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A038	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A03C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A040	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A044	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A048	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A04C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A050	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A054	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A058	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A05C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A060	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A064	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A068	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A06C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A070	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A074	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A078	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A07C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A080	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A084	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A088	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A08C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A090	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A094	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A098	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A09C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0BC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0C0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0C4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0C8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0CC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0D8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0DC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0E8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0EC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0F8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A0FC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A100	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A104	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A108	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A10C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A110	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A114	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A118	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A11C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A120	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A124	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A128	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A12C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A130	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A134	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A138	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A13C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A140	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A144	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A148	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A14C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A150	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A154	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A158	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A15C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A160	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A164	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A168	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A16C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A170	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A174	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A178	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A17C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A180	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A184	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A188	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A18C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A190	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A194	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A198	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A19C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1BC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1C8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1CC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1D0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1D4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1D8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1DC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1E0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1E4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1E8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1EC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1F0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1F4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-166: ADSP-SC58x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x3108A1F8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A1FC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A200	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A204	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A208	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A20C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A210	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A214	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A218	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A21C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A220	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x3108A7E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x3108A7E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x3108A7EC	TRU0_STAT	TRU0 Status Information Register	0x00000000
0x3108A7F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table A-167: ADSP-SC58x TWI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001400	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x31001404	TWI0_CTL	TWI0 Control Register	0x00000000
0x31001408	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x3100140C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x31001410	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000
0x31001414	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x31001418	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000
0x3100141C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x31001420	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x31001424	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x31001428	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x3100142C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000

Table A-167: ADSP-SC58x TWI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31001480	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x31001484	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x31001488	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000
0x3100148C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table A-168: ADSP-SC58x TWI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001500	TWI1_CLKDIV	TWI1 SCL Clock Divider Register	0x00000000
0x31001504	TWI1_CTL	TWI1 Control Register	0x00000000
0x31001508	TWI1_SLVCTL	TWI1 Slave Mode Control Register	0x00000000
0x3100150C	TWI1_SLVSTAT	TWI1 Slave Mode Status Register	0x00000000
0x31001510	TWI1_SLVADDR	TWI1 Slave Mode Address Register	0x00000000
0x31001514	TWI1_MSTRCTL	TWI1 Master Mode Control Registers	0x00000000
0x31001518	TWI1_MSTRSTAT	TWI1 Master Mode Status Register	0x00000000
0x3100151C	TWI1_MSTRADDR	TWI1 Master Mode Address Register	0x00000000
0x31001520	TWI1_ISTAT	TWI1 Interrupt Status Register	0x00000000
0x31001524	TWI1_IMSK	TWI1 Interrupt Mask Register	0x00000000
0x31001528	TWI1_FIFOCTL	TWI1 FIFO Control Register	0x00000000
0x3100152C	TWI1_FIFOSTAT	TWI1 FIFO Status Register	0x00000000
0x31001580	TWI1_TXDATA8	TWI1 Tx Data Single-Byte Register	0x00000000
0x31001584	TWI1_TXDATA16	TWI1 Tx Data Double-Byte Register	0x00000000
0x31001588	TWI1_RXDATA8	TWI1 Rx Data Single-Byte Register	0x00000000
0x3100158C	TWI1_RXDATA16	TWI1 Rx Data Double-Byte Register	0x00000000

Table A-169: ADSP-SC58x TWI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31001600	TWI2_CLKDIV	TWI2 SCL Clock Divider Register	0x00000000
0x31001604	TWI2_CTL	TWI2 Control Register	0x00000000
0x31001608	TWI2_SLVCTL	TWI2 Slave Mode Control Register	0x00000000
0x3100160C	TWI2_SLVSTAT	TWI2 Slave Mode Status Register	0x00000000

Table A-169: ADSP-SC58x TWI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31001610	TWI2_SLVADDR	TWI2 Slave Mode Address Register	0x00000000
0x31001614	TWI2_MSTRCTL	TWI2 Master Mode Control Registers	0x00000000
0x31001618	TWI2_MSTRSTAT	TWI2 Master Mode Status Register	0x00000000
0x3100161C	TWI2_MSTRADDR	TWI2 Master Mode Address Register	0x00000000
0x31001620	TWI2_ISTAT	TWI2 Interrupt Status Register	0x00000000
0x31001624	TWI2_IMSK	TWI2 Interrupt Mask Register	0x00000000
0x31001628	TWI2_FIFOCTL	TWI2 FIFO Control Register	0x00000000
0x3100162C	TWI2_FIFOSTAT	TWI2 FIFO Status Register	0x00000000
0x31001680	TWI2_TXDATA8	TWI2 Tx Data Single-Byte Register	0x00000000
0x31001684	TWI2_TXDATA16	TWI2 Tx Data Double-Byte Register	0x00000000
0x31001688	TWI2_RXDATA8	TWI2 Rx Data Single-Byte Register	0x00000000
0x3100168C	TWI2_RXDATA16	TWI2 Rx Data Double-Byte Register	0x00000000

Table A-170: ADSP-SC58x UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31003004	UART0_CTL	UART0 Control Register	0x00000000
0x31003008	UART0_STAT	UART0 Status Register	0x000000A0
0x3100300C	UART0_SCR	UART0 Scratch Register	0x00000000
0x31003010	UART0_CLK	UART0 Clock Rate Register	0x0000FFFF
0x31003014	UART0_IMSK	UART0 Interrupt Mask Register	0x00000000
0x31003018	UART0_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x3100301C	UART0_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x31003020	UART0_RBR	UART0 Receive Buffer Register	0x00000000
0x31003024	UART0_THR	UART0 Transmit Hold Register	0x00000000
0x31003028	UART0_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x3100302C	UART0_TSR	UART0 Transmit Shift Register	0x000007FF
0x31003030	UART0_RSR	UART0 Receive Shift Register	0x00000000
0x31003034	UART0_TXCNT	UART0 Transmit Counter Register	0x00000000
0x31003038	UART0_RXCNT	UART0 Receive Counter Register	0x00000000

Table A-171: ADSP-SC58x UART1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31003404	UART1_CTL	UART1 Control Register	0x00000000
0x31003408	UART1_STAT	UART1 Status Register	0x000000A0
0x3100340C	UART1_SCR	UART1 Scratch Register	0x00000000
0x31003410	UART1_CLK	UART1 Clock Rate Register	0x0000FFFF
0x31003414	UART1_IMSK	UART1 Interrupt Mask Register	0x00000000
0x31003418	UART1_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x3100341C	UART1_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x31003420	UART1_RBR	UART1 Receive Buffer Register	0x00000000
0x31003424	UART1_THR	UART1 Transmit Hold Register	0x00000000
0x31003428	UART1_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x3100342C	UART1_TSR	UART1 Transmit Shift Register	0x000007FF
0x31003430	UART1_RSR	UART1 Receive Shift Register	0x00000000
0x31003434	UART1_TXCNT	UART1 Transmit Counter Register	0x00000000
0x31003438	UART1_RXCNT	UART1 Receive Counter Register	0x00000000

Table A-172: ADSP-SC58x UART2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31003804	UART2_CTL	UART2 Control Register	0x00000000
0x31003808	UART2_STAT	UART2 Status Register	0x000000A0
0x3100380C	UART2_SCR	UART2 Scratch Register	0x00000000
0x31003810	UART2_CLK	UART2 Clock Rate Register	0x0000FFFF
0x31003814	UART2_IMSK	UART2 Interrupt Mask Register	0x00000000
0x31003818	UART2_IMSK_SET	UART2 Interrupt Mask Set Register	0x00000000
0x3100381C	UART2_IMSK_CLR	UART2 Interrupt Mask Clear Register	0x00000000
0x31003820	UART2_RBR	UART2 Receive Buffer Register	0x00000000
0x31003824	UART2_THR	UART2 Transmit Hold Register	0x00000000
0x31003828	UART2_TAIP	UART2 Transmit Address/Insert Pulse Register	0x00000000
0x3100382C	UART2_TSR	UART2 Transmit Shift Register	0x000007FF
0x31003830	UART2_RSR	UART2 Receive Shift Register	0x00000000
0x31003834	UART2_TXCNT	UART2 Transmit Counter Register	0x00000000

Table A-172: ADSP-SC58x UART2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x31003838	UART2_RXCNT	UART2 Receive Counter Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1000	USB0_FADDR	USB0 Function Address Register	0x00000000
0x310C1001	USB0_POWER	USB0 Power and Device Control Register	0x00000020
0x310C1002	USB0_INTRTX	USB0 Transmit Interrupt Register	0x00000000
0x310C1004	USB0_INTRRX	USB0 Receive Interrupt Register	0x00000000
0x310C1006	USB0_INTRTXE	USB0 Transmit Interrupt Enable Register	0x00000FFF
0x310C1008	USB0_INTRRXE	USB0 Receive Interrupt Enable Register	0x00000FFE
0x310C100A	USB0_IRQ	USB0 Common Interrupts Register	0x00000000
0x310C100B	USB0_IEN	USB0 Common Interrupts Enable Register	0x00000000
0x310C100C	USB0_FRAME	USB0 Frame Number Register	0x00000000
0x310C100E	USB0_INDEX	USB0 Index Register	0x00000000
0x310C100F	USB0_TESTMODE	USB0 Testmode Register	0x00000000
0x310C1010	USB0_EPI[N]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1012	USB0_EP0I_CSR[N]_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x310C1012	USB0_EP0I_CSR[N]_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000
0x310C1012	USB0_EPI[N]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1012	USB0_EPI[N]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1014	USB0_EPI[N]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1016	USB0_EPI[N]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1016	USB0_EPI[N]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1018	USB0_EP0I_CNT[N]	USB0 EP0 Number of Received Bytes Register	0x00000000
0x310C1018	USB0_EPI[N]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C101A	USB0_EP0I_TYPE[N]	USB0 EP0 Connection Type Register	0x00000000
0x310C101A	USB0_EPI[N]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C101B	USB0_EP0I_NAKLIMIT[N]	USB0 EP0 NAK Limit Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C101B	USB0_EPI[N]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C101C	USB0_EPI[N]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C101D	USB0_EPI[N]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C101F	USB0_EP0I_CFGDATA[N]	USB0 EP0 Configuration Information Register	0x0000001E
0x310C1020	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1020	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1020	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1024	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1024	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1024	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1028	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1028	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1028	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C102C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C102C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C102C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1030	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1030	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1030	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1034	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1034	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1034	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1038	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1038	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1038	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C103C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C103C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C103C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1040	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1040	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1040	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1044	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1044	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1044	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1048	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C1048	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C1048	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C104C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x310C104C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x310C104C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C1060	USB0_DEV_CTL	USB0 Device Control Register	0x00000000
0x310C1062	USB0_TXFIFOSZ	USB0 Transmit FIFO Size Register	0x00000000
0x310C1063	USB0_RXFIFOSZ	USB0 Receive FIFO Size Register	0x00000000
0x310C1064	USB0_TXFIFOADDR	USB0 Transmit FIFO Address Register	0x00000000
0x310C1066	USB0_RXFIFOADDR	USB0 Receive FIFO Address Register	0x00000000
0x310C1078	USB0_EPINFO	USB0 Endpoint Information Register	0x000000CC
0x310C1079	USB0_RAMINFO	USB0 RAM Information Register	0x0000008C
0x310C107A	USB0_LINKINFO	USB0 Link Information Register	0x0000005C
0x310C107B	USB0_VPLEN	USB0 VBUS Pulse Length Register	0x0000003C
0x310C107C	USB0_HS_EOF1	USB0 High-Speed EOF 1 Register	0x00000080
0x310C107D	USB0_FS_EOF1	USB0 Full-Speed EOF 1 Register	0x00000077
0x310C107E	USB0_LS_EOF1	USB0 Low-Speed EOF 1 Register	0x00000072
0x310C107F	USB0_SOFT_RST	USB0 Software Reset Register	0x00000000
0x310C1080	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C1082	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C1083	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C1084	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1086	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C1087	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C1088	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C108A	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C108B	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C108C	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C108E	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C108F	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C1090	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C1092	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C1093	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C1094	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C1096	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C1097	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C1098	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C109A	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C109B	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C109C	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C109E	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C109F	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10A0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10A2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10A3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10A4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10A6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10A7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10A8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10AA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10AB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10AC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10AE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10AF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10B0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10B2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10B3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10B4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C10B6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10B7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10B8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10BA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10BB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10BC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10BE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10BF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10C0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10C2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10C3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10C4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10C6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10C7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10C8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10CA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10CB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10CC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C10CE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10CF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10D0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10D2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10D3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10D4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10D6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10D7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C10D8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x310C10DA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x310C10DB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x310C10DC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x310C10DE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x310C10DF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x310C1100	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1102	USB0_EP0_CSR[n]_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x310C1102	USB0_EP0_CSR[n]_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000
0x310C1102	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1102	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1104	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1106	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1106	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1108	USB0_EP0_CNT[n]	USB0 EP0 Number of Received Bytes Register	0x00000000
0x310C1108	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C110A	USB0_EP0_TYPE[n]	USB0 EP0 Connection Type Register	0x00000000
0x310C110A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C110B	USB0_EP0_NAKLIMIT[n]	USB0 EP0 NAK Limit Register	0x00000000
0x310C110B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C110C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C110D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C110F	USB0_EP0_CFGDATA[n]	USB0 EP0 Configuration Information Register	0x0000001E
0x310C1110	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1112	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1112	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1114	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1116	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1116	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1118	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C111A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C111B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C111C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C111D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1120	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1122	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1122	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1124	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1126	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1126	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1128	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C112A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C112B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C112C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C112D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1130	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1132	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1132	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1134	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1136	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1136	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1138	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C113A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C113B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C113C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C113D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1140	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1142	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1142	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1144	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1146	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1146	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1148	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C114A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C114B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C114C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C114D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1150	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1152	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1152	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1154	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1156	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1156	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1158	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C115A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C115B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C115C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C115D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1160	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1162	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1162	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1164	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1166	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1166	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1168	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C116A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C116B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C116C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C116D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1170	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1172	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1172	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1174	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1176	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1176	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1178	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C117A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C117B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C117C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C117D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1180	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1182	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1182	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1184	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1186	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1186	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1188	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C118A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C118B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C118C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C118D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1190	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C1192	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C1192	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C1194	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C1196	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C1196	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C1198	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C119A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C119B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C119C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C119D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C11A0	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C11A2	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C11A2	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C11A4	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C11A6	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C11A6	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C11A8	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C11AA	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C11AB	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C11AC	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C11AD	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C11B0	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C11B2	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C11B2	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C11B4	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x310C11B6	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C11B6	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C11B8	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x310C11BA	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x310C11BB	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x310C11BC	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x310C11BD	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x310C1200	USB0_DMA_IRQ	USB0 DMA Interrupt Register	0x00000000
0x310C1204	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1208	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C120C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1214	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1218	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C121C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1224	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1228	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C122C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1234	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1238	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C123C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1244	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1248	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C124C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1254	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1258	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C125C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1264	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000

Table A-173: ADSP-SC58x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C1268	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C126C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1274	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x310C1278	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x310C127C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x310C1300	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1304	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1308	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C130C	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1310	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1314	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1318	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C131C	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1320	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1324	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1328	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x310C1344	USB0_CT_UCH	USB0 Chirp Timeout Register	0x00004074
0x310C1346	USB0_CT_HHSRTN	USB0 Host High-Speed Return to Normal Register	0x000005E6
0x310C1348	USB0_CT_HSBT	USB0 High-Speed Timeout Register	0x00000000
0x310C1360	USB0_LPM_ATTR	USB0 LPM Attribute Register	0x00000000
0x310C1362	USB0_LPM_CTL	USB0 LPM Control Register	0x00000000
0x310C1363	USB0_LPM_IEN	USB0 LPM Interrupt Enable Register	0x00000000
0x310C1364	USB0_LPM_IRQ	USB0 LPM Interrupt Status Register	0x00000000
0x310C1365	USB0_LPM_FADDR	USB0 LPM Function Address Register	0x00000000
0x310C1380	USB0_VBUS_CTL	USB0 VBUS Control Register	0x00000000
0x310C1381	USB0_BAT_CHG	USB0 Battery Charging Control Register	0x00000000
0x310C1382	USB0_IDCTL	USB0 ID Control	0x00000000
0x310C1394	USB0_PHY_CTL	USB0 PHY Control Register	0x00000000
0x310C1398	USB0_PLL_OSC	USB0 PLL and Oscillator Control Register	0x00000014

Table A-174: ADSP-SC58x USB1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x310C2000	USB1_FADDR	USB1 Function Address Register	0x00000000
0x310C2001	USB1_POWER	USB1 Power and Device Control Register	0x00000020
0x310C2002	USB1_INTRTX	USB1 Transmit Interrupt Register	0x00000000
0x310C2004	USB1_INTRRX	USB1 Receive Interrupt Register	0x00000000
0x310C2006	USB1_INTRTXE	USB1 Transmit Interrupt Enable Register	0x00000FFF
0x310C2008	USB1_INTRRXE	USB1 Receive Interrupt Enable Register	0x00000FFE
0x310C200A	USB1_IRQ	USB1 Common Interrupts Register	0x00000000
0x310C200B	USB1_IEN	USB1 Common Interrupts Enable Register	0x00000000
0x310C200C	USB1_FRAME	USB1 Frame Number Register	0x00000000
0x310C200E	USB1_INDEX	USB1 Index Register	0x00000000
0x310C200F	USB1_TESTMODE	USB1 Testmode Register	0x00000000
0x310C2010	USB1_EPI[N]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2012	USB1_EP0I_CSR[N]_P	USB1 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x310C2012	USB1_EP0I_CSR[N]_H	USB1 EP0 Configuration and Status (Host) Register	0x00000000
0x310C2012	USB1_EPI[N]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2012	USB1_EPI[N]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2014	USB1_EPI[N]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2016	USB1_EPI[N]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2016	USB1_EPI[N]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2018	USB1_EP0I_CNT[N]	USB1 EP0 Number of Received Bytes Register	0x00000000
0x310C2018	USB1_EPI[N]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C201A	USB1_EP0I_TYPE[N]	USB1 EP0 Connection Type Register	0x00000000
0x310C201A	USB1_EPI[N]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C201B	USB1_EP0I_NAKLIMIT[N]	USB1 EP0 NAK Limit Register	0x00000000
0x310C201B	USB1_EPI[N]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C201C	USB1_EPI[N]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C201D	USB1_EPI[N]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C201F	USB1_EP0I_CFGDATA[N]	USB1 EP0 Configuration Information Register	0x0000001E
0x310C2020	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2020	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2020	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2024	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2024	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2024	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2028	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2028	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2028	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C202C	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C202C	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C202C	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2030	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2030	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2030	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2034	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2034	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2034	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2038	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2038	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2038	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C203C	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C203C	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C203C	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2040	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2040	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2040	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2044	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2044	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2044	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C2048	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C2048	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C2048	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C204C	USB1_FIFOB[n]	USB1 FIFO Byte (8-Bit) Register	0x00000000
0x310C204C	USB1_FIFO[n]	USB1 FIFO Word (32-Bit) Register	0x00000000
0x310C204C	USB1_FIFOH[n]	USB1 FIFO Half-Word (16-Bit) Register	0x00000000
0x310C2060	USB1_DEV_CTL	USB1 Device Control Register	0x00000000
0x310C2062	USB1_TXFIFOSZ	USB1 Transmit FIFO Size Register	0x00000000
0x310C2063	USB1_RXFIFOSZ	USB1 Receive FIFO Size Register	0x00000000
0x310C2064	USB1_TXFIFOADDR	USB1 Transmit FIFO Address Register	0x00000000
0x310C2066	USB1_RXFIFOADDR	USB1 Receive FIFO Address Register	0x00000000
0x310C2078	USB1_EPINFO	USB1 Endpoint Information Register	0x000000CC
0x310C2079	USB1_RAMINFO	USB1 RAM Information Register	0x0000008C
0x310C207A	USB1_LINKINFO	USB1 Link Information Register	0x0000005C
0x310C207B	USB1_VPLEN	USB1 VBUS Pulse Length Register	0x0000003C
0x310C207C	USB1_HS_EOF1	USB1 High-Speed EOF 1 Register	0x00000080
0x310C207D	USB1_FS_EOF1	USB1 Full-Speed EOF 1 Register	0x00000077
0x310C207E	USB1_LS_EOF1	USB1 Low-Speed EOF 1 Register	0x00000072
0x310C207F	USB1_SOFT_RST	USB1 Software Reset Register	0x00000000
0x310C2080	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C2082	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C2083	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C2084	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C2086	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C2087	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C2088	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C208A	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C208B	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C208C	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C208E	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C208F	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C2090	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C2092	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C2093	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C2094	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C2096	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C2097	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C2098	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C209A	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C209B	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C209C	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C209E	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C209F	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20A0	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C20A2	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20A3	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20A4	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20A6	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20A7	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20A8	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C20AA	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20AB	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20AC	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20AE	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20AF	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20B0	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C20B2	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20B3	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20B4	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20B6	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20B7	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20B8	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C20BA	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20BB	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20BC	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20BE	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20BF	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20C0	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C20C2	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20C3	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20C4	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20C6	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20C7	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20C8	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C20CA	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20CB	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20CC	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20CE	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20CF	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20D0	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C20D2	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20D3	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20D4	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20D6	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20D7	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C20D8	USB1_MP[n]_TXFUNCADDR	USB1 MPn Transmit Function Address Register	0x00000000
0x310C20DA	USB1_MP[n]_TXHUBADDR	USB1 MPn Transmit Hub Address Register	0x00000000
0x310C20DB	USB1_MP[n]_TXHUBPORT	USB1 MPn Transmit Hub Port Register	0x00000000
0x310C20DC	USB1_MP[n]_RXFUNCADDR	USB1 MPn Receive Function Address Register	0x00000000
0x310C20DE	USB1_MP[n]_RXHUBADDR	USB1 MPn Receive Hub Address Register	0x00000000
0x310C20DF	USB1_MP[n]_RXHUBPORT	USB1 MPn Receive Hub Port Register	0x00000000
0x310C2100	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2102	USB1_EP0_CSR[n]_P	USB1 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x310C2102	USB1_EP0_CSR[n]_H	USB1 EP0 Configuration and Status (Host) Register	0x00000000
0x310C2102	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2102	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2104	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2106	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2106	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2108	USB1_EP0_CNT[n]	USB1 EP0 Number of Received Bytes Register	0x00000000
0x310C2108	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C210A	USB1_EP0_TYPE[n]	USB1 EP0 Connection Type Register	0x00000000
0x310C210A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C210B	USB1_EP0_NAKLIMIT[n]	USB1 EP0 NAK Limit Register	0x00000000
0x310C210B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C210C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C210D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C210F	USB1_EP0_CFGDATA[n]	USB1 EP0 Configuration Information Register	0x0000001E
0x310C2110	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2112	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2112	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2114	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2116	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2116	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2118	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C211A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C211B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C211C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C211D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2120	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2122	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2122	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2124	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2126	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2126	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2128	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C212A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C212B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C212C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C212D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2130	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2132	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2132	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2134	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2136	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2136	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2138	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C213A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C213B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C213C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C213D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2140	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2142	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2142	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2144	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2146	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2146	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2148	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C214A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C214B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C214C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C214D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2150	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2152	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2152	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2154	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2156	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2156	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2158	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C215A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C215B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C215C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C215D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2160	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2162	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2162	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2164	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2166	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2166	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2168	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C216A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C216B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C216C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C216D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2170	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2172	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C2172	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2174	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2176	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2176	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2178	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C217A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C217B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C217C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C217D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2180	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2182	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2182	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2184	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C2186	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2186	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2188	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C218A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C218B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C218C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C218D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2190	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C2192	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C2192	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C2194	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C2196	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C2196	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C2198	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C219A	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C219B	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C219C	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C219D	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C21A0	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C21A2	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C21A2	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C21A4	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C21A6	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C21A6	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000
0x310C21A8	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C21AA	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C21AB	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C21AC	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C21AD	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C21B0	USB1_EP[n]_TXMAXP	USB1 EPn Transmit Maximum Packet Length Register	0x00000000
0x310C21B2	USB1_EP[n]_TXCSR_H	USB1 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x310C21B2	USB1_EP[n]_TXCSR_P	USB1 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x310C21B4	USB1_EP[n]_RXMAXP	USB1 EPn Receive Maximum Packet Length Register	0x00000000
0x310C21B6	USB1_EP[n]_RXCSR_P	USB1 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x310C21B6	USB1_EP[n]_RXCSR_H	USB1 EPn Receive Configuration and Status (Host) Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C21B8	USB1_EP[n]_RXCNT	USB1 EPn Number of Bytes Received Register	0x00000000
0x310C21BA	USB1_EP[n]_TXTYPE	USB1 EPn Transmit Type Register	0x00000000
0x310C21BB	USB1_EP[n]_TXINTERVAL	USB1 EPn Transmit Polling Interval Register	0x00000000
0x310C21BC	USB1_EP[n]_RXTYPE	USB1 EPn Receive Type Register	0x00000000
0x310C21BD	USB1_EP[n]_RXINTERVAL	USB1 EPn Receive Polling Interval Register	0x00000000
0x310C2200	USB1_DMA_IRQ	USB1 DMA Interrupt Register	0x00000000
0x310C2204	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2208	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C220C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2214	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2218	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C221C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2224	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2228	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C222C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2234	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2238	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C223C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2244	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2248	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C224C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2254	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2258	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C225C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2264	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2268	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C226C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2274	USB1_DMA[n]_CTL	USB1 DMA Channel n Control Register	0x00000000
0x310C2278	USB1_DMA[n]_ADDR	USB1 DMA Channel n Address Register	0x00000000
0x310C227C	USB1_DMA[n]_CNT	USB1 DMA Channel n Count Register	0x00000000
0x310C2300	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000

Table A-174: ADSP-SC58x USB1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x310C2304	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2308	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C230C	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2310	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2314	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2318	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C231C	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2320	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2324	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2328	USB1_RQPKTCNT[n]	USB1 EPn Request Packet Count Register	0x00000000
0x310C2344	USB1_CT_UCH	USB1 Chirp Timeout Register	0x00004074
0x310C2346	USB1_CT_HHSRTN	USB1 Host High-Speed Return to Normal Register	0x000005E6
0x310C2348	USB1_CT_HSBT	USB1 High-Speed Timeout Register	0x00000000
0x310C2360	USB1_LPM_ATTR	USB1 LPM Attribute Register	0x00000000
0x310C2362	USB1_LPM_CTL	USB1 LPM Control Register	0x00000000
0x310C2363	USB1_LPM_IEN	USB1 LPM Interrupt Enable Register	0x00000000
0x310C2364	USB1_LPM_IRQ	USB1 LPM Interrupt Status Register	0x00000000
0x310C2365	USB1_LPM_FADDR	USB1 LPM Function Address Register	0x00000000
0x310C2380	USB1_VBUS_CTL	USB1 VBUS Control Register	0x00000000
0x310C2381	USB1_BAT_CHG	USB1 Battery Charging Control Register	0x00000000
0x310C2382	USB1_IDCTL	USB1 ID Control	0x00000000
0x310C2394	USB1_PHY_CTL	USB1 PHY Control Register	0x00000000

Table A-175: ADSP-SC58x WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31008000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0
0x31008004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x31008008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000

Table A-176: ADSP-SC58x WDOG1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x31009000	WDOG1_CTL	WDOG1 Control Register	0x00000AD0
0x31009004	WDOG1_CNT	WDOG1 Count Register	0x00000000
0x31009008	WDOG1_STAT	WDOG1 Watchdog Timer Status Register	0x00000000

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Analog Devices Inc.:

ADZS-SC589-MINI